Elisa Oinonen

Verkkopalvelun dynaaminen käyttöliittymä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
8.5.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Elisa Oinonen Verkkopalvelun dynaaminen käyttöliittymä 46 sivua 8.5.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikan koulutusohjelma
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	operatiivinen johtaja Martti Kuusanmäki lehtori Ilkka Kylmäniemi

Insinöörityön tavoitteena oli selvittää dynaamisen Ajax-pohjaisen verkkopalvelun suunnitteluun ja toteutukseen vaadittavat oleelliset teknologiat ja käytännöt. Työssä kartoitettiin parhaat kehitysmenetelmät ja suunnittelumallit ja tehtiin verkkopalvelun toteutukseen liittyvien haasteiden ratkaisuehdotukset.

Ongelmat ja parhaat suunnittelukäytännöt dynaamisten käyttöliittymien kehityksessä selvitettiin laajalla kirjallisella tutkimuksella. Selvisi, että tärkeimpänä osana dynaamisen verkkopalvelun kehitystä ovat erilaiset kehityskirjastot ja -työkalut, koodin huolellinen testaus, palvelun suorituskyvyn optimointi ja oikeat suunnittelumallit.

Dynaamisten käyttöliittymien suurimpana ongelmana on käytettävyys. Käytettävyyden parantamiseksi verkkosivun on toteutustavasta riippumatta toimittava aivan tavallisen verkkosivun tavoin ja käyttäjää on informoitava virhetilanteiden sattuessa. Koodimäärän kasvaessa on erityisen tärkeää pitää huoli, että sivun latausajat eivät kasva liian suuriksi.

Ajax-pohjaisiin verkkopalveluihin liittyy vielä joitakin ongelmia, joille ei löydy täydellistä ratkaisua. Insinöörityössä koottuja suunnitteluohjeita seuraamalla dynaamisten verkkopalveluiden toteutus on kuitenkin aiempaa tehokkaampaa ja palveluista saadaan nopeampia ja luotettavampia.

Insinöörityönä toteutettiin JobSqr-niminen dynaaminen työnhakupalvelu. Palvelun toteutuksessa hyödynnettiin erilaisia dynaamisuutta lisääviä kirjastoja ja yhdisteltiin työnhaun helpottamiseksi eri ohjelmointirajapintoja, kuten Google Maps -karttapalvelu ja Facebook API. Työssä kehitettyjä komponentteja käytetään hyväksi myös tilaajayrityksen muissa palveluissa.

Avainsanat	verkkopalvelu, käyttöliittymä, Ajax, JavaScript

Metropolia

Abstract

Author Title Number of Pages Date	Elisa Oinonen Dynamic web application 46 pages 8 May 2012
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital media
Instructors	Martti Kuusanmäki, Chief Operating Officer Ilkka Kylmäniemi, Senior Lecturer

The purpose of the project was to go through the process of creating dynamic Ajax based web applications, including required technologies and best design practices. Wide literary study was used to map out the problems and the best design practices in dynamic web application development.

The study showed that JavaScript frameworks and libraries are an integral part of productive user interface development. Furthermore, thorough testing and performance optimization is very important as well as choosing the right design pattern for implementing the application.

Usability is one of the biggest challenges of dynamic web application development. The application should operate like a traditional web page, despite of distinct implementation methods. Users should be informed of any errors and the page load time needs to be kept down, even though the amount of code grows.

There are still some issues with Ajax based applications that do not have a perfect solution. However, by following the design instructions explained in this bachelor's thesis the development of dynamic web applications can be more effective and the applications will be faster and more reliable.

Dynamic job search platform called JobSqr was created as a result of the final year project. Lots of different libraries were used to increase the efficiency of the development process and the usability of the application. Application programming interfaces such as Google Maps and Facebook API were used in order to facilitate job search.

Keywords	web application, user interface, Ajax, JavaScript

Sisällys

1 Johdanto

Insinöörityön tavoitteena on suunnitella ja toteuttaa dynaaminen käyttöliittymä uudelle karttapohjaiselle työnhakupalvelulle. Työssä tutkitaan ja esitetään parhaat ratkaisut dynaamisen Ajax-pohjaisen käyttöliittymän toteutukseen ja käydään läpi siihen liittyvät haasteet ja ratkaisumahdollisuudet.

Projekti toteutetaan Skyhood Oy:lle, joka on työnhakumarkkinoille keskittynyt internet-palveluja kehittävä yritys. Järjestelmän suunnitteluun ja toteutukseen osallistuu kaiken kaikkiaan kuusi henkilöä. Insinöörityön tilaajayritys vastaa palvelun ulkonäön suunnittelusta ja asettaa vaatimukset käyttöliittymän toteutukselle, kun taas insinöörityön tekijä on vastuussa käyttöliittymän teknisestä suunnittelusta ja toteutuksesta. Järjestelmän palvelinpuolen kehitys toteutetaan Python-ohjelmointikielellä, eikä sitä käsitellä insinöörityöraportissa.

Käyttöliittymän toteutuksen ensisijaiseksi vaatimukseksi asetettiin helppokäyttöisyys ja nopeus. Palvelun tulee toimia laajasti käyttöjärjestelmästä ja selaimesta riippumatta, ilman selaimeen asennettavia lisäosia. Näiden vaatimusten vuoksi käyttöliittymä toteutetaan Ajax-tekniikalla, joka yhdistelee perinteisiä internetkehityksessä käytettäviä tekniikoita ja toimii kaikissa yleisissä käyttöjärjestelmissä ja selaimissa.

Insinöörityön tuloksena julkaistaan JobSqr-työnhakupalvelun beta-versio. Projektissa on uuden työnhakupalvelun toteutuksen ohella tarkoitus kehittää dynaamisia komponentteja ja toimivia ratkaisumalleja, joita voidaan käyttää uudelleen myös tilaajayrityksen muissa palveluissa.

2 Dynaamiset käyttöliittymät

2.1 Verkkosivujen historia

Aluksi kaikki verkkosivut olivat täysin staattisia, eivätkä ne sisältäneet lainkaan toiminnallisuuksia. Sivustojen staattisuutta ei edes pidetty ongelmana, sillä internetiä käytettiin ainoastaan tutkijoiden ja yliopistojen tutkimusten ja tietojen vaihtoon. Henkilökohtaisten tietokoneiden suosion ja uusien työpöytäohjelmistojen myötä myös odotukset verkkosivuja kohtaan kasvoivat. [1, s. 3–4.]

Ensimmäinen ratkaisu dynaamisten verkkosivujen toteuttamiseen oli CGI. Sen käyttö on kuitenkin suuri tietoturvariski, sillä se sallii käyttäjän suorittaa muita ohjelmia järjestelmien sisällä. CGI-tekniikkaa seurasivat vuonna 1995 markkinoille tuodut kehittyneemmät Java-ohjelmointikielellä luodut sovelmat. Kun tuolloin markkinoita hallinnut Netscape Navigator -selain tarjosi tuen Java-sovelmille, alkoi sovelmien valtakausi. Sovelmien ongelmana oli kuitenkin se, että ne vaativat toimiakseen aina tietyn Java-version. Huonosti laaditut sovelmat aiheuttivat myös käyttäjien koneissa ongelmia, ja niiden suosio laantui. [1, s. 4–5.]

Netscape kehitti JavaScript-ohjelmointikielen samoihin aikoihin, kun Java luotiin. JavaScript kehitettiin parantamaan verkkosivujen käyttökokemusta mahdollistamalla HTML-elementtien muokkaaminen dynaamisesti. JavaScript ei kuitenkaan saavuttanut suosiota nopeasti, sillä kieltä varjostivat pitkään ongelmat selaintuessa, turvallisuudessa, testaamisessa ja virheiden jäljittämisessä. Nykyisinkin vielä jotkin JavaScript-koodit toimivat eri tavoin selaimesta riippuen, mutta suurin osa ongelmista on voitettu kattavien sovelluskehysten ja -kirjastojen ansiosta. JavaScriptin suuri suosio alkoi vasta Ajax-sovellusten yleistyttyä. [1, s. 6–13; 2, s. 2–4.]

Ajax on yhdistelmä tekniikoita, jotka yhdessä mahdollistavat dynaamisten käyttöliittymien toteutuksen käyttäen hyväksi asynkronisia HTTP-kutsuja. Ajax ei ole uusi keksintö, sillä sen mahdollistava teknologia on vuodelta 1999. Kuitenkin selainten kehittyminen tukemaan Ajax-teknologiaa on vienyt aikaa. Ensimmäisen kerran termiä AJAX käytti Jesse James Garret vuonna 2005. Tämän jälkeen tekniikan käyttö on yleistynyt nopeasti. Nykyisin käyttäjät vaativat monipuolisia ja helppokäyttöisiä verkkosovelluksia

ilman ylimääräisten asennuspakettien tarvetta. Ajax-teknologian avulla näihin vaatimuksiin pystytään vastaamaan. Ajax on tällä hetkellä merkittävin vaihtoehto dynaamisten työpöytäsovelluksia muistuttavien verkkosovellusten toteuttamiseen. [1, s. 14–16; 2, s. 2; 3.]

2.2 Käyttöliittymien tulevaisuus

Verkkosovellukset ovat kehittyneet huimasti, mutta ne ovat edelleen monien mielestä käyttökokemukseltaan heikompia kuin tavalliset työpöytäsovellukset. Ero verkkosovellusten ja työpöytäsovellusten välillä on kuitenkin kutistumassa. Nopeus on suurin työpöytäsovellusten etu, joka verkkosovellusten on vielä saavutettava. Vaikka Ajax-teknologian ansiosta palvelut ovat huomattavasti perinteisiä nopeampia, ei kutsujen käsittely kuitenkaan tapahdu käyttäjän huomaamatta. Ajax-sovelluksissa on yleisesti käytössä latauskuvakkeet, jotka ilmaisevat päivityksen olevan käynnissä ja käyttäjän tulee odottaa vastausta. [1, s. 13; 3; 4.]

Ratkaisuksi verkkopalveluiden hitauteen Alex MacCaw tarjoaa ajattelutavan muutosta perinteisestä pyyntö–vastaus-mallista. Tekniikkaa kutsutaan nimellä Asynchronous User Interfaces (AUI), ja sen perusajatuksena on kutsujen erottaminen käyttöliittymästä. AUI:ssä käyttöliittymä päivitetään jo ennen varsinaisten kutsujen lähetystä, jolloin käyttäjä näkee muutoksen tapahtuvan välittömästi. Menetelmä lupaa uutta suuntaa käyttöliittymäkehitykseen, mutta se ei kuitenkaan sovellu kaikkiin tilanteisiin, kuten verkkomaksuihin ja sivustoille, joille varsinainen sisältö ladataan Ajax-pyyntöjen perusteella. [4.]

Verkkopalveluilla on kuitenkin huomattavia etuja työpöytäsovelluksiin nähden. Verkkopalvelut eivät ole riippuvaisia päätelaitteesta tai paikasta, ne tehostavat yhteistyötä ja mahdollistavat sovellusten kytkemisen osaksi sosiaalista mediaa. Uudet tekniikat, kuten HTML5, CSS3 ja WebGL, tarjoavat runsaasti lisää mahdollisuuksia palveluiden kehitykseen. Näiden uusien teknologioiden ansiosta voidaan luoda yhä tehokkaampia, monipuolisempia ja käyttäjäystävällisempiä sovelluksia. Nähtäväksi jää, milloin verkkopalvelut syrjäyttävät työpöytäsovellukset. [4; 5.]

2.3 Dynaamisen käyttöliittymän tekniikat

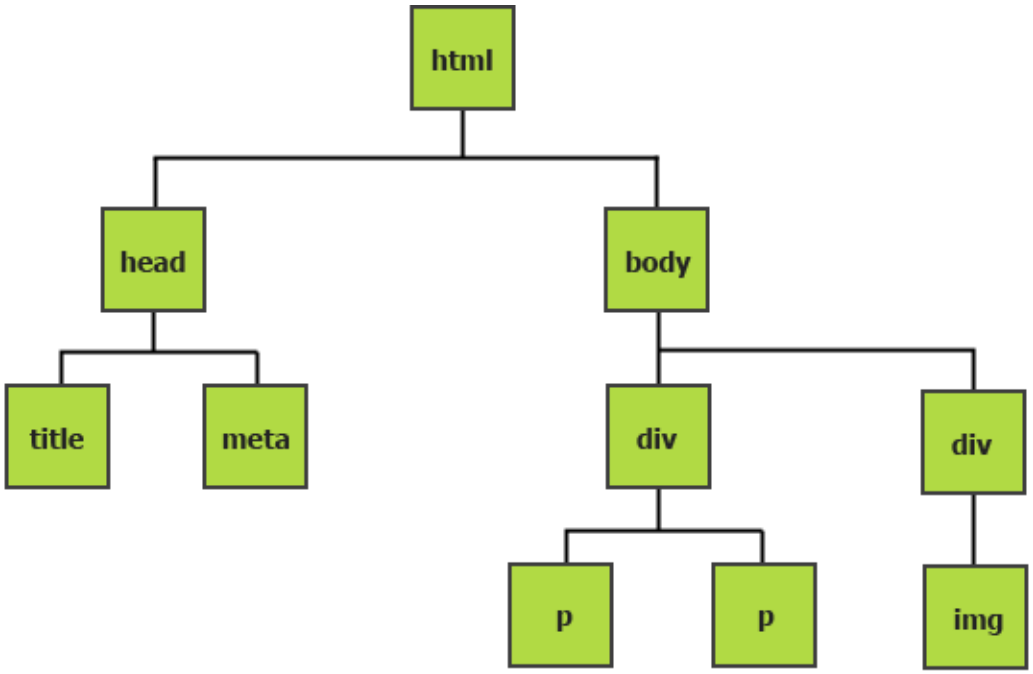
2.3.1 HTML-merkintäkieli ja CSS-tyylimäärittelyt

HTML ja CSS ovat verkkosivun toteutuksen perustekniikoita. HTML on kuvauskieli, jolla verkkosivut rakennetaan. HTML:llä voidaan kuvata hypertekstiä ja tekstin rakennetta elementtien avulla. XHML on HTML:stä kehitetty tiukemmat säännöt sisältävä versio, joka täyttää XML:n muotovaatimukset. XHTML-määrittelyssä muun muassa tunnisteet tulee kirjoittaa pienillä kirjaimilla ja elementit tulee sulkea. [6; 7.] XHTML-dokumentin perusrakenne on havainnollistettu esimerkkikoodissa 1.

```
<!DOCTYPE html>
<html>
  <head>
      <title>Dokumentin otsikko</title>
      <meta name="description" content="Sisällön kuvaus" />
  </head>
  <body>
      <h1>Tervetuloa!</h1>
      <div>
        <p>Tässä havainnollistetaan XHTML-dokumentin rakenne</p>
        <p>Toinen tekstikappale</p>
      </div>
      <div><img alt="kuva" src="kuva.png" /></div>
  </body>
</html>
```

Esimerkkikoodi 1. XHTML-dokumentin perusrakenne [7].

Yksi keskeisimmistä asioista dynaamisia verkkosivuja luotaessa on XHTML-dokumenttien sisällön muokkaaminen käyttäen DOM-ohjelmointirajapintaa JavaScript-koodin avulla. Tällöin verkkosivua ei tarvitse ladata kokonaan uudestaan, vaan XHTML-dokumentista voidaan muokata vain halutut kohdat. DOM-mallissa XHTML-dokumentin data esitetään puumaisena tietorakenteena, kuten kuvassa 1 esitetään. Ilman DOM-ohjelmointirajapintaa dynaamisten käyttöliittymien toteuttaminen olisi mahdotonta. [8; 9.]



Kuva 1. XHTML-dokumentti DOM-tietorakenteena [8].

HTML5 on HTML:n viides versio, ja se sisältää monia uudistuksia. Myös HTML5 tukee XHTML-syntaksia, ja DOM on olennainen osa HTML5-määritelmää. HTML5:tä kehitetään edelleen, ja se oletetaan standardoitavan vuonna 2020. Uusia HTML5-elementtejä ovat muun muassa canvas-, video- ja audio-elementit. Varsinkin canvas-elementti on mielenkiintoinen lisä, sillä se mahdollistaa grafiikan tekemisen verkkosivulle dynaamisesti. Vaikka HTML5 ei vielä ole standardoitu eikä kaikkialla tuettu, sitä voidaan jo hyödyntää verkkopalveluissa. Sivustolle voidaan lisätä erillinen JavaScript-tiedosto, jonka ansiosta osa HTML5-elementeistä ja ominaisuuksista toimii myös vanhemmissa selaimissa. [10.]

