jQuery-ohjelmakirjaston käyttäminen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Mediatekniikan koulutusohjelma

Riihimäki/Syksy 2014

Pete Nykänen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

RIIHIMÄKI
Mediatekniikan koulutusohjelma
Ohjelmointi

Tekijä Pete Nykänen **Vuosi** 2014

Työn nimi jQuery-ohjelmakirjaston käyttäminen

TIIVISTELMÄ

Tämä opinnäytetyö perehtyi JavaScript -ohjelmakirjasto jQueryn käyttämiseen webohjelmointiprojekteissa. jQuery-ohjelmakirjaston ominaisuuksien ohella opinnäytetyössä sivutaan myös HTML-rakenteita, JavaScript-tekniikoita ja muita webohjelmoinnin periaatteita yleisesti. Työn tavoitteena on tuoda esille jQueryn ominaisuudet, sen hyvät ja huonot puolet sekä sen oikeaoppinen käyttäminen. Opinnäytetyö pyrkii myös vastaamaan, milloin jQueryä tulisi käyttää. Vaikka opinnäytetyö olettaa, että lukijalla on pääpiirteinen kuva JavaScriptin käyttämisestä, käydään tärkeimpiä aiheeseen liittyviä asioita läpi hieman tarkemmin.

Työssä sovelletaan teoriaa ja käytäntöä vuoropuhelun tavoin. Työssä olevat ohjelmakoodit ovat toimia, lyhyitä koodiesimerkkejä jotka saattavat tarvita lisäksi yksinkertaisia HTML-rakenteita, joita opinnäytetyöstä ei ole. Työvälineinä käytettiin sen hetkisiä verkkoselaimia (Google Chrome 36+, Mozilla Firefox 31+, Internet Explorer 11) ja avoimen lähdekoodin koodieditoria Bracketsia. Aineistona toimi pääasiassa jQueryn oma dokumentaatio ja oppimisportaali sekä jonkin verran muuta aiheiseen liittyvää kirjallisuutta.

Työn johtopäätöksenä todetaan, että jQuery on äärimmäisen hyödyllinen JavaScript-kirjasto, joka ei kuitenkaan automaattisesti tee koodista laadukasta. Opinnäytetyön tilaajana toimi Hämeen ammattikorkeakoulun Mediatekniikan verstas.

Avainsanat HTML, JavaScript, web, ohjelmointi

Sivut 45 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Degree Programme in Media Technology
Programming

Author Pete Nykänen **Year** 2014

Subject of Bachelor's thesis Using jQuery – a small, fast and feature-rich JavaScript library

ABSTRACT

This thesis focuses on jQuery, a small, fast and feature-rich JavaScript-library. It also touches on other subjects related to web-programming, such as HTML, basic JavaScript techniques and general guidelines on how to write good, clean and proficient JavaScript. The objective of this thesis is to introduce jQuery's features, uncover its good and bad sides and show best practices of using jQuery. Even though this theses assumes that the reader has basic understanding of JavaScript's main concepts, it also walks through some of the vital parts of the subject.

The thesis mixes theory and practical use in a dialogue-like fashion. The JavaScript code samples found in the thesis are usable, working pieces of code, but they might need some simple HTML-structures that are not present in this thesis. Tools used in this thesis were the current web browsers (Google Chrome 36+, Mozilla Firefox 31+, Internet Explorer 11) and the open-source code editor Brackets. The material used to produce the theoretical parts is mostly from jQuery's own documentation and learning portal, but also features some related literature.

The bottom line of this thesis is that jQuery is an extremely useful JavaScript-library that doesn't automatically guarantee good, clean and proficient code. This thesis was commissioned by the Media Workshop at HAMK University of Applied Sciences.

Keywords HTML, JavaScript, web, programming

Pages 45 p.

SISÄLLYS

TERMISTÖ

Ajax	Asynchronous JavaScript and XML, ohjelmistotekniikka joka on tarkoitettu asynkronisten web-applikaatioiden luomiseen.
AngularJS	Googlen kehittämä suosittu JavaScript MVC-kirjasto
Array	Joukko jonomuotoista dataa.
BackboneJS	Suosittu JavaScript-kirjasto, joka mahdollistaa mallien, näkymien, kokoelmien ja tapahtumien yhdistämisen.
Boolean	Muuttuja, jonka arvo on tosi tai epätosi.
Bower	Pakettimanageri, joka helpottaa webprojektien hallintaa ja ylläpitoa.
Brackets	Adoben alullepanema vapaan lähdekoodin koodieditori.
Bugi	Ohjelmakoodista löytyvä virhe.
Callback	Funktio, joka ajetaan kun funktio tai metodi on suoritettu loppuun.
camelCase	Tekstin esittämismuoto, jossa sanat ovat kirjoitettu yhteen siten että ensimmäisen sanan ensimmäinen kirjain on pieni ja muiden kapitalisoitu.
CommonJS	Ryhmä, joka tähtää JavaScript ekosysteemin luomiseen webpalvelimille, työpöydille, komentoriviohjelmille sekä selaimille.
Creative Commons	Voittoa tavoittelematon järjestö, joka tarjoaa erilaisia tekijänoikeuslakeihin liittyviä lisenssejä.
CSS	Cascading Style Sheets, WWW-dokumenteille luotu tyylien merkitsemistapa.
Debug	Ohjelmakoodista löytyvien virheiden etsimistä ja korjaamista kuvaava termi.
Deferred-objekti	Deferred-objekti on jQueryn objekti, joka auttaa asynkronisten operaatioiden ajoittamista.
DOM	Document Object Model, rajapinta jonka avulla voidaan näyttää ja muokata HTML, SVG ja XML dokumentteja.

DRY	Don’t Repeat Yourself-periaate, periaate jossa kehotetaan välttämään useasti toistuvaa ohjelmakoodia.
ECMAScript	Standardoitu skriptikieli johon myös JavaScript perustuu.
EmberJS	Malleihin perustuva JavaScript-ohjelmistokehys.
Event	Selaimessa tapahtuva tapahtuma, jonka on laukaissut käyttäjä tai ohjelmakoodi.
Event handler	Tapahtumien käsittelijä eli funktio, joka määrittää miten tapahtumia tulisi käsitellä.
Funktio	Osa ohjelmaa, joka suorittaa tietyn toiminnon.
Git	Versionhallintajärjestelmä.
Google	Kansainvälinen yhtiö.
Google Chrome	Googlen kehittämä ilmainen verkkoselain.
HTML	Hyper Text Markup Language, internetin sisällön merkitsemistapa.
Internet Explorer	Microsoftin verkkoselain.
Java	Oraclen ohjelmointikieli.
JavaScript	Moniparadigmainen ohjelmointikieli, jota käytetään pääsiassa WWW-sivuilla.
jQuery	Yksi suosituimmista JavaScript-ohjelmakirjastoista.
jQuery Foundation	jQuery-ohjelmakirjastoa kehittävä yhdistys.
jQuery UI	jQueryn käyttöliittymäkirjasto.
JSON	JavaScript Object Notation, pienikokoinen datansiirto formaatti, joka muistuttaa JavaScript objektia.
Liitännäinen	Selaimeen tai ohjelmakoodiin lisätty osa joka tuo koodiin lisää ominaisuuksia tai muokkaa olemassa olevaa toiminnollisuutta.
MaxCDN	Yksi suurimmista sisällöntoimitusverkkojen tarjoajista.
Metodi	Objektiin arvo, joka on funktio.
Microsoft	Kansainvälinen yhtiö.
Minified	Kevyt JavaScript-kirjasto.
MIT	Massachusetts Institute of Technology, tunnettu teknillinen korkeakoulu.
MIT-lisenssi	MIT-lisenssi on MIT:ssä kehitetty vapaa ohjelmistolisenssi, joka sallii teoksen käytön myös kaupallisissa