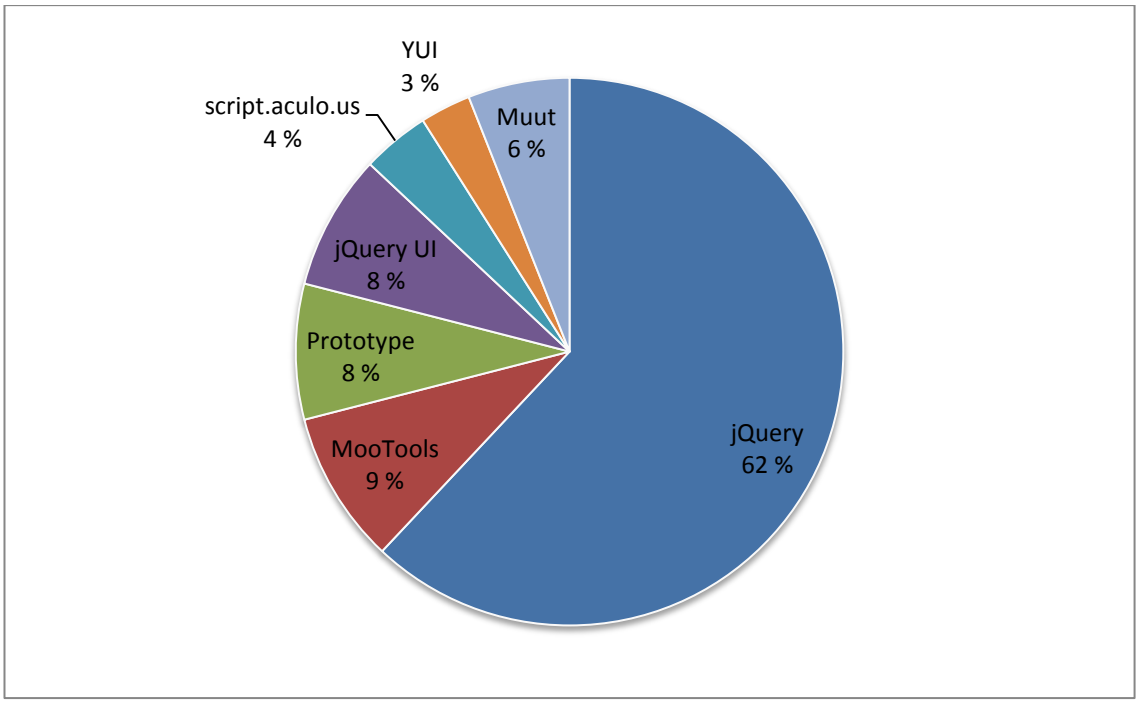
CSS-tyyliohjeet määrittelevät, kuinka data esitetään verkkosivuilla. Myös CSS sallii sen tyyliohjeiden muokkaamisen dynaamisesti. CSS onkin yksi parhaita tapoja muokata verkkosivuja dynaamisesti, sillä se mahdollistaa muutosten nopean havainnollistamisen käyttäjälle. CSS:n avulla käyttäjille voidaan helposti korostaa verkkosivun toiminnalliset ja päivitetyt elementit. Tämän vuoksi se on eräs tärkeimmistä rakennuspalikoista dynaamisia Ajax-verkkosovelluksia luodessa. CSS3 lisää nykyisen CSS2-määritelmän päälle uusia käytettävyyttä ja ulkoasua parantavia ominaisuuksia, kuten siirtymät, varjostukset, reunojen pyöristykset ja tärkeimpänä Media Queryt, joiden avulla voidaan dynaamisesti muokata sivun ulkoasua selainikkunan koon mukaan. [6; 11.]

2.3.2 JavaScript-ohjelmointikieli

JavaScript on prototyyppeihin perustuva oliopohjainen komentosarjakieli, jota käytetään pääasiassa verkkosivuilla lisäämään niiden dynaamisuutta. JavaScript-koodi sisällytetään HTML-dokumenttiin script-elementtien sisään joko viittaamalla erilliseen tiedostoon tai asettamalla koodi suoraan elementin sisällöksi. Koodi suoritetaan käyttäjän selaimessa, jolloin se ei vaadi verkkosivun päivittämistä ja verkkosivu pystyy vastaamaan käyttäjän pyyntöihin entistä nopeammin. JavaScript pystyy tunnistamaan ja reagoimaan erilaisiin käyttäjän tekemiin toimintoihin, kuten näppäinten painamiseen, mikä mahdollistaa työpöytäsovellusten kaltaisten verkkopalveluiden luomisen. [9, s. 2–6; 12.]

JavaScript on tärkein Ajaxin mahdollistava tekijä, sillä se hallitsee Ajax-pyynnöt ja HTML ja CSS -koodien päivityksen. Ajaxin myötä JavaScriptistä on tullut yksi internetin suosituimmista ohjelmointikielistä. JavaScriptin kasvaneen suosion vuoksi sen ympärille on kehittynyt suuri määrä kattavia ohjelmistokehyksiä ja -kirjastoja, jotka ovat parantaneet JavaScriptin ohjelmointikäytäntöjä ja tehneet sen käytöstä helpompaa ja tehokkaampaa. JavaScript-kirjastojen hyödyntäminen on merkki viisaasta kehittäjästä, sillä niiden käyttö säästää paljon aikaa. Ei ole järkevää käyttää aikaa tekemällä uudelleen asioita, jotka on jo tehty ammattimaisesti ja testattu huolellisesti. Suurin osa kirjastoista on avoimen lähdekoodin kirjastoja eli saatavilla ilmaiseksi mihin tahansa käyttöön. Luvussa 3.3 verrataan Ajax-pyynnön vaatimaa koodia ilman lisäkirjastoja ja jQuery-kirjaston kanssa. [1, s. 14; 13, s. 6–8; 14.]

JavaScript-kirjastoa valittaessa on syytä kiinnittää huomiota toivottuihin toiminnallisuuksiin, selainten yhteensopivuuteen, kirjaston kokoon ja erityisesti sen luotettavuuteen. Luotettavuudesta kertovat etenkin se, kuinka aktiivisesti kirjastoa päivitetään ja kuinka suuri käyttäjäkunta sillä on. Juuri käyttäjäkunta ja sen tyytyväisyys takaavat kirjaston aktiivisen kehittämisen myös tulevaisuudessa. [15; 16.] Kuten kuva 2 havainnollistaa, suosituimmat JavaScript kirjastot ovat jQuery, MooTools, Prototype, jQuery UI, script.aculous ja YUI.



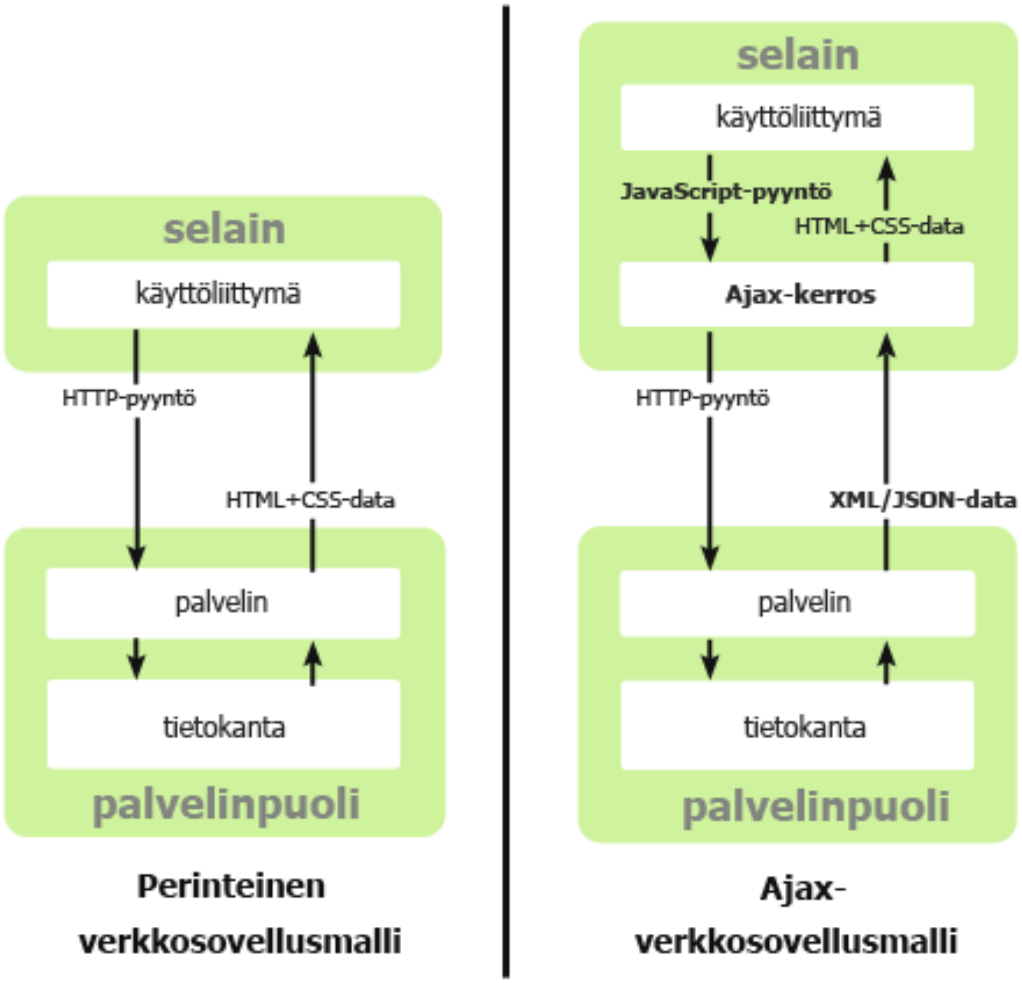
Kuva 2. JavaScript-kirjastojen markkinaosuudet [16].

2.3.3 Ajax-teknologia

Ajax on lyhenne sanoista Asynchronous JavaScript and XML, ja se tarkoittaa verkkosivujen sovelluskehitykseen käytettävien teknologioiden yhdistelmää, joiden avulla voidaan luoda dynaamisia verkkosivuja. Ajax-teknologioista puhuttaessa tarkoitetaan muun muassa XHTML-, DOM-, CSS-, XML-, JSON- ja JavaScript-tekniikoita ja erityisesti JavaScriptin XMLHttpRequest-objektin käyttöä. Ajax ei siis ole yksittäinen tekniikka, vaan se viittaa useiden tekniikoiden käyttöön yhdessä. [1, s. 16; 3.]

Perinteisessä verkkosivujen tiedonsiirtomallissa selain lähettää HTTP-pyynnön palvelimelle, joka vastaa lähettämällä sopivan vastauksen, kuten HTML-sivun. Jokainen HTTP-pyyntö palvelimelle aiheuttaa koko sivun uudelleenlatauksen, jolloin käyttäjä joutuu odottamaan sivun latautumista, ennen kuin työskentelyä voi jatkaa. Ajax mahdollistaa asynkronisten eli ei-reaaliaikaisten pyyntöjen lähettämisen ja tietojen vastaanottamisen palvelimelta ilman, että koko verkkosivua tarvitsisi ladata uudelleen. Asynkronisten kutsujen avulla voidaan siis päivittää vain osa verkkosivusta, jolloin käyttäjä voi jatkaa työskentelyä ilman keskeytyksiä palvelimen suorittaessa pyyntöä. Kun palvelin on saanut suoritettua pyynnön, se palauttaa vastauksen selaimelle, joka päivittää verkkosivun sisällön dynaamisesti. [1, s. 17; 3.]

Ajax-malli muuttaa perinteistä verkkosovellusmallia siten, että se lisää Ajax-kerroksen asiakkaan ja palvelimen välille. Ajax-kerros huolehtii tiedonsiirrosta palvelimen kanssa ja käyttöliittymän päivittämisestä, kuten kuvassa 3 esitetään. Tiedonsiirto perustuu JavaScriptin XMLHttpRequest- eli XHR-objektiin. Kun käyttäjä laukaisee tapahtuman, joka vaatii dataa palvelimelta, XHR-objekti lähettää palvelimelle HTTP-pyynnön, joka prosessoi tiedon ja palauttaa vastauksen takaisin Ajax-kerrokselle. Tämän jälkeen Ajax-kerros päivittää käyttöliittymän vastaanotettujen tietojen mukaisesti ja muokkaa DOM-elementtejä ja CSS-tyyliohjeita. [3.]



Kuva 3. Perinteinen verkkosovellusmalli ja Ajax-malli [3].

Ajax-pyyntö muodostetaan luomalla ensin viittaus käytettävään XHR-objektiin. Käytettävä objekti on lähes kaikissa selaimissa XMLHttpRequest. Ainoastaan Microsoftin Internet Explorer -selaimissa XHR-objekti on nimeltään ActiveXObject. Kun viittaus XHR-objektiin on tehty, kutsutaan open()-funktiota, jolle asetetaan parametreiksi HTTP-pyynnön metodi, kuten ”GET” tai ”POST” ja URL-osoite, johon pyyntö lähetetään. Kol-

mantena parametrina voidaan asettaa boolean- eli totuusarvo siitä, tehdäänkö kutsu asynkronisena vai ei. Viimeisen arvon asettaminen ei kuitenkaan ole pakollinen, sillä oletuksena kutsu on aina asynkroninen. XHR-objektille asetetaan myös onreadystatechange-tapahtumankäsittelijä, joka aktivoituu XHR-objektin tilan muuttuessa. Lopuksi lähetetään varsinainen pyyntö send()-funktion avulla. [1, s. 30; 9, s. 363–366]

Kun XHR-objektin readyState-ominaisuuden tila on 4 ja objektin ”status”-tila on 200, tiedetään, että pyyntö on suoritettu onnistuneesti, ja tämän jälkeen voidaan suorittaa tarvittavat paluuarvojen käsittelyt. Kuten esimerkkikoodista 2 käy ilmi, käyttämällä esimerkiksi JavaScriptin jQuery-kirjastoa helpottuu Ajax-pyynnön tekeminen huomattavasti. [1, s. 31; 9, s. 367.]

```
//Ajax-pyyntö ilman kirjastoja
function getData(dataSource, divId){
  var XMLHttpRequestObject = false;
  if(window.XMLHttpRequest){
    XMLHttpRequestObject = new XMLHttpRequest();
  }else if(window.ActiveXObject){
    XMLHttpRequestObject=new ActiveXObject('Microsoft.XMLHTTP');
  }

  if(XMLHttpRequestObject){
    XMLHttpRequestObject.open('GET', dataSource);

    XMLHttpRequestObject.onreadystatechange = function(){
      if(XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200){
          var obj = document.getElementById(divID);
          obj.innerHtml = XMLHttpRequestObject.responseText;
      }
    }
    XMLHttpRequestObject.send();
  }
}
```

```
// Ajax-pyyntö jQuery-kirjaston avulla
function getDatajQuery(dataSource, divId){
  $.get(dataSource, function(data){
    $('#' + divId).html(data);
  });
}
```

Esimerkkikoodi 2. Ajax-pyyntö ilman kirjastoja ja jQuery-kirjastoa käyttäen [13, s. 85; 17].

Informaation välitys Ajax-pyynnössä tapahtuu XML-, JSON-, HTML- tai tekstimuodossa. XML on tekstimuotoinen yleiskäyttöinen merkintäkieli, jota käytetään yleisesti datansiirrossa. XML-data on raakamuotoista, eikä sitä voi suoraan näyttää sivustolla, vaan data tulee jäsentää ja käsitellä ennen käyttöliittymään päivittämistä. XML:n tavoin JSON on yksinkertainen tiedonsiirtomuoto, joka on yleisesti käytössä Ajax-sovelluksissa vaihtoehtona XML-formaatille. JSON pohjautuu JavaScript-objektien merkintätapaan, mutta se on kuitenkin kielestä riippumaton. Koska JSON-formatoitu teksti on useimmiten myös oikeamuotoista JavaScriptiä, voidaan JavaScriptiin sisäänrakennettua eval()-funktiota helposti käyttää JSON-muotoisen datan jäsentämiseen. HTML- ja tekstimuotoista tiedonsiirtoa käytetään yleensä silloin, kun halutaan päivittää Ajax-pyynnön vastaus suoraan sivustolle. [18, s. 4–5.]