	suljetun lähdekoodin ohjelmistoissa.
MooTools	JavaScript-ohjelmistokehys.
Mozilla Firefox	Mozilla-yhteisön tuottama ilmainen verkkoselain.
MVC	Model-View-Controller, tapa jossa ohjelmisto on jaettu malliin (data), näkymään ja kontrolleriin (toiminnollisuus).
Objekti	Datapaketti, joka koostuu nimi-arvo-pareista.
Ohjelmakirjasto (Library)	Kokoelma ohjelmakoodia, luokkia, aliohjelmia tai ohjelmia joita käytetään ohjelman suorittamisen aikana.
Ohjelmistokehys (Framework)	Ohjelmoinnin apuväline, jonka päälle voidaan rakentaa tietokoneohjelma.
Ohjelmointirajapinta eli API	Määritelmä, jonka mukaan eri ohjelmat voivat keskustella keskenään.
Olio-ohjelmointi	Ohjelmoinnin lähestymistapa, jossa ohjelmakoodi koostuu olioista eli objekteista.
Promise/A	CommonJS:n ehdotus lupaus-mallista, joka määrittää kuinka asynronisten operaatioiden kanssa tulee toimia.
Same-origin policy	Konsepti, jossa sallitaan skriptien ajaminen vain saman protokollan, verkkonimen ja portin välillä.
Shadow DOM	HTML5-standardin tekniikka, jossa renderöidyn DOM:in pohjalla on toinen rakenne Shadow DOM johon voidaan piilottaa DOM-alapuita.
Sisällöntoimitusverkko (CDN)	Sisällöntoimitusverkot (CDN, Content Delivery Network) ovat verkkoja, jossa sisällönjakopalvelimet on hajautettu ympäri maailmaa.
Skripti- eli komentosarjakieli	Kieli, jolla kirjoitetaan komentosarjoja. Tarkoittaa nykyään kieltä, joita voidaan suorittaa ilman kääntämistä.
SVG	XML:llään pohjautuva vektorigrafiikkaformaatti.
Säännöllinen lauseke	Tietojenkäsittelyteorian lauseke, jonka avulla voidaan tarkistaa ja muokata merkkijonoja. Englanniksi Regular Expression, yleensä lyhennettynä RegExp tai Regex.
URL	Uniform Resource Locator, osa tietynlaista merkkijonoa, joka viittaa

	resurssiin. Tunnetaan myös nimellä verkko-osoite.
Verkkoselain	Ohjelma jolla voidaan selata WWW-sivuja.
Widget Factory	jQuery UI:n liitännäissysteemi.
WWW	World Wide Web, internetissä toimiva hypertekstijärjestelmä.
XML	Extensible Markup Language, tapa esittää sisältöä muodossa jota pystyy lukemaan sekä ihmiset että koneet.
XMLHttpRequest (XHR)	Tapa joka mahdollistaa sivun kommunikoinnin palvelimen kanssa ilman sivun uudelleenlatausta.
ZeptoJS	Kevyt JavaScript-kirjasto.

1 JOHDANTO

JavaScriptistä on tullut aikojen saatossa yksi suosituimmista ohjelmointikielistä. Syy sen suosioon on selvä: sitä käytetään lähestulkoon jokaisella WWW-sivulla ja käytännössä jokaisessa tietokoneessa on sen tulkintaan käytettävä ohjelma, esimerkiksi verkkoselain. Sen suosio on luonut pohjan kattaville JavaScript-ohjelmakirjastoille (library) ja ohjelmistokehyksille (framework). Ohjelmakirjastojen ja ohjelmistokehyksien tarkoitus on auttaa kehittäjää antamalla hänelle työkaluja, joiden avulla tiettyjen tehtävien tekeminen on kehittäjälle yksinkertaisempaan. Vaikka käyttötarkoitus saattaisi muuttua, kirjastojen toteutus yleensä ei: tämä mahdollistaa koodin uudelleenkäyttämisen eri ohjelmien välillä. (Crockford 2008; jQuery n.d.)

JavaScript-ohjelmakirjastoista yksi suosituimmista on jQuery. Se on alustariippumaton, avoimen lähdekoodin kirjasto, jonka tarkoituksena on yksinkertaistaa monia webohjelmointiin liittyviä osa-alueita kuten

- Interaktiivista sisällönmuokkausta
- Ulkoasun muokkausta
- Tapahtuminen käsittelyä
- Animaatioita
- sekä Ajax:in (asynchronous JavaScript and XML) eli asynkronisten JavaScript ja XML -tapahtumien hallintaa jota käytetään internetissä datan hakemiseen ja lähettämiseen palvelimelta

Tämä tapahtuu ohjelmointirajapinnan eli API:n kautta. Ohjelmointirajapinta tarkoittaa tapaa jolla kehittäjä voi käyttää ohjelmakirjaston ominaisuuksia eli ”keskustella” ohjelmakirjaston kanssa. jQuery on tehty täysin JavaScriptillä, eikä näin ollen tarvitse selainliitännäisiä tai muita selainten ulkopuolisia ohjelmia. jQueryä kehittää jQuery Foundation -yhdistys, johon kuuluu laaja joukko sekä yksityishenkilöitä että yrityksiä. jQueryn ensimmäinen vakaa versio julkaistiin elokuussa 2006 ja vuonna 2014 jQueryä käyttää noin 61 % kaikista internetin sivuista eli noin 94 % kaikista JavaScript-kirjastoa käyttävistä sivuista. (jQuery n.d.a; Resig 2006; W3Techs 2014)

Tässä opinnäytetyössä käydään lävitse jQueryn käyttöönotto, perusomaisuudet sekä sen laajentaminen jQuery -liitännäisten kautta. Opinnäytetyö pyrkii selvittämään miksi, milloin ja miten tulisi webkehittäjän käyttää jQueryä webohjelmassaan. Opinnäytetyön tavoitteena on antaa opinnäytetyön tilaajan, Hämeen ammattikorkeakoulun Riihimäen-yksikön Mediaverstaan, työntekijälle jQueryn tehokkaan hyödyntämisen perustaidot. Opinnäytetyössä olevat JavaScript- ja jQuery -esimerkit ovat kokonaisia ja toimivia, mutta saattavat tarvita ylimääräistä HTML -koodia toimiakseen.

Opinnäytetyössä on käytetty työvälineenä yleisimpien selaimien sen hetkisiä versioita (Google Chrome 36+, Mozilla Firefox 31+, Internet Explorer 11) ja ilmasta, avoimen lähdekoodin koodieditoria Bracketsia. Opinnäytetyössä käytetyt termit ovat kuitenkin universaaleja, eivätkä ne rajoitu tiettyyn ohjelmaan tai ohjelmaversioon.