3 Käyttöliittymän kehityksen parhaat suunnittelukäytännöt

3.1 Kehitysmenetelmät

Perussääntönä dynaamisten sivustojen kehittämisessä on, että tyylimäärittelyjen lisäksi myös toiminnallisuus erotetaan sisällöstä. Sen sijaan, että määrittelyt asetettaisiin suoraan sisällön joukkoon, CSS- ja JavaScript-tiedostot liitetään erillisinä tiedostoina XHTML-dokumenttiin. Elementtien tyyli- ja toiminnallisuusmäärittelyjä hallitaan dynaamisesti DOM-ohjelmointirajapinnan välityksellä, jolloin elementit voidaan tunnistaa niiden tyypin tai id- ja class-attribuuttien avulla. Kuten aikaisemmin luvussa 2.3.2 on mainittu, kattavien JavaScript-kirjastojen käyttö on erittäin suositeltavaa. Kirjastot eivät tarjoa ainoastaan suurta määrää valmiita toiminnallisuuksia, vaan myös takaavat laajan selaintuen ilman lisätyötä. [19; 20.]

Koodin kommentointi on tärkeä asia muistaa myös JavaScript-koodia kirjoitettaessa. Kommentteja ei tarvitse eikä kannata lisätä joka riville, vaan esimerkiksi selventämään, mitä jokin funktio tekee ja miksi se tekee juuri niin. Kun koodia on kirjoitettuna useita tuhansia rivejä, on yllättävän vaikea muistaa, mitä jokin funktio tekee, mikäli sitä ole lainkaan kommentoitu. Kommentointi on erityisen tärkeää, mikäli projektin parissa työskentelee tai tulee tulevaisuudessa työskentelemään useita eri henkilöitä. Voi olla niinkin, että sovelluksen pääkehittäjä joutuu jättämään työnsä, minkä jälkeen muiden on erittäin hankalaa jatkaa kehittämistä, jos koodia ei ole kunnollisesti kommentoitu. Vaikka olisikin varmaa, että kirjoitettua koodia ei kukaan muu kehittäjän lisäksi koskaan lue, on kommentointi hyvä tapa myös oman muistin virkistämiseksi. [18, s. 272–275.]

Ajax-sovelluksissa on tärkeää informoida käyttäjää, mikäli jokin pyyntö tai tapahtuma ei jostain syystä onnistunut. Käyttäjät mieluummin tiedostavat tapahtuneen virheen kuin odottavat palvelua, joka ei vastaa. Useimmat JavaScript-kirjastot tarjoavat valmiita tapahtumankäsittelyfunktioita, jotka reagoivat eri tilanteissa. Esimerkiksi jQueryn ajax()-metodi sisältää kolme eri tilanteissa reagoivaa tapahtumankäsittelyfunktiota, jotka on esitetty esimerkkikoodissa 3. Virheiden hallinnassa näistä tärkein on error()-funktio, jota voidaan käyttää muun muassa virheen esittämiseen käyttöliittymässä. Käyttäjää on myös informoitava, mikäli palvelu ei jostain syystä tue käytössä olevaa

selainta. Selaintarkistukseen on olemassa useita valmiita kirjastoja ja liitännäisiä, joiden avulla tarkistus on helppo lisätä palveluun. [20.]

```
$.ajax({
  // ...
  success: function(msg){
    // pyyntö onnistui
    showData();
  },
  error: function(msg){
    // pyyntö epäonnistui
    showError();
  },
  complete: function(){
    //pyyntö suoritettu
    showStatus();
  }
});
```

Esimerkkikoodi 3. jQueryn ajax()-metodin tapahtumankäsittelyfunktiot.

Mikäli palvelu reagoi käyttäjän toimiin aktiivisesti ja lähettää jatkuvasti uusia Ajax-pyyntöjä, voi niiden samanaikainen määrä kasvaa suureksi. Tämän seurauksena palvelu usein hidastuu ja käyttäjä havaitsee käyttöliittymässä niin sanottua tökkimistä, kun Ajax-pyyntö toisensa jälkeen päivittää käyttöliittymää. Tällaiset kutsut tuleekin hallita järkevästi, eikä käynnissä saisi olla kuin yksi samantyyppinen kutsu kerrallaan. Käynnissä oleva Ajax-pyyntö voidaan keskeyttää käyttämällä abort()-metodia, jonka jälkeen taas voidaan lähettää uusi kutsu. Hidastumista voidaan myös havaita, mikäli käyttöliittymään tehtäviä päivityksiä ei tehdä viisaasti. Sen sijaan, että päivitetään jokainen elementti yksitellen, on suositeltavaa päivittää suurempi osuus kerrallaan. Suurien HTML-kokonaisuuksien päivittämiseen kannattaa käyttää erilaisia mallifunktioita, joiden toteuttamiseen on myös useita helppokäyttöisiä kirjastoja. Mikäli tarkoituksena on käydä läpi suuria datamääriä, paras ja nopein vaihtoehto on Mustache-kirjasto. [21; 22.]

3.2 Suunnittelumallit ja nimiavaruus

Dynaamisia käyttöliittymiä toteutettaessa koodimäärä kasvaa usein suureksi. Mikäli koodilla ei ole minkäänlaista rakennetta, on kehittäjien äärimmäisen vaikea hallinnoida sitä. Kun globaaleja muuttujia on suuri määrä, voi myös sattua niin, että uusi muuttuja nimetään samoin kuin jo olemassa oleva, jolloin aiempi osa koodista rikkoutuu. Näiden nimitörmäysten estämiseksi ohjelmoinnissa käytetään nimiavaruuksia, joiden sisällä muuttujat määritellään. Yksinkertaisin esimerkki nimiavaruuden käytöstä on suora määritys. Esimerkkikoodissa 4 havainnollistetaan, kuinka muuttujien ”id” ja ”status” nimiavaruudeksi määritetään ”obj”. Nimiavaruudet auttavat myös koodin järjestämisessä helposti hallittaviksi loogisiksi kokonaisuuksiksi. [23.]

```
var obj = {};
obj.id = 1;
obj.status = 'OK';
```

Esimerkkikoodi 4. Nimiavaruuden suora määritys.

Mahdollisuus käyttää sisäkkäisiä nimiavaruuksia tarjoaa edellytyksen koodin järjestäytyneen hierarkian toteuttamiseen. Sisäkkäisiä nimiavaruuksia voidaan toteuttaa objektimuotoisella merkintätavalla, jossa on sarja pilkulla erotettuja nimi/arvo-pareja hakasulkeiden sisällä, kuten esimerkkikoodi 5 havainnollistaa. Ongelmana objektimuotoisessa merkintätavassa on se, että ne usein kasvavat pitkiksi syntaktisiksi rakennelmiksi, joiden muuttaminen syntaksia rikkomatta voi olla haasteellista. Objektimuotoista merkintätapaa on suositeltavaa käyttää, mikäli nimi/arvo-syntaksia tarvitaan. Parhaiten merkintätapa soveltuu esimerkiksi sovelluksen tai liitännäisen perusasetusten määrittämiseen. [23; 24; 25.]

```
var config = {
  language: 'finnish',
  theme: {
    skin: 'blue',
    toolbars: {
      index: 'default-toolbar',
      pages: 'custom-toolbar'
    }
  }
}
```

Esimerkkikoodi 5. Sisäkkäisten nimiavaruuksien objektimuotoinen esitystapa.

Monimutkaiset palvelut ja sovellukset tulisi toteuttaa käyttäen modulaarisia komponentteja. Moduuleja tulisi käsitellä itsenäisinä yksikköinä, jotka toimivat muista yksiköistä riippumatta. JavaScriptissä muuttujia ei voi määritellä suoraan yksityisiksi tai julkisiksi, joten näkyvyystasoja pyritään simuloimaan funktioiden avulla, jotka toimivat oletuksena niin kutsuttuina sulkeumina. Modulaarinen malli tarjoaa sisäkkäisten nimiavaruuksien lisäksi menetelmän julkisten ja yksityisten metodien hallintaan sulkeumien avulla. Modulaarinen sovellusarkkitehtuuri on turvallinen, helposti mukautuva ja skaalautuva. [24; 25.]

Modulaarinen malli palauttaa sovelluksen julkisen ohjelmointirajapinnan ja vastaavasti estää yksityisten metodien ja muuttujien vuotamisen globaaliin näkyvyysalueeseen. Mallin toimintaa havainnollistaa esimerkkikoodi 6, jossa objektin ”id”- ja ”status”-arvot palautetaan julkisiksi käyttäen return-metodia, kun taas ”arr”-taulukko pysyy yksityisenä. Modulaarisen mallin syntaksi on hyvin samankaltainen kuin Immediately-Invoked Function Expression eli IIFE-syntaksi. Erona näiden välillä on kuitenkin se, että kun IIFE:n palauttaa funktion, modulaarinen malli palauttaa objektin. [23; 25.]

```
var obj = (function(){
  //yksityinen
  var id = 0;
  var status = 'OK';
  var arr = [];

  return { // julkinen
    getId: function(){
      return id;
    },
    getStatus: function(){
      return status;
    }
  }
}());
```

Esimerkkikoodi 6. Modulaarien suunnittelumalli.

Jo parin vuoden ajan on ollut kehitteillä CommonJS-niminen ohjelmointirajapinta, jonka tavoitteena on toimia rikkaana standardoituna JavaScript-kirjastona. CommonJS-ohjelmointirajapinnan tarkoituksena on myös tehdä kehittäjille mahdolliseksi kirjoittaa sovelluksia, joita voidaan suoraan suorittaa JavaScript-tulkista ja ympäristöstä riippumatta. Myös CommonJS perustuu moduulipohjaiseen suunnittelumalliin. Kuten esimerkkikoodi 7 havainnollistaa, CommonJS-moduuleilla on kaksi pääkomponenttia: exports-objekti ja require-funktio. Exports-objekti sisältää ne objektit, jotka moduuli antaa muiden saataville, kun taas require-funktio määrittelee muiden moduulien export-objektit, joita halutaan käyttää. CommonJS-formaatissa olevien moduulien lataaminen vaatii erillisen JavaScript-kirjaston, joita on jo tälläkin hetkellä useita, esimerkkeinä RequireJS ja node.js. CommonJS-moduulien rakenne on melko virheetön, joten se on varteenotettava vaihtoehto perinteiselle modulaariselle mallille. [25.]

```
//speak.js
exports.hello = function(){
  return 'Hello World!';
}

//program.js
var hello = require('speak.js').hello;
hello(); // Hello World!
```

Esimerkkikoodi 7. Yksinkertaiset CommonJS-esimerkkimoduulit.

Funktiot määritellään edellä mainituissa suunnittelumalleissa käyttäen ilmaisumetodia. Funktio-objekti voidaan kuitenkin luoda myös käyttämällä ”new”-operaattoria. Tätä funktion konstruktiometodia tulee kuitenkin välttää, sillä se hidastaa koodin suoritusta. Ilmaisumetodissa funktion määritelmä jäsennetään ainoastaan kerran, toisin kuin konstruktiometodissa, jossa määritelmä jäsennetään joka kerta, kun funktiota kutsutaan. Funktion konstruktiometodi ei kuitenkaan ole täysin käyttökelvoton, vaan sopivia käyttökohteita sille ovat esimerkiksi erilaiset mallipohja-funktiot, joita käytetään yleensä HTML-koodin dynaamiseen rakentamiseen. [26.]

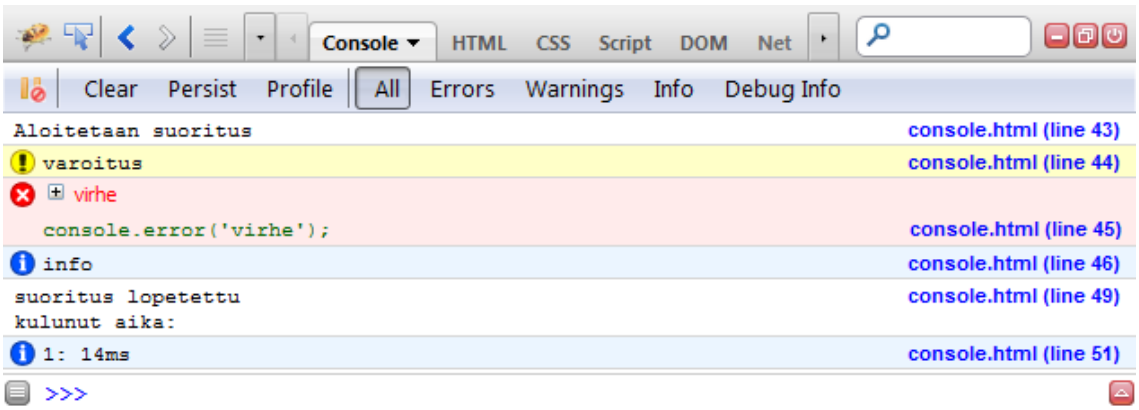
3.3 Kehitystyökalut ja ohjelmointiympäristöt

Jokaisen käyttöliittymäkehittäjän tärkeimpien työkalujen joukossa ovat varmastikin selaimessa olevat kehittäjille suunnatut työkalut. Ne tarjoavat välineet sivuston suunnitteluun, analysointiin ja virheiden jäljitykseen. Tunnetuin työkalu on varmaankin Firebug, joka on Firefox-selaimeen asennettava lisäosa. Firefox-selaimen lisäksi myös muilla selaimilla on omat kehitystyökalunsa. JavaScript-kehitystä ajatellen yksi kehitystyökalujen tärkeimmistä ominaisuuksista on konsoli. Konsolin ominaisuuksia on esimerkiksi automaattinen virheiden, varoitusten ja huomautusten kirjaaminen. Näiden automaattisten viestien lisäksi kehittäjä voi myös itse tallentaa konsoliin esimerkiksi haluamiaan huomautuksia ja muuttujien arvoja käyttäen erityyppisiä JavaScript-komentoja, kuten esimerkkikoodissa 8 esitetään. [18, s. 111–112; 20.]

```
console.time(1);
console.log('Aloitetaan suoritus');
console.warn('varoitus');
console.error('virhe');
console.info('info');
console.log(
  'suoritus lopetettu\n',
  'kulunut aika:'
);
console.timeEnd(1);
```

Esimerkkikoodi 8. Konsoliin tulostavat JavaScript-komennot.

Nämä komennot näkyvät FireBug-konsolissa kuvan 4 havainnollistamalla tavalla.

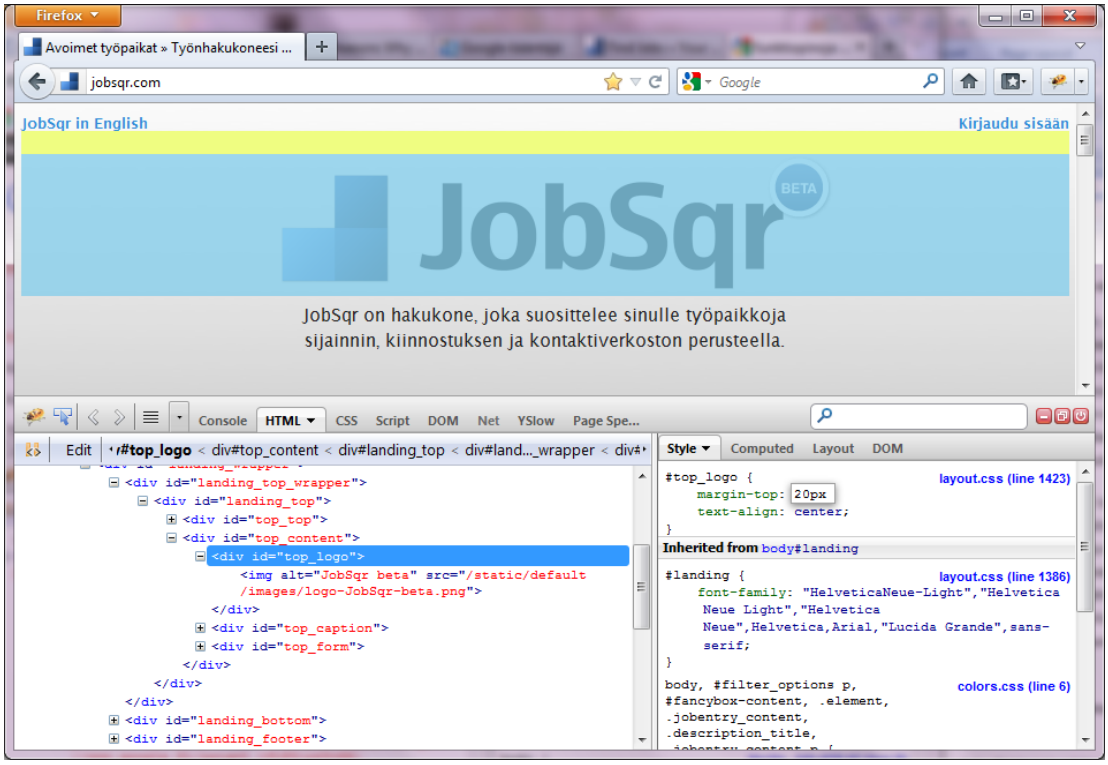


Kuva 4. FireBug-konsolin näkymä ja erityyppiset viestit.

Konsoliin asetettavien viestien lisäksi kehitystyökaluilla on mahdollista asettaa JavaScript-koodiin keskeytyskohtia (breakpoint), joiden ansiosta pystytään tarkastelemaan kullakin hetkellä vallitsevaa nimiavaruutta ja muuttujien arvoja. Tämä ominaisuus helpottaa koodin virheiden jäljitystä huomattavasti, sillä keskeytyskohdasta lähtien koodia voidaan suorittaa selaimessa kohta kohdalta, kunnes ongelmakohta selviää. Keskeytyskohdissa voidaan tarkastella myös funktiopinoja, jotka ovat muodostuneet sisäkkäisten nimiavaruuksien käytöstä. [20.]

JavaScript-työkalujen lisäksi on olemassa erilliset osiot HTML- ja CSS-koodien käsittelyyn. Niiden avulla voi kätevästi tarkastella sivun rakennetta ja ulkoasua sekä muokata niitä, jolloin muutokset näkyvät sivulla välittömästi. Kuten kuva 5 havainnollistaa,

HTML-muokkain korostaa aina valittuna olevat elementit ja mahdollistaa samalla myös niiden CSS-tyylimäärittelyiden muokkauksen. Tämä helpottaa palvelun ulkoasun toteutusta huomattavasti, kun kehittäjä voi nähdä suoraan selaimesta, tuottavatko muutokset halutun lopputuloksen. Kun muutoksiin ollaan tyytyväisiä, ne täytyy kopioida työkalusta lähdekoodiin, sillä muutokset ovat paikallisia ja ne katoavat sivun päivityksen yhteydessä. Kehitystyökalujen käyttö nopeuttaa sivuston toteutusta, kun tarvetta sivun jatkuvaan päivitykseen kehitysvaiheessa ei ole. [20.]



Kuva 5. Sivuston elementtien muokkaaminen käyttämällä Firefox-selaimen Firebug-lisäosaa.

Myös valitulla ohjelmointiympäristöllä on vaikutusta siihen, kuinka tehokkaasti sivusto voidaan toteuttaa. Vaikka verkkosivut pystyykin toteuttamaan pelkän Windowsin muistion avulla, se ei kuitenkaan ole tehokasta eikä kannata. Kunnolliset muokkaimet eli editorit auttavat kehittäjää tarjoamalla koodin täydennystä, valmiita funktioita ja virheentarkistusta. Esimerkkejä hyvistä editoreista käyttöliittymien kehittämiseen Windows- ja Linux-ympäristöissä ovat Aptana, Notepad++ ja Eclipse. Mikäli käyttöjärjestelmänä on Mac OS X, Aptanan lisäksi varteenotettava vaihtoehto on Coda. [20.]

3.4 Testaus

JavaScript-koodin testaus ja virheiden jäljittäminen on ollut vaikeaa, minkä vuoksi monet kehittäjät eivät ole aiemmin halunneet käyttää kieltä lainkaan. Koodin testaukseen on kuitenkin nykyisin olemassa monia eri menetelmiä, joiden suorittamiseen on olemassa todella monia eri kirjastoja. Tästäkin huolimatta vain noin puolet kehittäjistä testaa JavaScriptiä. Koodin järjestelmällinen testaus on kuitenkin tärkeää, jotta kaikki mahdolliset virhetilanteet saadaan kiinni ajoissa, ennen kuin ne päätyvät tuotantoympäristöön. [1, s. 14; 27.]

JavaScriptiä voidaan testata kolmella eri metodilla, jotka ovat yksikkötestaus, käyttäytymistestaus ja toiminnallinen testaus. Yksikkötestauksessa koodi puretaan pienemmiksi osiksi ja testataan yhtä metodia kerrallaan. Tällä tavoin pystytään selvittämään koodin oikeellisuus ja helposti osoittamaan, mikä osa koodista on viallista. Suosituimpia testikirjastoja ovat JSUnit, QUnit ja YUITest. JSUnit on ensimmäinen yksikkötestikirjasto, ja se on vuodelta 2001. Kirjasto on jo kuitenkin hieman vanhentunut, eikä sitä enää päivitetä aktiivisesti. QUnit on uudempi testikirjasto, joka tukee jQuery-kirjastoon perustuvan koodin yksikkötestausta. YUITest taas on vastaava kirjasto YUI-kirjastolle. Perinteisistä yksikkötestikirjastoista poiketen JavaScript-kirjastot tukevat myös asynkronisten funktioiden testausta. [27; 28.]

Käyttäytymistestaus on hyvin samankaltaista kuin yksikkötestaus. Se eroaa yksikkötestauksesta siten, että yksikköjen sijaan testataan kokonaisia tehtäviä, joita koodin on tarkoitus suorittaa. Suosituimpia käyttäytymistestikirjastoja ovat Screw.Unit ja JSSpec. Yksikkö- ja käyttäytymistestit luodaan manuaalisesti. Mikäli sivun toiminnallisuutta halutaan testata tuotantoympäristössä, on testauksen automatisointi tärkeää. Toiminnallinen testaus on tällaista automatisoitua testausta, ja sen avulla voidaan tallentaa ja automatisoida käyttäjän sivustolla tekemät toiminnot. Paras työkalu tällaiseen automatisoituun verkkosivun toiminnan testaukseen on Selenium. Dynaamisia verkkopalveluita toteutettaessa varsinkin JavaScriptin yksikkö- ja toiminnallinen testaus on tärkeää. [27.]

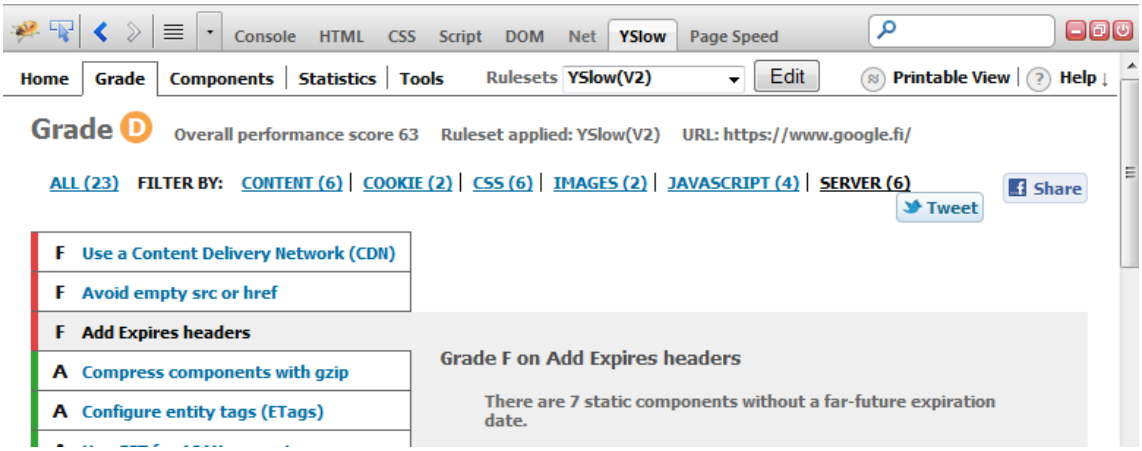
JavaScript-testauksen ongelmana on huono skaalautuvuus. Käyttäjille on tarjolla paljon erilaisia selaimia, joista on myös saatavilla useita eri versioita. JavaScript-koodi suorite-

taan selaimessa, joista jokainen saattaa suorittaa koodin hieman eri tavoin. Testisarjat täytyy suorittaa siis erikseen jokaiselle eri selaimelle eri käyttöliittymillä. Lisäksi testit täytyy suorittaa uudelleen jokaisen päivityksen yhteydessä. JavaScript-testaus ei siis skaalaudu hyvin, ja se vie paljon aikaa. Mahdollisena ratkaisuna tähän esitetään vapaaehtoisten käyttäjien voimin toimivaa testiverkostoa nimeltä TestSwarm. Ratkaisu ei kuitenkaan ole toimiva kaupallisille palveluille, jotka eivät halua paljastaa uusia toiminnallisuuksia ennen niiden julkaisua. [27.]

4 Dynaamisen käyttöliitymän kehityksen haasteet ja ongelmat

4.1 Suorituskyky

Vaikka suorituskyky yleensä paranee dynaamisuuden lisääntyessä, voi tilanne myös kääntyä toisinpäin monimutkaisia ja suuria datamääriä käsittelevien sivujen kohdalla. Monimutkaiset verkkopalvelut voivat sisältää useita, jopa kymmeniä JavaScript-tiedostoja, joissa voi olla useita tuhansia rivejä koodia. Kun tiedostot täytyy ladata ensimmäisen sivunlatauksen yhteydessä, jotta käyttäjällä olisi kaikki mahdolliset toiminnallisuudet käytössään, voi sivunlataus kestää useita sekunteja. Käyttäjät eivät jaksa odottaa pitkää sivunlatausta varsinkaan ensimmäistä kertaa sivulle tullessaan. Tutkimusten mukaan puolet käyttäjistä olettaa sivun latautuvan alle kahdessa sekunnissa ja sivu hylätään, mikäli sen latautuminen kestää yli kolme sekuntia. Oman sivuston nopeuden määrittämiseen on useita selaimeen liitettäviä työkaluja, tunnetuimpina Google Page Speed ja Yahoo YSlow. Yslow, jonka Firefox-lisäosaa kuva 6 havainnollistaa, antaa myös sivukohtaisia neuvoja ja parannusehdotuksia sivuston nopeuden lisäämiseksi. [29, s. 2; 30.]



Kuva 6. Firefoxin Firebug-kehitystyökalun Yahoo Yslow -lisäosa.

Sivuston latausaikaa voidaan parantaa useilla eri keinoilla. Suurin osa latausajasta kuluu sivuston eri komponenttien, kuten kuvien, CSS-tyylitiedostojen ja JavaScript-tiedostojen lataamiseen. Tärkein vaihe latausajan pienentämisessä on HTTP-pyyntöjen minimointi. Ladattavien tiedostojen määrän vähentäminen vähentää myös vaadittujen HTTP-pyyntöjen määrää. Pyyntöjen määrää voidaan vähentää yhdistämällä kaikki tyy-

limäärittelyt yhteen tiedostoon ja JavaScript-koodi yhteen. Myös kuvatiedostot voidaan yhdistää samaan kuvatiedostoon käyttämällä CSS Sprite -tekniikkaa. Sprite-tekniikassa erilliset kuvatiedostot yhdistetään yksittäiseksi kuvaksi, josta näytetään verkkosivulla oikea kohta CSS background-position -määrittelyllä. Sprite-tekniikka sopii parhaiten kuviin, jotka toimivat elementtien taustakuvina. Vaaka- ja pystysuunnassa toistuviin kuviin ei Sprite-tekniikkaa voi hyödyntää. [31; 32.]

Selaimet pystyvät lataamaan useita tiedostoja rinnakkain, mikä nopeuttaa sivun latautumista. JavaScript-tiedostot sen sijaan estävät tiedostojen rinnakkaislataukset ja hidastavat sivun latautumista. Tämän vuoksi on hyvin suositeltavaa laittaa skriptitiedostot sivun loppuun. Tilanteissa, joissa suurin osa sivun sisällöstä ladataan Ajax-pyyntöjen kautta, tämä voi olla erittäin haastavaa. Tällöin tulee harkita tarkoin tiedostojen sijainnit ja tarkistaa, mikä sijoittelu antaa parhaan lopputuloksen. [31; 32.]

CSS-tyylitiedostojen sijoittelulla HTML-koodissa on vaikutusta siihen, kuinka nopeasti sivu vaikuttaa latautuvan. Tyylitiedostot tulisi sisällyttää HTML-koodin head-osaan, jolloin mahdollistetaan sivun latautuminen progressiivisesti. Muutoin käyttäjä näkee ainoastaan tyhjän valkoisen sivun, kunnes koko sivu on latautunut. Progressiivinen latautuminen sen sijaan mahdollistaa sisällön näyttämisen käyttäjälle sitä mukaa, kuin se latautuu, jolloin käyttökokemus parantuu. [31.]

JavaScript-koodi suoritetaan käyttäjän selaimessa, jolloin haasteena on suuri koodimäärä, joka hidastaa sivunlatauksia ja voi vanhemmissa tietokoneissa toimia hitaanlaisesti. Koodin määrä on pyrittävä pitämään mahdollisimman pienenä, ja koodin rakenne on suunniteltava niin, että samoja funktioita voidaan käyttää uudelleen eri tilanteissa. Tärkeää on myös tiedostojen minimointi eli tarpeettomien merkkien poistaminen. JavaScript- ja CSS-tiedostojen minimointi vähentää niiden kokoa keskimäärin 21 %. Suosituimpia työkaluja minimointiin ovat JSMin ja YUI Compressor. Minimoinnin lisäksi myös tiedostojen pakkaus vähentää vasteaikoja. Tällä hetkellä suosituin ja tehokkain pakkaustapa on Gzip, joka vähentää HTTP-vastauksen kokoa noin 70 %. [1, s. 224; 31; 32.]

Tärkeä keino sivuston latausajan lyhentämiseen on komponenttien tallennus selaimen välimuistiin. Välimuistiin voidaan tallentaa kaikki sivuston komponentit, kuten kuvat ja

tyyli-, skripti- ja flash-tiedostot. Kun sivustolla vieraillaan ensimmäistä kertaa, valitut komponentit tallennetaan välimuistiin. Myöhemmillä vierailukerroilla selain voi avata komponentit suoraan välimuistista, ilman aikaa vieviä HTTP-pyyntöjä. Välimuistin hallintaa varten palvelimet käyttävät HTTP-pyynnön Express header- ja Cache-Control-tietoja. Näihin HTTP-pyynnön tietoihin asetetaan päivämäärä, jolloin komponentti vanhentuu ja se täytyy ladata uudelleen palvelimelta selaimen välimuistin sijaan. Näiden lisäksi entiteettitietojen määrittäminen auttaa pitämään välimuistin ajan tasalla. Entiteettitiedon avulla voidaan selvittää, vastaako välimuistissa oleva komponentti palvelimella olevaa. Toisin sanoen voidaan tarkistaa, onko komponenttia päivitetty ja tarvitseeko se ladata uudelleen. [31.]

Myös suurten datamäärien lataaminen Ajax-pyyntöjen välityksellä on raskasta ja vie aikaa. Käyttäjä voi joutua odottamaan vastausta useita sekunteja, jolloin käytettävyys heikkenee. Tämä on haasteena erityisesti erityyppisissä hakupalveluissa, joissa liikkuu suuria datamääriä. Yleisin ratkaisu ongelmaan on sivuttaminen, eli vastausten jako usealle sivulle. Mikäli kaikki tulokset halutaan kuitenkin näyttää samanaikaisesti, voidaan lisää tuloksia hakea esimerkiksi, kun sivu on vieritetty loppuun, tai vaihtoehtoisesti ohjeistaa käyttäjää lataamaan lisää tuloksia. Esimerkiksi hakupalveluissa tulosten yhtäaikainen suuri määrä ei kuitenkaan ole edes käytettävyyden vuoksi toivottavaa, joten tulosten määrää voidaan rajoittaa ja ohjeistaa käyttäjiä rajaamaan hakutuloksia. Näistä ratkaisuvaihtoehdoista mikään ei kuitenkaan ole käytettävyyden kannalta täydellinen, mutta niiden joukosta löytyy useimmiten soveltuva ratkaisu kaikkiin palveluihin. [18, s. 12; 33.]