2 ENNEN JQUERYN KÄYTTÖÖNOTTAMISTA

2.1 Websivuista yleisesti

Ennen jQueryn käyttöönottamaista, tulisi käyttäjällä olla perustietämys internet-sivujen toiminnasta. Tämän selvittämiseksi käydään ensiksi läpi tavallisen WWW-sivun anatomia. Nykyaikainen WWW-sivu koostuu kolmesta osasta: HTML:stä, CSS:stä ja JavaScript -koodista. (jQuery Learning Center n.d.a)

HTML:ää (Hyper Text Markup Language) käytetään sisällön esittämiseen eli toisin sanottuna websivut on kirjoitettu käyttäen HTML:ää. Se on semanttista esitystapaa, eli sivuilla olevilla elementeillä on tietty merkitys. Elementit voivat esimerkiksi olla otsikoita, leipätekstiä, kuvia ja niin edelleen. (jQuery Learning Center n.d.a)

CSS:ää (Cascading Style Sheets) käytetään antamaan HTML-dokumenteille ulkoasu. Sen tarkoituksena on saada sisältö näyttämään paremmalta määrittämällä sille fontteja, tyylejä ja muita ulkoasun tehokeinoja. Sen tehokkuus perustuu siihen, että tyylit voidaan erotella sisällöstä. Näin ollen samoja tyylejä voidaan käyttää monien eri sisältöjen kanssa, mikä on tärkeää kun halutaan rakentaa monipuolisia sivuja jotka toimivat useiden eri laitteiden kanssa. (jQuery Learning Center n.d.a)

JavaScriptiä käytetään interaktiivisuuteen. Ilman JavaScriptiä sivut olisivat yksinkertaisia, staattisia ja tylsiä. JavaScript auttaa tuomaan sivut eloon. (jQuery Learning Center n.d.a)

2.2 JavaScriptistä yleisesti

Vaikka JavaScriptistä on tullut aikojen saatossa yksi suosituimmista ohjelmointikielistä, se on myös herättänyt paljon hämmennystä ja jopa halveksintaa. Jo sen nimi, JavaScript, sekoittuu helposti Java-ohjelmointikieleen, vaikka nämä ovat kaksi täysin eri kieltä. Sen ensimmäiset versiot olivat ominaisuuksiltaan heikkoja, mikä muodostaa vääriä mielikuvia edelleen. Myöskään vanhojen selainten bugisuus, aiheeseen liittyvän kirjallisuuden heikko laatu ja vaikeaselkoinen standardi eivät auttaneet asiaa. Vaikka se on alun perin luotu selaimessa käytettäväksi skriptikieleksi, on JavaScript nykyään täysimittainen olio-ohjelmointikieli, jota käytetään myös palvelinpuolen sovelluksissa. (Crockford 2001a; jQuery Learning Center n.d.a)

Ohjelmointi perustuu JavaScriptissä objekteihin. Objektit ovat nimi-arvopareja. Nimet ovat merkkijonoja, arvot ovat merkkijonoja, numeroita, boolean-arvoja (tosia / epätosia) tai toisia objekteja kuten arvojoukkoja (array) tai funktioita. Jos objektin arvo on funktio, sitä sanotaan metodiksi. Kun

objektin metodia kutsutaan, muuttuja ”this” asetetaan objektiin. Metodi voi näin ollen päästä käsiksi instanssin omiin muuttujiin ”this”-muuttujan kautta. (Crockford 2001b)

Tämän sisäistäminen on oleellista jQueryn käyttämisen kanssa. jQuery-metodit palauttavat aina jQuery-objektin. Tähän konseptiin perehdytään tarkemmin kappaleessa 5, jQuery API.

2.3 JavaScriptin ja jQueryn kirjoittamisen työkalut

Jotta JavaScriptiä ja näin ollen jQueryä voidaan kirjoittaa, tarvitsee kehittäjä vähintään seuraavat työkalut

- Verkkoselain
- Tekstieditori

Äärimmäisen hyödyllisiä ovat myös niin sanotut kehittäjän työkalut, joita löytyy nykyään jokaisesta selaimesta. Kehittäjän työkalut auttavat kehittäjää debuggaamaan eli testaamaan ja korjaamaan ohjelmakoodista löytyviä virheitä. (jQuery Learning Center n.d.a)

Vaikka koodia voidaan kirjoittaa yksinkertaisilla tekstieditorilla kuten Windowsin Notepadilla, auttaa kehittyneempi teksti- tai koodieditori kehittäjää luomaan koodia esimerkiksi syntaksikorostuksen avulla.

3 JQUERYSTÄ YLEISESTI

Kuten aiemmin mainittiin, jQuery on JavaScriptillä kirjoitettu ohjelmakirjasto, jonka tehtävänä on yksinkertaistaa monia moderneissa websivuissa käytätettyjä tekniikoita. Se on nopea, pieni ja monipuolinen kirjasto, jonka yhdistelmä kattavuutta ja jatkettavuutta on muuttanut tapaa jolla miljoonat ihmiset kirjoittavat JavaScriptiä. (jQuery n.d.)

Miksi kehittäjän tulisi ottaa käyttöön jQuery? Kehittäjälle jQueryn käyttö tarkoittaa ohjelmistokehityksen yksinkertaistamista ja nopeuttamista marginaalisen pienen performanssihäviön kustannuksella. jQuery auttaa kehittäjää pääsemään ylitse webohjelmistokehityksen ongelmakohdista, kuten selainversioiden eroavaisuuksista ja paikoitellen vaikeaselkoisesta syntaksista. Selainversioiden eroavaisuuksia voi tarkastella esimerkiksi Can I Use -sivuston kautta. (jQuery n.d.; Can I Use n.d.)

Kuvitellaan tilanne, jossa kehittäjä haluaa sivulleen napin, jota painamalla käyttäjälle ilmoitetaan ”hei maailma”. Seuraavassa HTML-merkintä napille

```
<button id="nappi" name="nappi">Klikkaa minua</button>
```

Tyypillisesti napin klikkaukseen liittyvä JavaScript -koodi näyttää tältä:

```
var nappi = document.getElementById("nappi");
nappi.addEventListener("click",function() {alert("hei
maailma")});
```

Koodi toimii kahdessa osassa. Ensimmäisessä rivissä muuttujalle ”nappi” annetaan arvo, joka vastaa HTML-dokumentista löytyvää merkintää id:llä ”nappi”. Toisen rivin ”addEventListener()”-metodi on selaimiin sisäänrakennettu metodi, jolla käyttäjän klikkaustapahtumiin voidaan lisätä toiminnollisuutta. Se tarvitsee toimiakseen kaksi pakollista parametria: tapahtuman tyypin (”click”) sekä tapahtumaa seuraavan funktion (tässä tapauksessa anonyymi funktio, joka kutsuu selainten ”alert()” -metodia). (Mozilla Developer Network n.d.a)

Nappi toimii kehittäjän selaimessa mutta käyttäjä A valittaa, ettei nappi toimi hänen selaimessaan. Käyttäjä A:n tietokone on vanha ja sen ainoa selain on Internet Explorer 8. Internet Explorer 8:ssa ”addEventListener()”-nimistä metodia ei ole, vaan sen tilalla on vanhentunut ”attachEvent()”-metodi. Kehittäjä joutuu lisäämään tämän tarkistuksen koodiinsa, minkä jälkeen koodi näyttää seuraavalta.

```
var nappi = document.getElementById("nappi");
  if (nappi.addEventListener) {
    nappi.addEventListener("click",function(){alert("hei
maailma")}, false);
  } else if (nappi.attachEvent) {
    nappi.attachEvent("onclick",function(){alert("hei
maailma")});
}
```

Jos koodi ajetaan selaimella, joka tukee ”addEventListener()”-metodia, kiinnittää selain napin toiminnollisuuden sillä. Jos näin ei ole, käyttää selain vanhentunutta ”attachEvent()”-metodia. Nyt nappi toimii myös käyttäjän koneella, mutta koodin määrä kasvoi lähes puolella. Verrataan edellisen JavaScript-koodin toteutusta jQueryllä.

```
$("#nappi").on("click", function(){alert("hei maailma")});
```

Huomataan, että jQuery-toteutuksessa ei ole ainoastaan vähemmän koodia, se on myös selvästi helpommin luettavaisissa ja kirjoitettavissa. Sitä miten edeltävä koodi toimii, käsitellään tarkemmin kappaleessa 5, jQuery API. Varoituksen sana: yllä oleva esimerkki toimii Internet Explorer 6-8 -selaimissa vain jQueryn kehityshaarassa 1.x, tästä lisää luvussa 4 jQueryn versiot ja käyttöönotto. (jQuery n.d.; Mozilla Developer Network n.d.)

Kehittäjän etuna on myös jQueryn yleisyys: jQueryn dokumentaatio on laadukasta ja helposti ymmärrettävää. Koska jQuery on erittäin laajasti käytössä, löytyy sille laaja skaala valmiita esimerkkejä ja liitännäisiä. Se on myös yhteensopiva lähestulkoon kaikkien muiden JavaScript-ohjelmakirjastojen kanssa.

4 JQUERYN VERSIOT JA KÄYTTÖÖNOTTO

4.1 Versiot

jQueryä päivitetään säännöllisen epäsäännöllisesti. Siitä on saatavilla kahta kehityshaaraa, 1.x sekä 2.x. Kehityshaarojen erona on, ettei 2.x kehityshaara toimi lainkaan vanhoilla Internet Explorerin versioilla eli versioilla 7, 8 ja 9. Tämän lisäksi 1.x-haara sai suuria muutoksia version 1.9.0 myötä, jolloin päivitettäessä vanhoista versioista olisi jQuery Migrate -liitännäisen käyttäminen suositeltavaa. Näiden lisäksi jQuerystä on saatavilla pakkaamaton versio ja pakattu versio. (jQuery n.d.a; Methvin 2013)

Mikä näistä kehittäjän tulisi valita? Jos projekti on vanha, tulee kehittäjän ensiksi tarkastella, onko projektissa käytetty jotain jQueryn vanhempaa (< 1.9) versiota. Jos on, tulisi päivittäessä ottaa käyttöön myös edellä mainittu jQuery Migrate -liitännäinen. jQuery Migrate -liitännästä voidaan käyttää löytämään ja tuomaan takaisin API:ja tai ominaisuuksia jotka on deprikoitu ja myöhemmin poistettu jQuerystä. Pakkaamaton versio liitännäisestä varoittaa selaimen konsoliin jos vanhentuneita ominaisuuksia on käytössä, jolloin kehittäjä voi päivittää vanhentuneet metodit. Pakattu versio ei näytä näitä varoituksia, jolloin sitä voi käyttää yksinkertaisena yhteensopivuus liitännäisenä, mutta tätä suositellaan vain väliaikaiseksi ratkaisuksi. (Github n.d)

Jos projekti aloitetaan tyhjältä pohjalta tai siinä ei ole aiemmin käytetty jQuery-kirjastoa, tulee kehittäjän valita tukeeko sivu myös vanhentuneita selaimia. Valintaa miettiessä on hyvä vastata seuraaviin kysymyksiin:

- Tulevatko sivustoa käyttämään pääasiassa käyttäjät, joiden voidaan olettaa käyttävän moderneja selaimia?
- Onko vanhojen selainversioiden tukemiseen käytetty aika laskettu mukaan projektin toteutusvaiheen kestoon tai onko vanhojen selainten tukeminen ylipäätään mahdollista?

Jos kehittäjä päätyy tukemaan vanhoja selainversioita, tulee kehittäjän valita jQueryn kehityshaara 1.x. Nimestään huolimatta se sisältää samat ominaisuudet kuin haara 2.x. jonka etuna on pienempi koko ja nopeus johtuen poistetusta Internet Explorer 6-8 -tuesta. Tällöin versio 2.x.-haara tulee valita jos sivuston tähtäimenä on nopeus tai selainten modernien ominaisuuksien käyttäminen. (jQuery n.d.b)

Koska jQuerystä on saatavilla pakkaamaton ja pakattu versio, on suositeltavaa että kehittäjä lataa molemmat. Pakkaamatonta versiota tulee käyttää kehityksen aikana, koska se takaa paremmat debug-mahdollisuudet. Kun kehitystyö tai versio on valmis, tulee tämä vaihtaa pakattuun versioon. Pakatusta versiosta on muun muassa poistettu kommentit, lukemista helpottavat rivinvaihdot ja muuttujien nimet on muutettu juokseviksi kirjaimiksi (a, b, c ja niiden edelleen). Tämän vuoksi pakattu jQuery-koodi ei ole enää helposti luettavissa, mutta tietokone käsittelee koodin nopeammin ja se vaatii pienemmän kokonsa ansiosta vähemmän kaistaa palvelimelta. (jQuery n.d.b)

4.2 Käyttöönotto

jQuery voidaan ottaa käyttöön monella tavalla. Yksinkertaisin niistä on ladata jQuery tietokoneelle ja siirtää se projektikansioon. Esimerkiksi ladataan jQueryn version 1.11.1 pakattu ja pakkaamaton versio ja sijoitetaan ne projektikansion juureen. jQuery otetaan käyttöön sivustolla viittaamalla siihen ”<script>”-elementin ”src”-attribuutilla. ”<script>”-elementti voidaan sijoittaa joko ” <head>”-elementin sisälle tai ”<body>”-elementin loppuun.

```
<script src="jquery-1.11.1.js"></script>
```

Kun projekti on valmis julkaistavaksi, vaihdetaan jQuery pakatuksi versioksi yksinkertaisesti vaihtamalla siihen viittaavan elementin ”src”-attribuutti viittaamaan pakattuun versioon.

```
<script src="jquery-1.11.1.min.js"></script>
```