4.2 Käytettävyys

Dynaamisten käyttöliittymien tarkoituksena on sivuston käytettävyyden parantaminen. Suurin syy dynaamisten käyttöliittymien suosioon on niiden nopeus verrattuna perinteisiin verkkosivuihin. Verkkosivun dynaamisuus parantaa sivun suorituskykyä huomattavasti, sillä kokonaisen sivulatauksen sijasta dynaamisessa käyttöliittymämallissa ladataan ainoastaan tarvittavat tiedot. Myös visualisointi on erittäin tärkeässä roolissa käyttöliittymiä toteutettaessa. Visualisoinnin avulla voi joko parantaa käyttökokemusta entisestään tai pilata sen.

Suurimpia haasteita mitä tahansa käyttöliittymiä suunniteltaessa ja toteutettaessa on käytettävyys. Varsinkin dynaamisia käyttöliittymiä toteutettaessa on tärkeää, että verkkosivun visuaalinen ilme ja toiminnalliset elementit ovat selkeitä ja yksiselitteisiä. Perinteisten nappien sijasta toiminnallisuuksia voivat laukaista mitkä tahansa käyttäjän verkkosivulla suorittamat toiminnot, kuten hiiren tai vierityspalkin liikuttaminen. Mikäli toiminnalliset elementit eivät ole selkeitä, käyttäjän voi olla vaikea ymmärtää, mistä tai miten eri toiminnallisuudet saa käyttöönsä. Käyttäjälle on myös selkeästi havainnollistettava, mikä osa sivusta on päivitetty. [18, s. 16–17.]

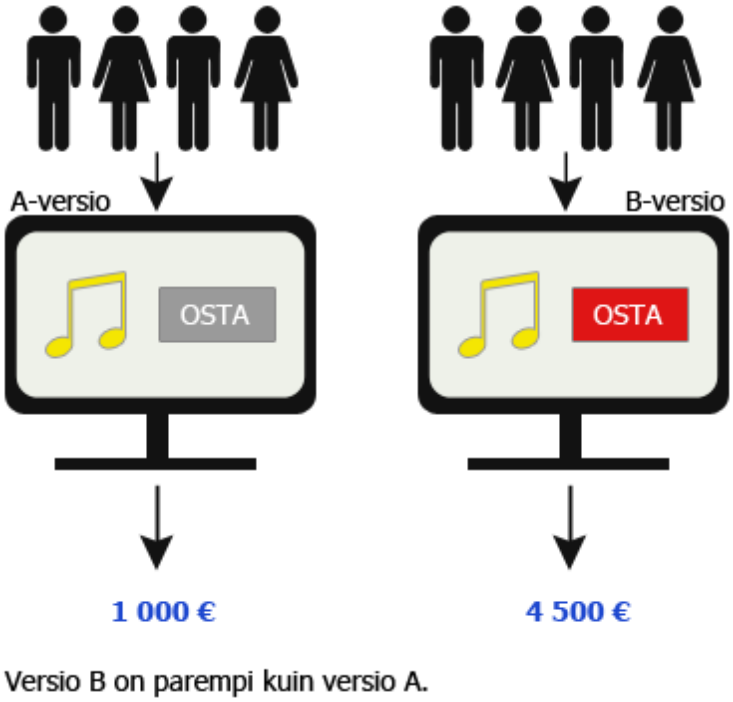
Verkkopalveluita ja -sivustoja kehittävät henkilöt ovat usein oman alansa asiantuntijoita, jolloin heidän voi olla hyvinkin vaikeaa nähdä sivustot peruskäyttäjien näkökulmasta. Tämän takia on erittäin tärkeää tehdä käytettävyystestausta erilaisilla kohderyhmään kuuluvilla henkilöillä. Myös A/B-testaus (ks. s. 25) ja käyttäytymisen seuranta erilaisilla kävijäseurantatyökaluilla on hyvä tapa parantaa sivuston käytettävyyttä. [34, s. 2.]

Käytettävyystestaus on tapa varmistaa, että sivusto on käyttäjilleen helppokäyttöinen. Testiin osallistuvat henkilöt antavat palautetta sivustosta, ja siinä ilmenevät mahdolliset ongelmakohdat nousevat esiin testauksen aikana. Perussääntö käytettävyystestauksessa on, että käytetään työasemia ja selaimia, joita suurin osa käyttäjistä todennäköisesti käyttää vieraillessaan sivustolla. Testiin osallistuvilla henkilöillä olisi hyvä olla ainakin vuoden kokemus internetin käytöstä, jotta varmistutaan siitä, että käyttäjä hallitsee internetin käytön perusteet. Tällöin tutkimuksessa voidaan keskittyä ainoastaan kyseisen sivuston käytettävyyden arviointiin. [34 s. 19; 35, s. 5–6.]

Käytettävyystestausta on kolmea eri tyyppiä: käyttäjäperusteinen, ammattilaisperusteinen ja automatisoitu testaus. Kun käyttöliittymäsuunnittelun ammattilainen tekee käyttöliittymätestauksen, selviää käyttöliittymässä usein laajempia ongelmia, jotka voivat rikkoa käyttöliittymäsuunnittelun periaatteita. Ammattilaiset pystyvät myös antamaan parannusehdotuksia ja ohjeita esimerkiksi ulkoasun yhtenäistämiseen ja käytettyyn sanastoon. Yleisimmin käytettävyystestauksessa käytetään kuitenkin tavallisia käyttäjiä, joiden tekemissä testeissä selviää usein pienempiä ja yksityiskohtaisempia ongelmia, jotka eivät ammattilaiselle ole niin ilmeisiä. Automatisoidussa testauksessa käytetään

erilaisia sovellustyökaluja, jotka etsivät sivulta tavallisimpia käytettävyysongelmia. [34, s. 206–225.]

Automatisoitua käyttäjäperusteista testausta voidaan tehdä mielenkiintoisella tavalla A/B-testauksen avulla käyttäjien sitä itse huomaamatta. Kuvan 7 mukaisesti A/B-testaus tarkoittaa sitä, että verkkosivun ulkoasusta tehdään kaksi erilaista versiota, tyypillisesti vanha ja uusi ulkoasu. Verkkosivun liikenne jaetaan näiden versioiden välille, ja kummankin suorituskykyä mitataan haluttujen mittareiden, kuten myynnin, konversioprosentin ja välittömän poistumisprosentin mukaan. Konversioprosentti ilmaisee suhdetta, kuinka suuri osa sivustolle saapuvista käyttäjistä päätyy tekemään halutun tapahtuman, ja välitön poistumisprosentti sitä, kuinka moni sivustolle saapuvista kävijöistä poistuu välittömästi. Lopuksi versioista valitaan käyttöön se, joka suoriutuu parhaiten. A/B-testaus auttaa saavuttamaan halutut tavoitteet käytettävyyden parantamisesta rekisteröityneiden käyttäjien tai myynnin kasvuun. [36.]



Kuva 7. Verkkopalvelun käyttöliittymän A/B-testaus.

Erilaiset kävijäseurantatyökalut ovat kätevä tapa analysoida verkkosivustojen käytettävyyttä. Työkalujen avulla on mahdollista seurata tarkasti, mistä käyttäjät päätyvät sivustolle, miten he liikkuvat siellä ja mitä toimintoja he suorittavat. Työkalut tallentavat

myös kattavasti tiedot käyttäjän selaimesta, käyttöliittymästä ja sijainnista. Näiden tietojen perusteella on helppo selvittää esimerkiksi, käytetäänkö käyttöliittymää oikein, tukeeko palvelu yleisimpiä käytettyjä selaimia ja kuinka paljon eri sivuston osioita käytetään. Suomessa suosituin ilmainen kävijäseurantapalvelu on Google Analytics, joka sopii hyvin niin pienen kuin suuremmankin sivuston työkaluksi. [37.]

4.3 Muut tekniset haasteet

Navigaatio

Dynaamisilla verkkosivuilla ei perinteinen URL-pohjainen navigaatio toimi, kun käyttäjä ei siirry sivulta toiselle, vaan pysyy samalla sivulla, kun ohjelmakoodi muokkaa sivun sisältöä käyttäjän toimintojen mukaisesti. Tämä aiheuttaa sen, että käyttäjän painaessa selaimen taakse- ja eteen-nappuloita, selain siirtyykin sivulta pois, viimeiseen oikeasti vaihtuneeseen osoitteeseen. Kun URL-osoite pysyy samana, myöskään kirjanmerkit tai linkit eivät toimi oletetulla tavalla vaan ohjaavat käyttäjän aina sivuston etusivulle. [1, s. 18.]

Tämän ongleman ratkaisuna on verkkosivun URL-osoitteen perään lisättävä tiedon osan kuvaava tunniste, joka erotetaan verkkosivun varsinaisesta osoitteesta ristikkomerkillä (”#”). Tunnisteita voi lisätä osoitteen perään rajattomasti, ja niiden lisääminen tai poistaminen ei aiheuta sivun päivittämistä uudelleen, joten tunnisteita voidaan muokata sitä mukaa, kuin käyttäjä työskentelee sivustolla. Tällöin käyttäjä voi myös halutessaan päivittää sivun, laittaa sen talteen kirjanmerkkeihin tai jakaa linkin eteenpäin ja sivusto käyttäytyy oletetulla tavalla, avaten aina oikean sisällön. Tunnisteiden hallinnan voi toteuttaa haluamallaan tavalla JavaScript-koodissa itse, tai vaihtoehtoisesti voi käyttää valmiita saatavilla olevia lisäosia. [38.]

Suurin navigaation haaste tulee hakupohjaisilla verkkosivuilla. Kaikki käytetyt hakuehdot tulee lisätä verkkosivun URL-osoitteen perään, jotta uudelleen osoitteeseen tultaessa hakuehdot pysyvät samoina. Tämän seurauksena verkkosivun osoitteesta tulee nopeasti hyvinkin pitkä ja sekava. Pitkät ja vaikeasti luettavat osoitteet eivät houkuttele käyttäjiä siirtymään osoitteeseen. Myös hakukoneiden algoritmit pitävät pitkiä osoitteita

epäsuotuisina ja hakukonenäkyvyys voi huonontua. Tämä on huono asia niin käyttäjiä kuin verkkosivun käytettävyyttä ja suosiotakin ajatellen. [39.]

Tiedostojen lataus

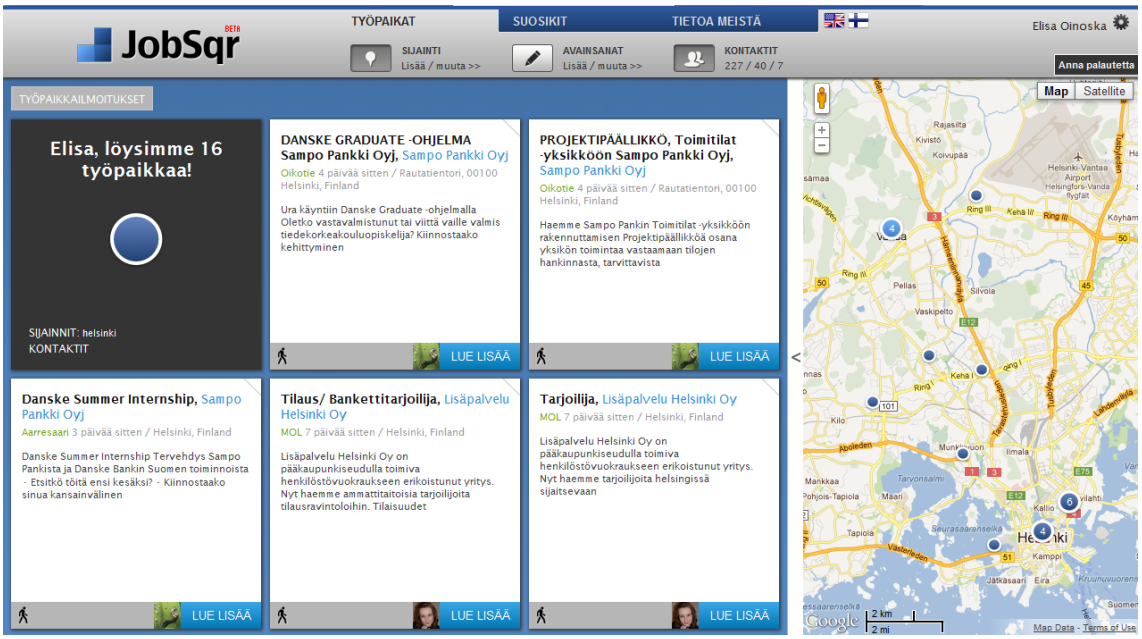
Perinteisesti tiedostojen lataus ei ole onnistut asynkronisesti ja sivu on täytynyt aina päivittää uudelleen tiedostojen latauksen yhteydessä. Tämä toimintamalli ei tietenkään sovellu Ajax-sovelluksiin, ja ongelma on pyritty kiertämään käyttämällä piilotettua If-rame-elementtiä, jossa tiedosto ladataan palvelimelle. Iframe on HTML-elementti, joka mahdollistaa erillisen HTML-dokumentin sisällyttämisen toisen dokumentin sisään. If-rame-elementtien käyttö ei kuitenkaan ole suositeltavaa, sillä kaikki selaimet eivät tue niitä, eivätkä ne ole XHTML 1.0 Strict-standardin mukaisia. HTML5 tuo kuitenkin ratkai-sun tähän ongelmaan File API -nimisellä ohjelmointirajapinnalla. Tämä määritelmä lisää XMLHttpRequest-oliolle mahdollisuuden hallita tiedostojen lataamista. Kaikki selaimet eivät kuitenkaan vielä tue tätä ohjelmointirajapintaa, joten kehittäjien on toistaiseksi toteutettava vaihtoehtoiset ratkaisut, joista suoritetaan oikea käyttäjän selaimesta riip-puen. [40; 41.]

5 Dynaaminen työnhaun verkkopalvelu

5.1 Vaatimukset

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa verkossa toimiva dynaaminen JobSqr-niminen työnhakupalvelu. Palvelun tuli olla helppokäyttöinen, nopea ja skaalautuva. Palvelun täytyi toimia yleisimmillä selaimilla ja mobiililaitteilla. Käyttöliittymävaatimuksia olivat tuki eri kieliversioille, integraatio Google Maps -karttapalveluun ja yhteys Facebook API -ohjelmointirajapintaan. Käyttöliittymä tuli myös laatia moduuleittain, niin että yksittäiset moduulit voitaisiin helposti päivittää Ajax-tekniikkaa hyödyntäen ja käyttäjä pysyisi työpaikkoja selatessaan koko ajan samalla sivulla.