Toinen tapa ottaa jQuery-käyttöön käyttää sisällöntoimitusverkkoja. Sisällöntoimitusverkot eli CDN:t (Content Delivery Network) ovat verkkoja, joissa sisällönjakopalvelimet ovat hajautettu ympäri maailmaa. jQueryn virallisen CDN-verkon tarjoaa MaxCDN, minkä lisäksi jQuery on saatavilla myös esimerkiksi Googlen ja Microsoftin CDN-verkoista. CDN:n käyttämisen etuna on se, ettei sivuston käyttäjän tarvitse välttämättä ladata jQuery-kirjastoa uudelleen, jos käyttäjä on jo aiemmin ladannut kirjaston samalta CDN:ltä. Tällöin myös sivuston palveluntarjoajan kaistaa säästyy. Haittapuolena voidaan pitää sitä, että jos CDN ei ole saatavilla, ei käyttäjä pysty lataamaan kirjastoa ja sivusto ei toimi oikein. jQuery otetaan käyttöön CDN:n kautta samalla tavalla kuin lokaalisti, mutta tiedostopolun sijaan viitataan CDN:n osoitteeseen. (jQuery n.d.b)

```
<script src="//code.jquery.com/jquery-
1.11.1.min.js"></script>
```

Kaikki muut JavaScript-tiedostot jotka ovat riippuvaisia eli käyttävät jQuery-kirjastoa, tulee sijoittaa ohjelmakoodiin varsinaisen jQuery-kirjaston jälkeen. Tämä koskee myös jQuery-liitännäisiä, kuten aikaisemmin mainittu jQuery Migrate. Liitännäisiä ja niiden toimintaa käydään tarkemmin läpi kappaleessa 6, Liitännäiset ja jatkettavuus.

jQuery ja muut siihen liittyvät projektit on julkaistu MIT-lisenssin alla, mikä tarkoittaa että jQuery on käytettävissä missä tahansa projekteissa (myös kaupallisissa) sisällä ehdolla että copyright-tieto säilytään muokkaamattomana. jQueryn sivuilla olevat koodiesimerkit on julkaistu Creative Commons CC0 -lisenssin alla, mikä antaa luvan käyttää koodia missä tahansa asiayhteydessä ilman minkäänlaisia viittauksia tai copyright-merkintöjä. (jQuery n.d.c)

Edistyneempiä tapoja jQueryn käyttöönottoon on esimerkiksi asentaa se Bower-pakettimanagerilla tai rakentaa se lähdekoodista käyttäen Git-versionhallintajärjestelmää, mutta näitä tapoja ei käsitellä tässä opinnäytetyössä aiheiden laajuuden vuoksi. (jQuery n.d.b)

5 JQUERY API

Tässä luvussa tarkastellaan jQuery API:n käyttöä. jQuery API on rajapinta, jonka kautta kehittäjä voi käyttää jQueryn ominaisuuksia eli käytännössä metodeja. Näitä metodeja on satoja, joten niiden kaikkien läpikäyminen ei tämän opinnäytetyön puitteissa ole mahdollista. Kappaleen tarkoituksena on kuitenkin antaa yleiskuva jQueryn tarjoamista mahdollisuuksista sekä esitellä joitain yleisemmin käytettyjä metodeja. jQuery API on dokumentoitu yksityiskohtaisesti osoitteessa http://api.jquery.com/.

jQuery API on jaettu noin kahteenkymmeneen eri kategoriaan, joiden ominaisuuksia voi kehittäjä yhdistellä tarpeidensa mukaan. Monet metodeista kuuluvat myös useampaan eri kategoriaan. Nämä kategoriat sisältävät metodeja, jotka kattavat johdannossa esitellyt ominaisuudet interaktiivista sisällönmuokkauksesta AJAX-ominaisuuksiin. (jQuery n.d.c)

5.1 $ ja $()

Luvussa 3 olevassa jQuery-esimerkissä käytettiin $-symbolia. Monet JavaScript-kirjastot käyttävät tätä funktion tai muuttujan nimenä, kuten myös jQuery. jQueryn tapauksessa $ on alias funktiolle ”jQuery”. $-aliasta käytetään, jotta koodista tulisi helppolukuisempaa. Käytännössä tämä tarkoittaa että seuraavat rivit vastaavat toisiaan.

```
$("#nappi").remove();
jQuery("#nappi").remove();
```
(jQuery n.d.d)

Useimmat jQuery-metodit kutsutaan jQuery-objekteihin kuten yllä. Näiden metodien sanotaan olevan osa $.fn-namespacea tai ”jQuery prototyyppiä” ja niiden voidaan ajatella olevan osa jQueryn objektien metodeja. On kuitenkin olemassa metodeja, joita ei käytetä valittuihin elementteihin. Nämä metodit ovat osa jQuery-namespacea ja niitä voidaan pitää jQueryn ytimen osana. Namespace tarkoittaa nimeämiskäytäntöä, samannimiset tunnisteet voidaan erottaa ylemmän tason nimen perusteella. (jQuery Learning Center n.d.b)

Edellä mainittu erottelu voi olla haastavaa uusille jQueryn käyttäjille. Tähän apuna on olemassa kaksi muistisääntöä:

– jQueryn valitsimiin kutsutut metodit kuuluvat $.fn-namespaceen, ja saavat ja palauttavat aina valinnan ”this”-muuttujan kautta.

– Metodit, jotka ovat $- eli jQuery-namespacessa ovat yleensä apufunktioita ja ne eivät toimi valitsimien kanssa. Niille ei anneta automaattisesti argumentteja ja niiden palautusarvot vaihtelevat.

On tapauksia joissa objektimetodit ja ydinmetodit ovat samannimisiä, kuten ”$.each()” sekä ”.each()”. Tällöin tarvitsee kiinnittää erityistä huomiota dokumentaatiota tarkastellessa: tarkista aina että luet oikeaan metodiin viittaavaa dokumentaatiota. (jQuery Learning Center n.d.b)

Jatkossa tässä opinnäytetyössä viitataan objektimetodeihin nimellä ”.metodinNimi()” ja ydinmetodeihin nimellä ”$.metodinNimi()”.

5.2 $(document).ready()-metodi

Koska selaimet lataavat sivustot osissa, ei sivustoa voi muokata ”turvallisesti” ennen kuin dokumentti on ”valmis”. Oletetaan tilanne, jossa kehittäjä haluaa liittää toiminnollisuutta sivulla olevaan nappiin. Sivun latauduttua nappi ei kuitenkaan toimi halutulla tavalla. Monesti ongelma on, että sivuston JavaScript-koodi on ajettu, ennen kuin selain oli luonut kyseisen napin, eli sivusto ei ollut valmis.

jQuery tarjoaa tähän ongelmaan metodin, ”.ready()”. Kaikki koodi, joka on ”$(document).ready()”-metodin sisällä, ajetaan yhden kerran kun sivun Document Object Model eli DOM on valmis. (jQuery Learning Center n.d.b)

DOM on rajapinta HTML-, XML- sekä SVG -dokumenteille. Se tarjoaa rakenteellisen esitysmuodon dokumentista ja se määrittää tavan jolla ohjelmat voivat hyödyntää tätä rakennetta muokatakseen dokumentin rakennetta, tyyliä ja sisältöä. Käytännössä se yhdistää verkkosivut skripteihin tai ohjelmointikieliin. Verkkosivu on dokumentti, jota voidaan näyttää selaimessa tai koodieditorissa lähdekoodina, mutta se on edelleen sama dokumentti. DOM on toinen tapa näyttää, muokata ja manipuloida tuota samaa dokumenttia. jQuery on siis kirjoitettu JavaScriptillä, mutta se käyttää DOM:ia dokumentin läpikäymiseen. Toisin sanoen jos koodia ajetaan ennen kuin DOM on valmis, ei elementtiä johon koodi kohdistetaan ole vielä välttämättä olemassa. (jQuery Learning Center n.d.c; Mozilla Developer Network n.d.b)

Seuraava koodi ajetaan vasta kun DOM on valmis käytettäväksi

```
$(document).ready(function () {
  console.log("DOM valmis");
});
```

5.3 Elementtien valitseminen

jQueryn yksinkertaisin konsepti on ”valita joitain elementtejä ja tehdä jotain niille”. jQuery tukee useimpia CSS3 -valitsimia sekä muutamia ei-standardoituja valitsimia. Elementtejä voi valita esimerkiksi elementin id:n, luokan, attribuutin $, näiden yhdistelmien tai niin kutsuttujen pseudo-valitsimien perusteella. . (jQuery Learning Center n.d.d)

Tyyppi	Valitsin	Selitys
Id-valitsin	`$("#nimi")`	Valitsee elementin id:llä ”nimi”.
Luokkavalitsin	`$(".luokka")`	Valitsee kaikki elementit joilla on luokka ”luokka”.
Attribuuttivalitsin	`$("input[name='etunimi']")`	Valitsee kaikki välittömästi input-elementit, joiden ”name” attribuuttina on ”etunimi”.
Yhdistelmävalitsin	`$("#sisallys ul li")`	Valitsee li-elementit, jotka ovat ”sisallys”-id:llä olevan elementin sisällä olevien ul-elementtien sisällä
Pseudovalitsin	`$("tr:odd")`	Valitsee joka toisen tr-elementin

Taulukko 1. Yhteenveto erityyppisistä valitsimista

Miten valitsin sitten tulisi valita? Hyvän valitsimen valinta on yksi tapa parantaa JavaScriptin suorituskykyä. Valinnan tarkentaminen, kuten esimerkiksi lisäämällä elementin tyyppi valittaessa elementtejä luokan nimen perusteella voi auttaa jo paljon. Toisaalta myöskään liian tarkat valitsimet eivät välttämättä ole hyviä. Esimerkiksi ”#taulu thead tr th.special” on aivan liian tarkka kun ”#taulu th.special” tuottaa saman tuloksen. (jQuery Learning Center n.d.d)

jQuery tarjoaa myös mahdollisuuden valita elementtejä osittaisten hakutulosten perusteella käyttäen yksinkertaistettuja säännöllisiä lausekkeita (Regular Expression, lyhyesti RegEx tai RegExp). Tämä voi kuitenkin olla hidasta vanhoilla selaimilla, joten aina kun mahdollista, tulisi käyttäjän käyttää valitsimia jotka koostuvat elementtien id:istä, luokkien nimistä ja elementtien tyypeistä. Alla esimerkki valitsimesta joka valitsee kaikki luokat jotka sisältävät sanan ”mediaverstas”. (jQuery Learning Center n.d.d; jQuery n.d.e)

```
$('[id*="mediaverstas"]')
```

Monesti valintojen jälkeen halutaan tarkistaa, tuottiko valinta ainuttakaan tulosta. Yleinen virhe on käyttää valitsinta ”if”-lausekkeen sisällä, joka taas ei toimi halutulla tavalla. Kun elementtejä valitaan käyttäen ”$()”-metodia,

jQuery palauttaa aina objektin. Tyhjäkin objekti on JavaScriptissä totuusarvoltaan ”tosi”, jolloin ”if”-lausekkeen sisällä oleva koodi ajetaan vaikka valinta ei olisikaan tuottanut tulosta. Paras tapa tarkistaa valitsiko valinta yhtään elementtiä, on käyttää ”.length”-ominaisuutta. (jQuery Learning Center n.d.d)

```
//väärin
if ($("div.foo")) {
  //tämä koodi ei toimi, koska tyhjäkin objekti on ”tosi”
}

//oikein
if ($("div.foo").length)) {
  //tämä koodi ajetaan vain jos elementtejä on useampi kuin
yksi
}
```

Ohjelmakoodia tehdessä on myös muistettava, ettei jQuery talleta valintoja automaattisesti esimerkiksi välimuistiin. Käytännössä tämä tarkoittaa, että valintojen tulokset, joita tullaan käyttämään useasti, tulisi tallettaa muuttujaan.

```
var divs = $("div");
```

Tällöin muuttujaa ”divs” voidaan käyttää jatkossa samalla tapaa kuin valitsinta $(”div”) ilman että jQueryn tarvitsee valita elementtejä uudestaan. Muuttuja ei kuitenkaan päivity automaattisesti jos sivulle tulee tai siitä poistetaan valintaa vastaavia elementtejä. Käyttäjän tulee itse päivittää muuttujan arvo hakemalla elementit uudestaan. (jQuery Learning Center n.d.d)

Valintoja voi myös filtteröidä ja tarkentaa useilla jQueryn tarjoamilla metodeilla. Monesti tarkkakin valinta sisältää epätoivottuja elementtejä. Filteröintimetodien, kuten ”.has()”, ”.not()”, ”.filter()” sekä ”.eq()”, avulla voidaan valintoja karsia entisestään. Esimerkiksi seuraavassa koodinpätkässä karsitaan hakutulos sisältämään vain elementit, joilla on ”<p>”-elementtejä. (jQuery Learning Center n.d.d)

```
$("div.foo").has("p");
```

jQuery tarjoaa myös monia pseudovalitsimia ”<form>”-elementtien valitsimiseen, joiden tilojen tarkastelu on vaikeaa tavallisilla CSS3-valitsimilla. Esimerkiksi tilanteet, joissa täytyy tarkastella ovatko tietyt valintaruudut (checkbox) valittu, ovat triviaaleja käyttäen jQueryn ”:checked”-valitsinta. (jQuery Learning Center n.d.d)

5.4 Valintojen kanssa työskentely

5.4.1 Arvojen noutaminen ja asettaminen

jQuery toimii siten, että sen metodit toimivat arvojen noutajina (getter) sekä niiden asettajina (setter). Jos valittuihin elementteihin kohdistetaan metodi

antamatta sille mitään arvoja, metodi noutaa edellisen arvon (get). Jos metodille annetaan jokin arvo, se asettaa elementille kyseisen arvon (set). Seuraavassa esimerkissä käytetään ”.html()” -metodia sekä getterinä että setterinä. (jQuery Learning Center n.d.e)

```
// getter
$("h1").html();
//setter
$("h1").html("hei maailma")
```

Setterit palauttavat jQuery-objektin, jolloin samaan valintaan voi kutsua perättäin monta jQuery-metodia. Tätä kutsutaan ketjuttamiseksi. Sen sijaan getterit palauttavat arvon jota niiltä pyydettiin, joten näiden kanssa ketjuttaminen ei toimi. (jQuery Learning Center n.d.e)

Alla yleisimmin käytettyjä metodeja joita voidaan käyttää asettamaan ja noutamaan arvoja.