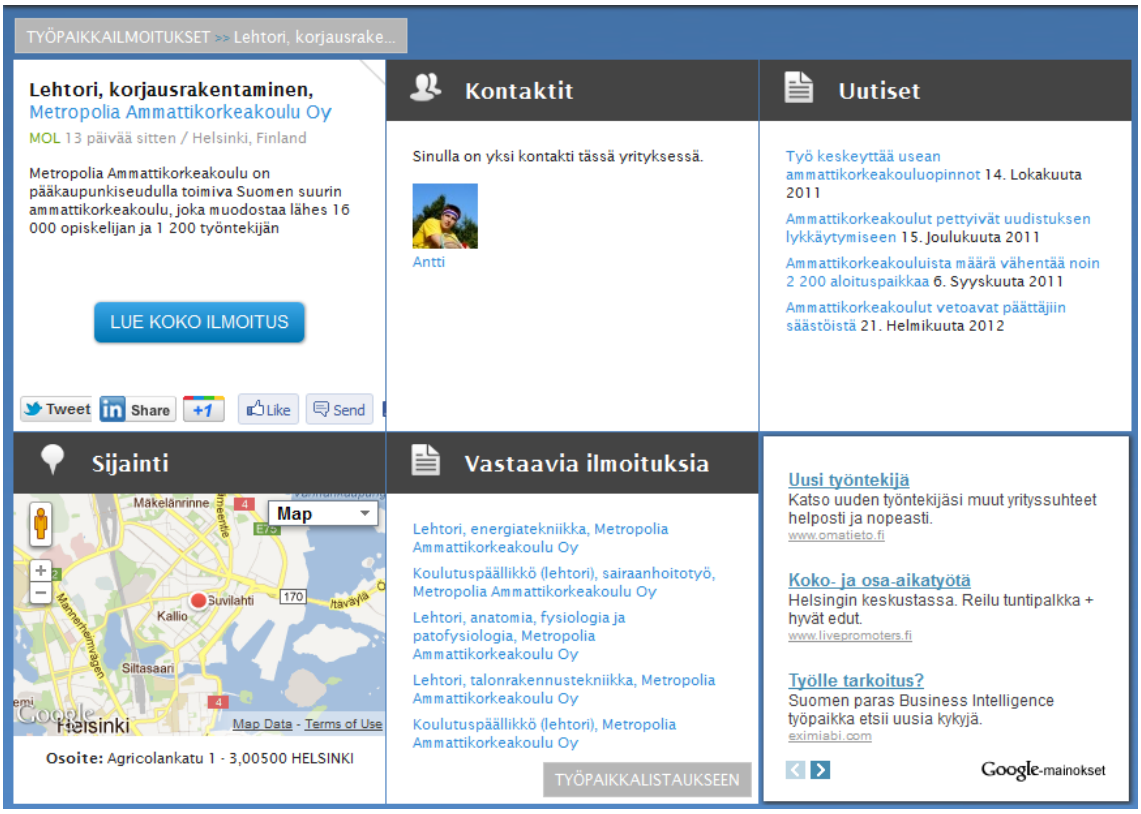
Käyttöliittymässä piti olla kuvan 8 mukaisesti suurina elementteinä yläpalkki, jossa sijaitsee sivun navigaatio ja hakukriteerit, työpaikkalistaus sivun vasemmalla puolella ja kartta oikeassa laidassa. Sekä yläpalkin että kartan täytyisi pysyä jatkuvasti näkyvissä käyttäjän selatessa työpaikkoja selaimen vierityspalkkia liikuttamalla. Hakukriteerejä käyttäjän pitäisi pystyä lisäämään helposti sisällön päälle avautuvista elementeistä. Itse työpaikat tuli toteuttaa yksittäisinä elementteinä.



Kuva 8. JobSqr-työnhakupalvelun elementit ja ulkoasu.

Työpaikkoja tulisi voida selata myös pelkästään kartan avulla. Työpaikkojen tuli näkyä kartalla kuvakkeina eli pinneinä. Pinnejä valitsemalla avautuisivat tiedot työpaikasta ja yrityksestä. Käyttäjän olisi mahdollista myös itse muuttaa kartan kokoa haluamakseen. Kartan tulisi kommunikoida aktiivisesti työpaikkailmoituslistan kanssa, jolloin esimerkiksi viemällä hiiren työpaikkalistauksen elementin päälle näkyisi kartalla oleva kyseisen työpaikan pinni korostettuna. Samoin myös kartalla olevia pinnejä hiirellä osoittamalla työpaikka korostuisi työpaikkalistauksessa.

Työpaikkaelementissä tulisi olla linkki, jota painamalla kaikki muut listauksen työpaikkaelementit piilotettaisiin ja tilalla näytettäisiin muita valittuun työpaikkaan liittyviä tietoja. Näytettävien lisätietojen tulisi olla yritykseen liittyvät käyttäjän omat Facebook-kontaktit, ajankohtaiset uutiset, työpaikan sijainti ja muut vastaavat työpaikkailmoitukset, kuten kuvasta 9 ilmenee. Käyttäjän tulisi pystyä myös lukemaan työpaikkailmoitus kokonaisuudessaan ja jakamaan se sosiaalisessa mediassa.



Kuva 9. Yksittäisen työpaikkailmoituksen näkymä työpaikkalistauksessa.

Muita vaadittuja ominaisuuksia olivat palautteen lähettäminen, sisäänkirjautuminen ja suosikkityöpaikkojen hallinnointi. Palautteen lähettäminen oli yksi komponenteista, joka

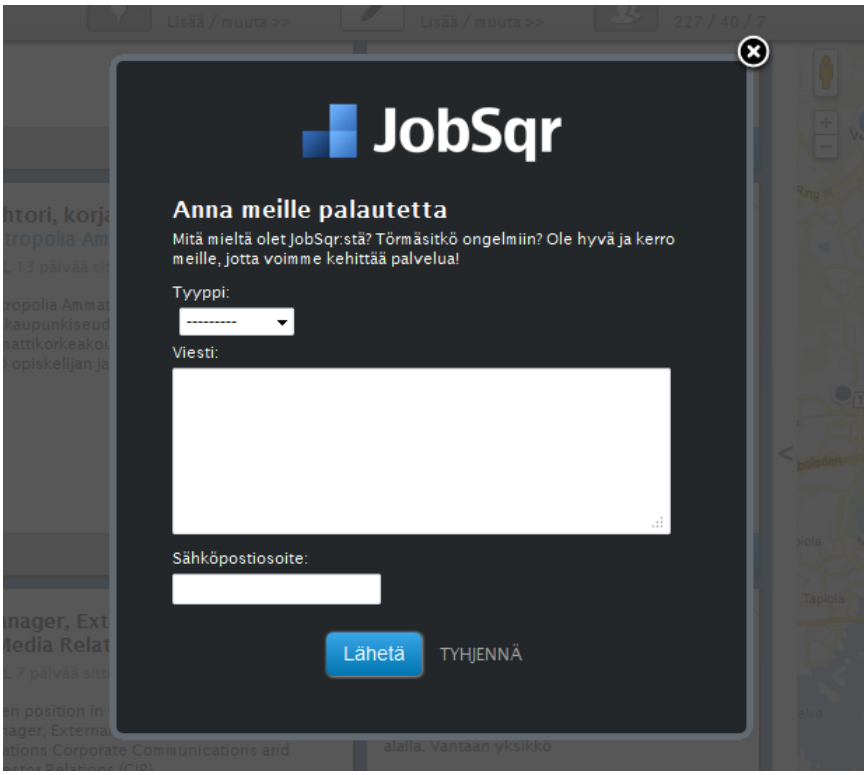
toteutettiin myös muita palveluja silmällä pitäen. Palautteen lähettäminen on komponentti, joka voidaan lisätä helposti mihin tahansa palveluun, mille tahansa sivulle. Sen lisääminen käyttöliittymään vaatii komponentin omien CSS- ja JavaScript-tiedostojen liittämisen sivulle ja pätkän HTML-koodia, palvelinpuolen koodia tietenkään unohtamatta. Myös sisäänkirjautuminen toteutettaisiin Ajax-pohjaisena komponenttina, joka olisi mahdollista helposti lisätä myös muihin palveluihin. Käyttäjän tulee voida kirjautua ja rekisteröityä palveluun joko Facebook-tunnuksilla tai perinteisesti lomaketta käyttäen. Käyttäjän tulisi pystyä rekisteröitymisen jälkeen asettamaan työpaikkalistauksesta suosikkityöpaikkoja, joita voisi tarkastella erillisellä sivulla.

5.2 Käyttöliittymän toteutus

5.2.1 Palaute ja sisäänkirjautuminen

Koska palautteen lähettäminen haluttiin tehdä komponenttina, jota olisi helppo hyödyntää myös muissa palveluissa, siitä tehtiin erillinen moduuli, joka erotettiin muusta koodista. Haluttiin myös, etteivät mahdolliset virheet muualla koodissa estäisi palautteen lähettämistä, jotta mahdolliset virheraportit saataisiin aina perille. Myös tämän vuoksi moduulimalli sopi palautteen lähettämiseen erinomaisesti. Koodit palautteen lähettämiseen asetettiin ladattavaksi ensimmäisenä sivulla, sillä mikäli koodissa olisi vakava virhe ennen palautekoodia, ei myöskään palautteen lähettämisen mahdollistava osuus toimisi.

Palautteen lähettäminen toimii painamalla käyttöliittymän palaute-linkkiä, jolloin selaimeen avautuu muun sisällön päälle uusi elementti, johon voi syöttää haluamansa palautteen, kuten kuvassa 10 näkyy. Tämä palaute-elementti niin sanotusti kelluu muun sisällön päällä, jolloin palautteen lähettäminen ei vaikuta alla olevaan sisältöön lainkaan ja palvelun käyttöä voi jatkaa välittömästi palautteen lähettämisen jälkeen. Käyttäjä voi valita palautteen tyypin vaihtoehdoista kehu, kysymys, idea tai ongelma, jolloin vastaanotettu palaute voidaan helposti järjestää tärkeysjärjestykseen, ja esimerkiksi mahdolliset ongelmat palvelun käytössä voidaan korjata välittömästi. Viestikenttään syötetään perinteisesti haluttu palaute, ja käyttäjä voi myös halutessaan lisätä sähköpostiosoitteensa, mikäli haluaa palautteeseen vastattavan.



Kuva 10. Sisällön päällä leijuva palautelomake.

Kun käyttäjä painaa ”Lähetä”-linkkiä, JavaScript tekee Ajax-pyynnön palvelimelle. Mikäli palautteessa ilmenee ongelmia, palvelin vastaa virheviestillä, joka esitetään käyttäjälle. Tämän jälkeen käyttäjä voi korjata mahdollisen syötevirheen ja lähettää palautteen uudelleen. Palvelimen vastattua onnistuneesti voi käyttäjä sulkea palauteikkunan ja jatkaa työskentelyä. Palautteen lähettäminen on myös helposti keskeytettävissä missä tahansa vaiheessa elementin yläkulmassa olevaa rastikuvaketta painamalla.

Sisällön päällä leijuva palaute-elementti on toteutettu käyttäen hyväksi Fancybox-nimistä jQuery-kirjaston kanssa toimivaa liitännäistä. Kuten muutkin liitännäiset, pystyy Fancyboxin ottamaan käyttöön lisäämällä sivustolle sen vaatimat CSS- ja JavaScript-tiedostot ja kutsumalla sen alustusfunktiota. Fancybox on työkalu, joka mahdollistaa kuvien, HTML-sisällön ja multimedian näyttämisen muun sisällön päällä. Vaikka ominaisuuden voisi itsekin toteuttaa, nopeuttaa kirjaston käyttö työskentelyä kuitenkin huomattavasti. Lisäksi ominaisuuden saa helposti lisättyä kaikkiin tarvittaviin elementteihin, joille jokaiselle voi asettaa omat määritykset yksinkertaisen objektimuotoisen merkintätavan avulla, joka on esitetty esimerkkikoodissa 9.

```
$("a#feedback_link").fancybox({
   'overlayColor' :'#454545',
   'centerOnScroll' : true,
   'width' : 450,
   'height' : 480,
   'autoDimensions' : false,
   'padding' : 5,

   // muita määrityksiä...
});
```

Esimerkkikoodi 9. Fancybox-liitännäisen alustaminen palautelomakkeeseen.

Itse palautteen lähettäminen suoritetaan käyttäen AJAX post()-metodia, joka mahdollistaa tiedon lähettämisen palvelimelle turvallisesti. Käyttäjän syöttämän palautteen lisäksi palautteen yhteydessä lähetetään palvelimelle tietoja itse käyttäjästä. Käyttäjästä kerättäviä tietoja ovat muun muassa käytettävä selain ja sen versionumero, käyttöjärjestelmä, näytön ja selainikkunan koko sekä se, ovatko JavaScript ja evästeet käytössä. Kerättävien tietojen tarkoituksena on helpottaa mahdollisten palvelussa tai sen käytössä esiintyvien ongelmien ratkaisua, sillä käyttäjät eivät usein itse osaa näitä tarvittavia tietoja selvittää.

Myös sisäänkirjautuminen ja rekisteröityminen on toteutettu käyttäen hyväksi samaa Fancybox-liitännäistä. Saman sisäänkirjautumiselementin sisällä on kolme näkymää, jotka mahdollistavat sisäänkirjautumisen ja rekisteröitymisen Facebookin välityksellä tai käyttäen perinteisiä lomakkeita. Näitä eri näkymiä vaihdellaan käyttäjän toimintojen mukaan muokkaamalla CSS-määrittelyjä dynaamisesti JavaScriptillä. Myös sisäänkirjautuminen on itsenäinen moduuli, jonka sisällä olevat funktiot hallitsevat koko kirjautumis- ja rekisteröitymisprosessia.

Facebook tarjoaa oman JavaScript-kehitysympäristön, joka mahdollistaa pääsyn palvelinpuolen ohjelmointirajapintoihin. Näiden ohjelmointirajapintojen välityksellä voidaan kirjata käyttäjä sisään palveluun käyttäen Facebook-tunnuksia. Sen avulla voidaan myös tuoda palveluun käyttäjästä erilaisia saatavilla olevia tietoja, kuten Facebookiin asetetun profiilikuvan, osoitteen ja työpaikka- ja kaveritiedot. Tietojen hakeminen vah-

vistetaan käyttäjältä sisäänkirjautumisvaiheessa, ja Facebookin käyttöehtojen mukaan tiedot tulee voida poistaa palvelusta käyttäjän niin halutessa. Haettuja tietoja voidaan hyödyntää palvelussa eri tavoin, jolloin palvelusta saadaan käyttäjille entistä henkilökohtaisempi. [42; 43.] Facebook-tietojen hyödyntämistä JobSqr-palvelussa käsitellään luvuissa 5.2.2 ja 5.2.4.

5.2.2 Työpaikkojen etsiminen

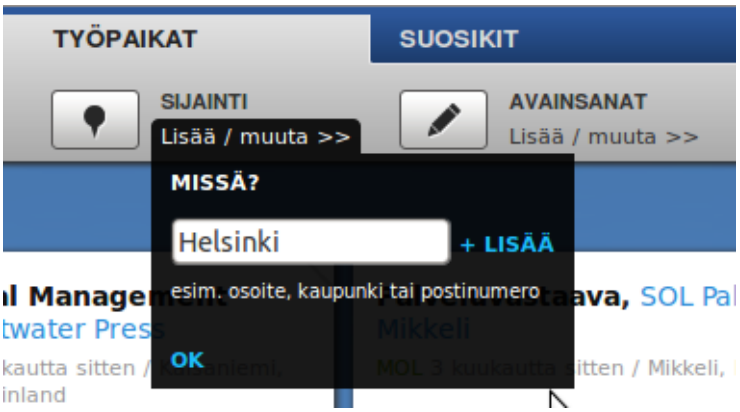
Käyttäjät voivat etsiä palvelun avulla avoimia työpaikkoja sijainnin ja avainsanojen mukaan. Tämän lisäksi Facebookin avulla sisäänkirjautuneet käyttäjät voivat hakea työpaikkoja perustuen omien kontaktien työpaikkatietoihin. Käytettävät hakukriteerit määritellään yläpalkkielementissä, jossa on eri hakukriteereille omat alueensa. Hakukriteerilomakkeet avautuvat sisällön päälle käyttäjän painaessa tekstimuotoista linkkiä. Avautuvassa lomakkeessa voi syöttää uusia hakukriteereitä ja poistaa vanhoja. Hakukriteerikuvaketta painamalla voidaan vaihdella kutakin hakukriteerityyppiä käyttöön ja pois käytöstä. Eri hakukriteerityyppejä voidaan käyttää samanaikaisesti, jolloin palvelin laskee keskimääräisesti parhaan tuloksen ja palauttaa sen.

Avainsanahaku toimii oletettavalla tavalla, eli syötetyt hakusanat vain asetetaan parametreina Ajax-pyyntöön, jolloin palvelin etsii tietokannan työpaikkatiedoista hakusanoja vastaavia työpaikkoja ja palauttaa ne vastauksena. Sijaintiin perustuva työpaikkahaku toteutettiin käyttäen Google Maps JavaScript API V3 -ohjelmointirajapintaa. Google Maps tarjoaa Geocoding-nimisen palvelun, jonka avulla osoitetietoja voidaan muuttaa maantieteellisiksi koordinaateiksi. Käyttäjän syöttäessä haluamansa osoitteen lomakkeeseen palvelu lähettää pyynnön Google Maps Geogoding -ohjelmointirajapinnalle, joka palauttaa vastauksena sopivat koordinaatit. Mikäli käyttäjän syöttämällä sijainnilla ei löydy sijaintitietoja, lähettää Geogoding-palvelu vastauksena virheviestin, joka esitetään käyttäjälle. Käyttäjän tulee tällöin korjata virheellinen osoitetieto ja yrittää syöttämistä uudelleen. Kun koordinaatit vastaanotetaan onnistuneesti, ne asetetaan parametreiksi palvelimelle lähetettävään Ajax-pyyntöön, joka palauttaa vastauksena lähimpänä syötettyä sijaintia olevat avoimet työpaikat.