Metodi	Mitä tekee
`.html()`	Hakee tai asettaa HTML-sisältöä.
`.text()`	Hakee tai asettaa tekstisisältöä poistaen HTML-merkinnät.
`.attr()`	Hakee tai asettaa attribuutin.
`.width()`	Hakee tai asettaa ensimmäisen elementin korkeuden.
`.height()`	Hakee tai asettaa ensimmäisen elementin korkeuden.
`.position()`	Hakee ensimmäisen elementin sijainnin suhteessa elementtiin jossa valinta sijaitsee.
`.val()`	Hakee tai asettaa kaavake elementin arvon.

Taulukko 2. Yleisimpiä “getter” ja “setter”-metodeja (jQuery Learning Center n.d.f)

5.4.2 Ketjuttaminen

Yksi jQueryn vahvimmista ominaisuuksista on metodien ketjuttaminen. Se on ominaisuus, jonka monet muista kirjastoista ovat ottaneet käyttöön sen jälkeen kun jQuery teki siitä suositun. Sen helppokäyttöisyydestä johtuen sitä on myös helppo käyttää väärin: pitkät metodiketjut tekevät koodista vaikeasti luettavaa, muokattavaa ja debuggattavaa. Ei ole olemassa tiettyä nyrkkisääntöä kuinka pitkiä ketjujen tulee olla, vaan tärkeintä on että koodi pysyy luettavana. (jQuery Learning Center n.d.e)

Jos metodi palauttaa jQuery-objektin, voi toisen metodin kutsua heti edellisen perään. Esimerkiksi seuraavassa käytetään ensin ”.find()”-metodia etsimään kaikki ”sisalto”-elementin ”<h3>”-tason otsikot, valitaan niistä kolmas ja asetetaan sille sisällöksi ”Uusi otsikkoteksti”. (jQuery Learning Center n.d.e)

```
$("#sisalto").find("h3").eq(2).html("Uusi otsikkoteksti!");
```

Luettavuuden kannalta voi olla myös hyvä rivittää metodit omille riveilleen. Huomaa ettei rivien loppuun tule puolipilkkua.

```
$("#sisalto").find("h3")
             .eq(2)
             .html("Uusi otsikkoteksti!");
```

jQueryn ”.end()”-metodi on kätevä, kun halutaan ajaa saman valinnan eri alivalinoille eri metodeja.

```
$("#sisalto").find("h3")
             .eq(2)
             .html("Kolmas otsikkoteksti!" )
             .end() // palautetaan valinta takaisin ”h3”-
             .eq(0)  // elementeiksi
             .html("Ensimmäinen otsikkoteksti!");
```

(jQuery Learning Center n.d.e)

5.5 Elementtien manipulointi

Luvussa 5.4.1 esiteltiin kuinka elementtien sisältöä muokattiin eli manipuloitiin käyttämällä ”setter”-metodeja. Sisällön muokkaaminen jQueryllä on triviaalia, mutta on muistettava että muutokset vaikuttavat kaikkiin valittuihin elementteihin. Tämän takia on oltava tarkkana että tehdyt valinnat kohdistuvat vain haluttuihin elementteihin. (jQuery Learning Center n.d.f)

5.5.1 Elementtien siirtäminen ja kopioiminen

Muokkaamisen lisäksi jQueryllä voidaan myös siirtää, kopioida, poistaa sekä luoda elementtejä. On olemassa useita tapoja, joilla elementtejä voi siirtää ympäri DOM:ia, mutta yleisempiä niistä on seuraavat kaksi:

- Asettaa valittu elementti (tai elementtejä) suhteessa toiseen elementtiin
- Asettaa elementti suhteessa valittuihin elementteihin.

jQuery esimerkiksi tarjoaa kaksi metodia ”.insertAfter()” ja ”.after()”, jotka nimiensä perusteella kuulostavat suurin piirtein samalta. Näiden erona on, että ”.insertAfter()” asettaa valitut elementit metodille annetun elementtiin ja ”.after()” asettaa argumentiksi annetun elementin valinnan jälkeen. Monet muut siirtämismetodit noudattavat tätä menettelyä kuten ”.insertBefore()” ja ”.before()”, ”.prepend()” ja ”.prependTo()” sekä ”.append()” ja ”.appendTo()”. (jQuery Learning Center n.d.f)

Se, kumpaa näistä lähestymistavoista tulisi käyttää, riippuu täysin siitä mitä elementtejä on valittu ja halutaanko valinta tallettaa muuttujaan. Jos viittaus halutaan tallentaa, tulisi aina käyttää ensimmäistä lähestymistapaa: lisätä valitut elementit metodille annettuun elementtiin, koska se palauttaa arvona elementit jotka siirrettiin. Tällöin ”.insertAfter()”, ”.insertBefore()”, ”.appendTo()” ja ”.prependTo()” ovat metodeja joita tulisi käyttää. (jQuery Learning Center n.d.f)

Seuraavassa esimerkissä listan ensimmäinen elementti siirretään viimeiseksi käyttäen molempia tapoja.

```
var li = $("#lista li:first").appendTo("#lista") // tapa 1
$("#lista").append($("#lista li:first")); // tapa 2
```

Molemmat saavuttivat saman lopputuloksen, mutta alempaa tapaa ei voitu käyttää viittaamaan siirrettyyn elementtiin: ”.append()”-metodi palautti siirretyn elementin sijasta itse listan, johon elementti siirrettiin. (jQuery Learning Center n.d.f)

Kopioiminen tapahtuu käyttämällä ”.clone()”-metodia. Se ottaa yhden argumentin, joka on oletusarvoisesti epätosi, ”false”. Tämän argumentin asettaminen todeksi (”true”) kopioi elementin lisäksi myös siihen asetetut tapahtumat ja siihen liitetyt data-arvot. Seuraavassa esimerkissä kopioidaan listan ensimmäinen elementti listan viimeiseksi säilyttäen siihen liitetyt tapahtumat ja datat. (jQuery Learning Center n.d.f)

```
$("#lista li:first").clone(true).appendTo("#lista")
```

5.5.2 Elementtien poistaminen ja luominen

Elementtejä voidaan poistaa “.remove()” ja “.detach()”-metodien avulla. Metodia ”.remove()” tulee käyttää, kun elementti halutaan poistaa pysyvästi sivulta. ”.remove()”-metodi palauttaa poistetut elementit jos ne halutaan esimerkiksi tallettaa muuttujaan, mutta ei säilytä niihin liitettyjä tapahtumia tai data-arvoja. (jQuery Learning Center n.d.f)

Jos data-arvot ja tapahtumat halutaan säilyttää, tulee käyttää ”.detach()”-metodia. ”.detach()”-metodi on arvokas kun elementille halutaan tehdä raskaita tai monimutkaisia manipulointeja. Tällöin voidaan elementti poistaa väliaikaisesti sivulta, tehdä sille halutut muutokset ja palauttaa se takaisin sivulle. Tällöin itse DOM:ille tehdään mahdollisimman vähän muutoksia, jotka selaimen näkökulmasta tarvitsevat paljon suorituskykyä. (jQuery Learning Center n.d.f)