Käyttäjän syöttämät hakukriteerit tallennetaan selaimen evästeisiin, jolloin tallennetut tiedot pysyvät tallessa myöhempiä vierailukertoja varten. Tavalliset avainsanat tallen-

netaan tekstimuotoisena, mutta sijaintitiedot tallennetaan sen sijaan pituus- ja leveysasteina. Varsinaista selkokielistä sijaintia ei tallenneta kuitenkaan erikseen, vaan se haetaan tallennettujen koordinaattien perusteella käyttäjän palatessa palveluun. Tätä toiminnallisuutta kutsutaan Google Maps -palvelussa nimellä reverse geogoding. Päinvastoin kuin tavallinen geogoding, se palauttaa syötetyille koordinaateille selkokielisen osoitteen. Osoitetietojen käsittelyyn Google Maps -palvelu on erinomainen, sillä palvelu on ilmainen ja se kattaa lähes koko maapallon osoitetiedot, syrjäisiä alueita lukuun ottamatta.

Avautuvat hakukriteerilomakkeet toteutettiin käyttäen hyväksi jQuery Tools -nimisen kirjaston Tooltip-ominaisuutta. Tooltip-ominaisuus eroaa palautteen lähettämiseen ja sisäänkirjautumiseen käytetystä Fancyboxista siten, että se luo efektin, joka näyttää siltä, kuin olemassa olevasta elementistä avautuisi uusi osio. Tooltip jäljittelee verkkosivulla olevan elementin title-attribuutin toimintaa, mutta se mahdollistaa paljon näyttävämmät visuaaliset efektit ja title-attribuutin pelkän informaatiotekstin näyttämisen sijaan minkä tahansa sisällön upottamisen Tooltip-elementtiin. Kuvassa 11 näkyy esimerkki sijaintihakukriteerien hallintaan käytettävästä Tooltip-elementistä.



Kuva 11. Sijaintihakukriteerien hallintaan käytettävä Tooltip-elementti.

Facebook-kontaktien perusteella toimiva työpaikkahaku toimii käyttäjillä, jotka ovat kirjautuneet palveluun Facebook-tunnusten avulla. Käyttäjän kirjautuessa sisään palvelinpuolen ohjelmakoodi selvittää yhteydet käyttäjän kavereiden työpaikkatiedoista tietokannassa oleviin yrityksiin. Kun käyttäjä valitsee hakukriteeriksi kontaktit, kirjautumisen yhteydessä selvitetyt yhteydet palautetaan ja käyttäjä näkee työpaikkalistauksessa avoimet työpaikat, joissa hänellä on henkilökohtaisia kontakteja. Tämä on tärkeä etu

työmarkkinoilla, sillä työpaikan saantiin vaikuttavat tunnetusti myös yrityksessä olevat kontaktit.

Kuten luvussa 4.3 on käyty läpi, navigaation toteuttaminen dynaamisissa verkkopalveluissa voi olla haastavaa. Käyttäjien ja hakukoneoptimoinnin kannalta on tärkeää pitää URL-osoitteet lyhyinä. On todennäköisempää, että kävijät vierailevat osoitteessa, joka on lyhyt ja helposti luettava. Tämä on kuitenkin erittäin vaikea toteuttaa hakupohjaisissa sovelluksissa, joissa hakukriteerien määrä on rajaton. On myös epätodennäköistä, että eri käyttäjät haluaisivat hakea työpaikkoja täsmälleen samojen kriteerien avulla. Tämän vuoksi lopulta päädyttiin jättämään haut yksilöivät URL-osoitteet kokonaan pois. Käyttäjien ei kuitenkaan tarvitse sivulla uudelleen vieraillessaan asettaa hakuehtoja toistamiseen, sillä ne pysyvät tallessa selaimen evästeissä. [39.]

5.2.3 Työpaikkalistaus

Työpaikkalistaus rakennetaan yksittäisistä työpaikkaelementeistä, joiden järjestys perustuu niiden relevanssiin suhteessa käyttäjän asettamiin hakukriteereihin. Hakukriteerit lähetetään siis Ajax-pyynnön parametreina palvelimelle, joka vastaa JSON-muotoisella tietorakenteella, jossa työpaikat on jaoteltu kolmeen eri tasoon relevanssin mukaan. Kunkin työpaikan relevanssi määritellään palvelinpuolen koodissa, jota ei tässä työssä käydä läpi. Jokaiselle relevanssitasolle lisätään oma otsikkoelementtinsä, josta käyttäjälle ilmenee, kuinka hyvin sitä seuraavat työpaikat vastaavat hakua, kuinka monta työpaikkaa löytyi ja mitkä ovat haulle asetetut hakukriteerit.

Yksittäinen työpaikka on kuvan 12 mukaisesti neliön muotoinen elementti, joka sisältää tiedot haettavasta työpaikasta, työnantajasta, työpaikan osoitteesta sekä ilmoituskanavasta ja –ajankohdasta ja osan työpaikan kuvauksesta. Mikäli käyttäjä on hakenut työpaikkoja sijainnin perusteella, on elementin alareunassa havainnollistava kuvake, joka kertoo, kuinka pitkä matka työpaikasta on käyttäjän syöttämään sijaintiin. Samoin työpaikkaan liittyvät mahdolliset Facebook-kontaktit näkyvät myös kuvina elementin alareunassa. Elementin oikeasta yläkulmasta rekisteröityneet käyttäjät voivat tallentaa työpaikan suosikkeihin. Suosikkeihin lisäys tapahtuu nopeasti Ajax-pyynnön välityksellä, jolloin käyttäjä voi jatkaa työpaikkojen selaamista välittömästi.

Kuva 12. Yksittäinen työpaikkaelementti.

Aluksi elementteihin liitettiin Isotope-niminen jQuery-lisäosa, jonka avulla niihin voitiin lisätä näyttäviä siirtymäefektejä. Isotopen animaatiot toimivat CSS3-tekniikalla, mutta koska vielä monet selaimet eivät tue CSS3-tekniikkaa, niiden animaatiot luodaan JavaScriptin avulla. Isotopen JavaScript-animaatiot aiheuttivat suuren taakan selaimelle, sillä palvelussa on myös paljon muita JavaScriptillä toimivia ominaisuuksia. Tämän vuoksi päätettiin lopulta luopua koko kirjaston käytöstä, sillä käyttäjille on tärkeämpää palvelun nopeus kuin hienot animaatiot. Isotopen animaatiot korvattiin kevyemmillä CSS3-animaatioilla, joiden toimimattomuudesta tietyissä selaimissa ei ole haittaa.

Kun työpaikkatiedot haetaan palvelimelta, ne myös tallennetaan array-muuttujaan taulukkomaisena rakenteena. Työpaikkaelementit luodaan käymällä läpi tämä muuttuja JavaScriptin for-silmukan avulla. Jokaisen työpaikan tiedot syötetään luvussa 3.2 esiteltyä konstruktorimetodia käyttäen Mustache-kirjaston mallifunktiolle, joka käsittelee ne vaaditulla tavalla. Tämän jälkeen konstruktiofunktio palauttaa luodun objektin, joka sisältää työpaikan tiedot ja HTML-elementin mallirakenteen. Lopuksi kutsutaan Mustache-kirjaston render()-metodia, joka asettaa tiedot mallirakenteeseen ja palauttaa valmiin HTML-elementin. Tällä tavoin luodaan kaikki työpaikkaelementit ja tallennetaan ne peräkkäin erilliseen muuttujaan. Mainoselementit lisätään joukkoon samaa metodia käyttäen säännöllisin välein. Kun kaikki elementit on luotu, ne asetetaan sivulle samanaikaisesti.

Sillä aikaa, kun selain suorittaa elementtienkäsittelyprosessia, käyttäjä näkee sivulla pyörivän latauskuvakkeen. Koska käyttäjän ei haluta joutuvan odottamaan latausta liian kauan, haetaan työpaikkoja kerrallaan vain parikymmentä. Käyttäjä voi kuitenkin

halutessaan hakea lisää työpaikkoja sivun alalaidassa olevasta ”Lisää tuloksia” -napista. Nappia painamalla JavaScript lähettää palvelimelle pyynnön, jonka ”start”-niminen parametri kertoo palvelinpuolen ohjelmakoodille kohdan, josta alkaen uudet työpaikkatiedot palautetaan. Uudet työpaikkaelementit lisätään sivulle samaa metodia käyttäen, edellisten elementtien perään. Eräänä vaihtoehtona napille oli uusien työpaikkojen lataaminen aina, kun sivu on selattu loppuun. Menetelmä soveltuu usein hyvin dynaamisiin palveluihin, ja sitä helpottamaan on luotu myös useita kirjastoja. Tähän projektiin se ei kuitenkaan lopulta soveltunut, sillä uusien työpaikkojen lataaminen lataa myös kartan pinnit uudelleen, jolloin käyttäjä saattaa hämääntyä. On siis parempi antaa käyttäjän itse valita uusien työpaikkojen lataaminen.

Kun yksittäinen työpaikkailmoitus avataan, poistetaan muut työpaikkaelementit sivulta ja niiden tilalle luodaan tyhjät lisätietoelementit. Näiden lisätietoelementtien sisältö ladataan asynkronisesti, jolloin jokainen niistä päivittyy heti, kun kyseessä oleva Ajax-pyyntö on suoritettu. Tämän esitystavan ansiosta saadaan käyttäjät ajattelemaan palvelun olevan nopeampi kuin se oikeasti onkaan, jolloin tietojen latautumista jaksetaan odottaa kauemmin. Kun työpaikan lisätiedot on ladattu, ne tallennetaan muuttujiin myöhempää käyttöä varten, jolloin samoja lisätietoja ei tarvitse ladata enää uudelleen. Yksittäisen työpaikkailmoituksen näkymästä siirryttäessä takaisin listaukseen lisätietoelementit poistetaan ja hakulistauksessa olleet elementit luodaan uudelleen tallessa olevien muuttujien perusteella.

Lähes kaikilla sosiaalisen median sovelluksilla on nykyisin omat verkkosivuille asetettavat lisäosansa, joiden avulla käyttäjät voivat muun muassa jakaa mielenkiintoista sisältöä omaan profiiliinsa. Lisäosien lisääminen staattiselle sivulle on hyvin helppoa, sillä se vaatii ainoastaan pienen koodinpätkän kopioimisen. Lisäosien liittäminen asynkronisesti ladatun sisällön joukkoon on kuitenkin vaikeampaa ja vaatii hieman lisätyötä. Vaikka ominaisuutta ei aina virallisesti tueta, onneksi suurin osa liitännäisistä sisältää omat render()-metodinsa, joita on mahdollista kutsua uudelleen aina, kun uusi työpaikkaelementti luodaan, jolloin liitännäiset saadaan ladattua.

Vaikka työnhakulistaus ei tuekaan pitkiä erillisiin hakuihin viittaavia URL-osotteita, yksittäiset työpaikkailmoitukset sen sijaan tukevat yksilöityjä osoitteita. Joka kerta kun työpaikkailmoitus avataan, lisätään URL-parametrien perään risuaitamerkillä erotettuna

kunkin työpaikan yksilöivä nimi ja numero -yhdistelmä. Ilman tätä ominaisuutta eivät kirjanmerkit tai sosiaalisen median liitännäiset toimisi, vaan ne osoittaisivat aina palvelun etusivulle. Erilliset URL-osoitteet ovat tärkeitä myös hakukoneoptimoinnin kannalta. Mikäli esimerkiksi Googlen hakukonerobotit eivät löydä palvelusta kuin yhden ainoan sivun, ei palvelu nouse hakutuloksissa kovinkaan korkealle. Tämän vuoksi on tärkeää pystyä osoittamaan, että sivulla on runsaasti hyödyllistä sisältöä.

5.2.4 Kartta

Palvelussa oleva kartta toteutettiin sijaintihaun tavoin käyttäen Google Maps -ohjelmointirajapintaa. Kartan käyttöönotto on helppoa, sillä se vaatii ainoastaan oman Google Maps JavaScript -tiedoston liittämisen sivustoon ja itse karttaelementin alustamisen. Kuten esimerkkikoodista 10 nähdään, kartan alustaminen vaatii ainoastaan perusasetusten määrittämisen, joiden perusteella uusi karttaelementti luodaan. Itse karttaobjekti on hyvä tallentaa johonkin muuttujaan, jotta karttaan voidaan vaikuttaa myöhemmin. Kartan kokoa voidaan esimerkiksi muuttaa ja siihen voidaan asettaa pinnejä ja tietoikkunoita sekä erilaisia tapahtumankuuntelijoita, jotka reagoivat käyttäjän tekemiin toimintoihin ja itse kartassa tapahtuviin muutoksiin. Karttaan liittyvät toiminnot rakennettiin palvelussa yhdeksi moduuliksi nimeltä mapManager, joka hallinnoi kaikkea karttaan liittyvää. Itse karttaobjekti asetettiin tähän nimiavaruuteen esimerkkikoodissa 10 näkyvällä tavalla.

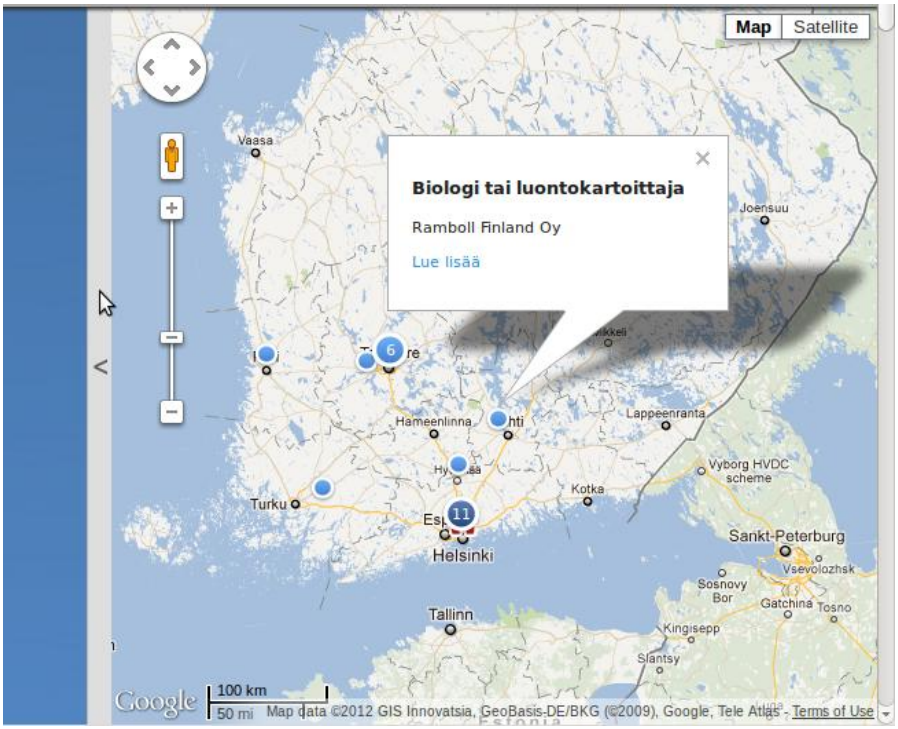
```
var options = {
  zoom: 8,
  center: new google.maps.LatLng(-34.397, 150.644),
  mapTypeId: google.maps.MapTypeId.ROADMAP
}
mapManager.map = new
google.maps.Map(document.getElementById("map_canvas"), options);
```

Esimerkkikoodi 10. Google Maps -kartan alustaminen.

Aina, kun hakukriteerejä muutetaan, tehdään tavallisen hakulistauksen lisäksi niin kutsuttu klusterointikysely. Vastauksena tähän pyyntöön palvelin palauttaa työpaikat sijainnin mukaan järjestettynä eli klusteroituna, jolloin lähekkäin olevat työpaikat liite-

tään yhdeksi pinniksi. Klusteroinnin ansiosta kartta pysyy selkeänä, kun pinnit eivät ole toistensa päällä. JavaScript-koodi käy läpi tämän palautetun JSON-muotoisen tietorakenteen, jonka mukaan se asettaa pinnit kartalle.

Klusterointikysely palauttaa ainoastaan työpaikkojen yksilöivät tunnukset eli id-numerot, joita verrataan hakukyselystä saatuun laajempaan tietorakenteeseen, joka sisältää työpaikkojen kaikki tiedot. Näiden tietojen perusteella jokaiseen pinniin saadaan lisättyä tietoikkuna, joka avautuu pinnejä hiirellä valitsemalla. Kuten kuvassa 12 näkyy, tietoikkuna sisältää tiedot työnantajasta ja haettavasta positiosta. Mikäli samassa pinnissä on klusterina useita työpaikkoja, jokaisen työpaikan tiedot listataan tietoikkunaan allekkain. Tämän ominaisuuden ansiosta työpaikkoja voidaan selata myös pelkästään karttaa käyttäen.



Kuva 12. Google Maps -pohjainen pinnejä ja tietoikkunoita sisältävä karttaelementti.

Tietoikkunoissa on lisäksi linkit, joita painamalla yksittäinen työpaikkailmoitus avautuu. Kartan dynaamisuutta lisää vielä se, että käyttäjän liikuttaessa hiirtä pinnien päällä työpaikkalistauksessa vastaavien työpaikkojen otsikot värjäytyvät, jolloin käyttäjän on helppo huomata, mitä listauksen työpaikoista mikäkin pinni havainnollistaa. Sama toimii myös päinvastoin, eli kun hiirtä liikuttaa listauksessa olevien työpaikkaelementtien ylitse, aktivoituvat kyseisten työpaikkojen pinnit kartalla. Aktivoituminen on esitetty vaih-

tamalla pinnin väri sinisestä huomiota herättäväksi punaiseksi. Pinnit aktivoituvat myös silloin, kun vastaava työpaikka on avattuna yksittäiseen työpaikkanäkymään.

Kartta-elementin koon muuttaminen on toteutettu käyttäen jQuery UI -kirjastoa. Kirjaston Resizable-lisäosa antaa käyttäjille mahdollisuuden muuttaa sivustolla olevien elementtien kokoa dynaamisesti. Jotta ominaisuus toimisi yhdessä kartan kanssa, täytyy kartta aina päivittää tapahtuman jälkeen. Tämä ei kuitenkaan ole hankalaa, sillä Resizable tarjoaa tapahtumankäsittelyfunktion, joka suoritetaan aina koon muuttamisen jälkeen. Tähän funktioon asetetaan kartan päivittämiseen vaaditut funktiokutsut, jolloin kartta päivittyy ja venyy halutun kokoiseksi. Käyttäjä voi muuttaa kokoa tarttumalla hiirellä kiinni kuvassa 12 näkyvään, kartan vasemmassa laidassa olevaan palkkiin ja liikuttamalla hiirtä vaakasuunnassa.

6 Yhteenveto

Verkkopalvelut kehittyvät jatkuvasti yhä enemmän työpöytäsovellusten kaltaisiksi työkaluiksi. Joskus tulevaisuudessa niiden välillä tuskin on lainkaan eroa. Dynaamisen verkkopalvelun toteutus ei ole vaikeaa, mutta mitä monimutkaisemmiksi palvelut käyvät, sitä tarkemmin niiden rakenne ja toteutustapa on mietittävä. JavaScript on kielenä tärkeä osa verkkopalveluiden kehitystä varsinkin nyt, kun dynaamiset Ajax-pohjaiset palvelut ovat varteenotettava vaihtoehto muille tekniikoille. JavaScriptiä ja sen mahdollisuuksia kuitenkin yhä vähätellään sen menneisyyden vuoksi, eikä sille esimerkiksi ole varattu nykyisessä Metropolian mediatekniikan koulutusohjelmassa sen ansaitsemaa läpikäyntiä.

Erittäin tärkeää JavaScript-kehityksessä on koodin erottaminen sisällöstä ja sen suunnitelmallinen rakentaminen erillisistä moduuleista. Huolellinen koodin testaaminen ja käyttöliittymän käytettävyystestaus ovat osa kehitystyötä. JavaScript-kehittäminen on nykyisin helppoa, sillä käytettävissä on valtava määrä erilaisia kirjastoja ja liitännäisiä, jotka tarjoavat ratkaisun lähestulkoon kaikkiin vastaantuleviin ongelmiin. Vaikka monet JavaScript-kehitykseen liittyvät ongelmat on jo ratkaistu, on kehityksessä vielä haasteita. Täydellistä selainyhteensopivuutta tuskin saadaan koskaan, sillä erot eri alustojen välillä ovat suuret. Kehittäjän tehtäväksi jää valita paras kompromissi suoritusnopeuden ja ominaisuuksien sekä käytettävyyden ja näyttävyyden väliltä.

Itse insinöörityöprojekti onnistui hyvin, ja työnhaun verkkopalvelun käyttöliittymän kaikki toivotut ominaisuudet pystyttiin toteuttamaan. Joitakin ongelmia ja suunnitteluvirheitä kuitenkin esiintyi, jotka vaikuttavat palvelun suoritusnopeuteen ja sitä kautta käytettävyyteen. Virheet liittyvät pääasiassa koodin rakenteeseen ja sen optimointiin. Koodin rakenne ei aivan seuraa luvussa 3.2 esiteltyä modulaarista suunnittelumallia, minkä seurauksena funktiot ja muuttujat ovat saatavilla globaalissa nimiavaruudessa. Myös joidenkin funktioiden alustamisessa toimittiin väärin, käyttäen konstruktiometodia ilmoitusmetodien sijaan. Nämä rakenteelliset ominaisuudet voivat aiheuttaa hitautta koodin suorittamisessa. Ne eivät kuitenkaan vaikuta palvelun toimivuuteen.

Raskaista lisäkirjastoista luopumalla ja hakutuloksia rajaamalla saatiin parannettua palvelun suorituskykyä huomattavasti. Parannettavaa suorituskyvyssä on kuitenkin edel-

leen. Vaikka tiedostot minimoidaan ja pakataan, palvelun latausvaiheessa tehdään vielä liikaa erillisiä HTTP-kutsuja. Onneksi tiedostojen yhdistäminen onnistuu myös jälkikäteen. Optimointi olisi kuitenkin pitänyt ottaa tiukemmin mukaan palvelun suunnitteluun jo alusta asti.

Googlen mainosten lisääminen sivulle aiheutti ongelmia, eivätkä mainokset jostain syystä aina näy kunnolla. Ongelmana on se, ettei Google varsinaisesti tue mainosten asettelua asynkronisen sisällön joukkoon, joten asialle ei ole paljoakaan tehtävissä. Optimaalisinta olisi myös ollut, jos navigaatio toimisi myös eri hakujen välillä. Tilapäinen ratkaisu sen pois jättämisestä on kuitenkin hyvä, kunnes ongelmaan löydetään tyydyttävä ratkaisu. Tämänkaltaiset ongelmat ovat vielä melko yleisiä Ajax-sovelluksia toteutettaessa, kun asynkronisia ratkaisuja ei vielä kaikkialla tueta.

Muilta osin JobSqr-palvelun käyttöliittymä ja halutut ominaisuudet toimivat aivan kuten suunniteltiin. Erilliset komponentit, kuten palautteen lähettäminen, sisäänkirjautuminen ja Facebook-kontaktihaku, on myös jo onnistuneesti liitetty osaksi yrityksen tärkeintä työnhakupalvelua Duunitoria, joka on Suomen suurin karttapohjainen työnhakupalvelu. JobSqr-palvelun beta-versio julkaistiin syyskuussa 2011. JobSqrin muitakin ominaisuuksia yhdistetään tulevaisuudessa Duunitoriin, jolloin käyttäjät pääsevät nauttimaan molempien palveluiden parhaista ominaisuuksista yhtä aikaa.

Kuten palvelun toteutusvaiheessa kävi ilmi, ei kaikkien insinöörityöraportissa läpikäytyjen suunnitteluohjeiden käytäntöön pano ole aivan helppoa, varsinkin kun parhaat käytännöt hieman vaihtelevat palvelusta riippuen. Useat aiemmin epäselvyyttä aiheuttaneet kohdat ovat kuitenkin nyt selkeitä, ja dynaamisen verkkopalvelun tehokas toteutusprosessi on kokonaisuudessaan selvillä.

Lähteet

1 Asleson, Ryan & Scutta, Nathanael, T. 2007. Ajax - Tehokas hallinta. Helsinki: Readme.fi

2 Zakas, Nicholas, C., McPeak, Jeremy, Fawcett, Joe. 2007. Professional Ajax, 2nd Edition. Indianapolis: Wiley Publishing.

3 Garrett, Jesse James. 2005. Ajax: A New Approach to Web Applications. Verkkodokumentti. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>. 18.2.2005. Luettu 15.2.2012.

4 MacCaw, Alex. 2011. Asynchronous UIs - the future of web user interfaces. Verkkodokumentti. < http://alexmaccaw.co.uk/posts/async_ui >. 16.11.2011. Luettu 20.3.2012.

5 Lingham, Vinny. 2007. Top 20 Reasons why Web Apps are Superior to Desktop Apps. Verkkodokumentti. 8.2.2007. Luettu 20.3.2012.

6 HTML & CSS. Verkkodokumentti. The World Wide Web Consortium. <http://www.w3.org/standards/webdesign/htmlcss>. Luettu 15.3.2012.

7 XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). 2002. Verkkodokumentti. The World Wide Web Consortium. <http://www.w3.org/TR/xhtml1/>. 1.8.2002. Luettu 15.3.2012.

8 Le Hégaret, Philippe. 2002. The W3C Document Object Model (DOM). Verkkodokumentti. <http://www.w3.org/2002/07/26-dom-article.html>. 2002. Luettu 15.3.2012.

9 Negrino, Tom & Smith, Dori. 2007. JavaScript - Tehokas hallinta. Helsinki: Readme.fi

10 HTML5. 2012. Verkkodokumentti. The World Wide Web Consortium. <http://dev.w3.org/html5/spec/single-page.html>. 29.3.2012. Luettu 1.4.2012.

11 Mahemoff, Michael. 2006. CSS: The Tech Ajax Forgot. Verkkodokumentti <http://softwareas.com/css-the-tech-ajax-forgot>. 31.7.2006. Luettu 20.3.2012.

12 JavaScript Overview. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en/JavaScript/Guide/JavaScript_Overview>. Luettu 15.3.2012.

13 Holzner, Steven. 2007. Ajax Bible. Indianapolis: Wiley Publishing.

14 Walsh, David. 2007. 6 Reasons To Use JavaScript Libraries & Frameworks. Verkkodokumentti. <http://davidwalsh.name/6-reasons-to-use-javascript-libraries-frameworks>. 5.9.2007. Luettu 15.3.2012.

15 How to Choose a Right JavaScript Framework. 2010. Verkkodokumentti. Design Reviver. <http://designreviver.com/tips/how-to-choose-a-right-javascript-framework/>. 31.3.2010. Luettu 15.3.2012.

16 JavaScript Frameworks. Verkkodokumentti. Wappalyzer. <http://wappalyzer.com/categories/javascript-frameworks>. Luettu 15.3.2012.

17 jQuery.get().Verkkodokumentti. jQuery API Documentation. <http://api.jquery.com/jQuery.get/>. Luettu 17.3.2012.

18 Lauriat, Shawn, M. 2008. Advanced Ajax. Massachusetts: Pearson Education.

19 Govella, Austin. 2005. Best Practices: Implementing javascript for rich internet applications. Verkkodokumentti. <http://thinkingandmaking.com/entries/63>. 27.7.2005. Luettu 26.3.2012.

20 24 Best Practices for AJAX Implementations. 2010. Verkkodokumentti. Nettuts+. <http://net.tutsplus.com/tutorials/javascript-ajax/24-best-practices-for-ajax-implementations/>. 10.12.2010. Luettu 26.3.2012.

21 Introduction to Ajax. Verkkodokumentti. About.com. <http://javascript.about.com/library/blajax11.htm>. Luettu 26.3.2012.

22 Landau, Brian. 2009. Benchmarking Javascript Templating Libraries. Verkkodokumentti. <http://viget.com/extend/benchmarking-javascript-templating-libraries>. 8.12.2009. Luettu 26.3.2012.

23 Osmani, Addy. 2011. Essential JavaScript Namespacing Patterns. Verkkodokumentti. <http://addyosmani.com/blog/essential-js-namespacing/>. 23.9.2011. Luettu 22.3.2012.

24 Rauschmayer, Axel. 2011. Patterns for modules and namespaces in JavaScript. Verkkodokumentti. <http://www.2ality.com/2011/04/modules-and-namespaces-in-javascript.html>. 19.11.2011. Luettu 22.3.2012.

25 Osmani, Addy. 2011. Patterns For Large-Scale JavaScript Application Architecture. Verkkodokumentti. <http://addyosmani.com/largescalejavascript/>. 4.9.2011. Luettu 23.3.2012.

26 Functions and function scope. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en/JavaScript/Reference/Functions_and_function_scope>. Luettu 25.3.2012.

27 Resig, John. 2009. Understanding JavaScript Testing. Verkkodokumentti. <http://fronteers.nl/congres/2009/sessions/understanding-javascript-testing>. 12.2.2009. Luettu 27.3.2012.

28 Unit testing. Verkkodokumentti. TechTarget. <http://searchsoftwarequality.techtarget.com/definition/unit-testing>. Luettu 26.3.2012.

29 Why web performance matters: is your site driving customers away. 2011. Verkkodokumentti. Gomez. <http://www.gomez.com/pdfs/wp_why_web_performance_matters.pdf>. Luettu 26.3.2012.

30 Gilbertson, Scott. 2010. How to Speed Up Your Site With YSlow and Page Speed. Verkkodokumentti. <http://www.webmonkey.com/2010/09/how-to-speed-up-your-site-with-yslow-and-page-speed/>. 1.9.2010. Luettu 27.3.2012.

31 Best Practices for Speeding Up Your Web Site. Verkkodokumentti. Yahoo! Developer Network. <http://developer.yahoo.com/performance/rules.html>. Luettu 27.3.2012.

32 Jackdon, Willie. Performance Unleashed: How To Optimize Websites and WordPress For Speed. Verkkodokumentti. <http://diythemes.com/thesis/improve-website-pagespeed/>. Luettu 27.3.2012.

33 Buffone, Robert. 2008. Dealing with Large Data in Ajax. Verkkodokumentti. <http://css.dzone.com/articles/dealing-large-data-ajax>. 16.3.2008. Luettu 27.3.2012.

34 Lazar, Jonathan. 2006. Web Usability. Pearson Education.

35 Nielsen, Jakob & Loranger, Hoa. 2006. Prioritizing Web Usability. CA: New Riders.

36 Chopra, Paras. 2010. The Ultimate Guide To A/B Testing. Verkkodokumentti. <http://www.smashingmagazine.com/2010/06/24/the-ultimate-guide-to-a-b-testing/>. 24.6.2010. Luettu 26.3.2012.

37 Mitä on web-analytiikka? Verkkodokumentti. Analytics.fi. <http://www.analytics.fi/mita-on-web-analytiikka/>. Luettu 26.3.2012.

38 Creating AJAX websites based on anchor navigation. 2008. Verkkodokumentti. Yensdesign. <http://yensdesign.com/2008/11/creating-ajax-websites-based-on-anchor-navigation/>. 26.11.2008. Luettu 20.3.2012.

39 Smarty, Ann. 2008. SEO Best Practices for URL Structure. Verkkodokumentti. <http://www.searchenginejournal.com/seo-best-practices-for-url-structure/7216/>. 3.7.2008. Luettu 28.3.2012.

40 Patel, Viral. 2008. Ajax Style File Uploading using Hidden iFrame. Verkkodokumentti <Ajax Style File Uploading using Hidden iFrame>. 18.11.2008. Luettu 28.3.2012

41 File API. 2011. Verkkodokumentti. The World Wide Web Consortium. <http://www.w3.org/TR/FileAPI/>. 20.10.2012. Luettu 22.3.2012.

42 JavaScript SDK. Verkkodokumentti. Facebook. <https://developers.facebook.com/docs/reference/javascript/>. Luettu 29.3.2012.

43 Facebook Platform Policies.Verkkodokumentti. Facebook. <https://developers.facebook.com/policy/>. Luettu 29.3.2012.