Jos poistamisen sijaan halutaan vain tyhjentää olemassa oleva elementti, voidaan käyttää metodia ”.empty()”, joka poistaa valitun elementin sisällä olevan HTML-koodin. (jQuery Learning Center n.d.f)

Uusien elementtien luominen on jQueryn avulla triviaalia käyttäen valintojen tekemiseen ”$()”-metodia. Uusi elementti voidaan luoda kirjoittamalla se suoraan HTML-merkintänä valitsimen paikalle tai luomalla se attribuutti-objektin kanssa. (jQuery Learning Center n.d.f)

```
// uusi elementti HTML-merkintänä luotuna
$("<a class="linkki" href="index.html">Etusivulle</a>");

// sama elementti käyttäen attribuuttiobjektia
$("<a/>", {
  html: "Etusivulle",
```

```
  "class": "linkki",
  href: "index.html"
});
```

Huomaa että “class” on lainausmerkeissä mutta ”html” ja ”href” eivät. Tämä johtuu siitä, että muutamat sanat kuten “class” ovat niin kutsuttuja varattuja sanoja, joilla on erityinen merkitys JavaScriptissä. Näitä varattuja sanoja tulee käyttää vain kun niitä käytetään niiden varsinaisessa merkityksessä, kuten ylläolevassa esimerkissä linkin ominaisuutena ”class”. Muuten ominaisuudet eivät vaadi lainausmerkkejä. (jQuery Learning Center n.d.f; jQuery Learning Center n.d.g)

Kun uusi elementti luodaan, sitä ei automaattisesti lisätä sivulle vaan se pitää tehdä itse käyttäen esimerkiksi jotain kappaleessa 5.5.1 mainittua metodia. Elementti voidaan myös lisätä suoraan sivustoon ilman sen tallentamista muuttujaan, mutta lisätylle elementille halutaan usein tehdä muutoksia lisäämisen jälkeen. Tällöin elementin tallentaminen muuttujaan on järkevää. Uusi elementti voidaan luoda myös suoraan esimerkiksi ”append()”-metodin argumenttina, mutta tällä tavalla luotua elementtiä ei voi myöskään suoraan referoida. (jQuery Learning Center n.d.f)

Koska elementtien luominen on yksinkertaista, on houkuttelevaa unohtaa että elementtien lisääminen DOM:iin toistuvasti on erittäin selaimen suorituskykyä verottavaa. Jos on tarpeen lisätä useita elementtejä samaan elementtiin, on suositeltavaa että HTML-elementit luodaan erikseen, lisätään yhteen arrayhin, yhdistetään yhdeksi merkkijonoksi ja vasta sen jälkeen lisätään elementtiin. (jQuery Learning Center n.d.f)

Esimerkissä lisätään 100 elementtiä listaan huonolla sekä suositellulla tavalla.

1. Huono tapa, 100 muutosta DOM:iin minkä lisäksi myös lista haetaan 100 kertaa.

```
var i = 0;
for (i; i < 100; i += 1) {
  $("#lista").append("<li>elementti " + i + "</li>");
}
```

2. Suositeltu tapa: Lista talletetaan muuttujaan ja uudet elementit tallennetaan arrayhin josta tehdään yksi merkkijono käyttäen ”.join()”-metodia. Yksi muutos DOM:iin, yksi listan nouto.

```
var elementit = [],
    lista = $("#lista"),
    i = 0;

for (i; i < 100; i += 1) {
  elementit.push("<li>elementti " + i + " </li>");
}
lista.append(elementit.join(""));
```

5.5.3 Attribuuttien manipulointi

Myös attribuuttien manipulointi jQueryllä onnistuu kattavasti. Tavalliset muutokset ovat helppoja mutta ”.attr()”-metodi mahdollistaa myös monimutkikkaat muutokset. Muutos voi olla täsmällinen arvo, tai arvo joka on funktion palautusarvo. Käytettäessä funktiosyntaksia, funktio saa kaksi argumenttia: nollasta lähtevän elementin ”index”-arvon sekä attribuutin arvon ennen muutosta. (jQuery Learning Center n.d.f)

```
//yhden attribuutin muuttaminen
$(”#lista a").attr("href", "toinen_sivu.html");

//useamman attribuutin muuttaminen
$("#lista a").attr({
  href: "toinen_sivu.html",
  "class": "uusi_luokka"
});

//attribuutin muuttaminen käyttäen funktio-syntaksia
$("#lista a").attr({
  href: function (idx, href) {
        return idx + "_" + href; //palauttaa esimerkiksi
"0_index.html"
  },
  "class": "uusi_luokka"
});
```

5.6 jQuery-objekti

Kun luodaan uusia elementtejä tai valitaan jo olemassa olevia, jQuery palauttaa elementit kokoelmana jota kutsutaan jQuery-objektiksi. Monet kehittäjät, joille jQuery ei ole tuttu, olettavat että tämä kokoelma on array-tyyppinen jono, koska se on nolla-indeksistä lähtevä sekvenssi DOM-elementtejä, sillä on joitain tuttuja array-funktioita, jonon pituuden ilmaiseva ”.length”-muuttuja ja niin edelleen. Todellisuudessa jQuery-objekti on paljon monimutkaisempi. (jQuery Learning Center n.d.h)

Kuten aiemmin mainittiin, jQueryn jQuery-objekti tarjoaa monia metodeja helpottamaan DOM:in muokkaamista. Kuten luvussa kolme todettiin, jQuery-objektin käyttäminen helpottaa yhteensopivuusongelmia ja sitä on usein myöskin käyttäjän kannalta mukavampi kirjoittaa sekä helpompi lukea verrattuna puhtaaseen JavaScript-koodiin. (jQuery Learning Center n.d.h)

Kun jQuery-funktiota $() kutsutaan jollain CSS-valitsimella, se palauttaa jQuery-objektin joka sisältää kaikki tähän valitsimeen liittyvät elementit. Esimerkiksi seuraava koodirivi valitsee kaikki ”h1”-tason otsikot jotka ovat jo sivulla. Jos sivulla on ”h1”-tason otsikoita, tulisi jQuery-elementin pituus eli ”.length”-muuttujan arvo olla enemmän kuin nolla. Tämän tarkistaminen on yleinen tapa tarkistaa, palauttiko valinta ainuttakaan elementtiä. (jQuery Learning Center n.d.h)

```
var otsikot = $("h1");
alert(otsikot.length);
```

Jos tavoitteena on hakea vain ensimmäinen valittu elementti, tarvitaan toinen vaihe. Tavoitteen saavuttamiseksi on monta tapaa, mutta helpoin niistä on käyttää metodia ”.eq()”.

```
var otsikot = $("h1"),
    ensimmainen_otsikko = otsikot.eq(0);
```

Nyt muuttuja ”ensimmainen_otsikko” on jQuery-objekti joka sisältää vain ensimmäisen ”h1”-tason otsikon sivulta. Ja koska ”ensimmainen_otsikko” on jQuery-objekti, sisältää se myös kaikki jQueryn hyödylliset metodit kuten ”.html()” sekä ”.after()”. Objektilla on myös metodi ”.get()” joka palauttaa jQuery-objektin sijasta itse DOM -elementin. (jQuery Learning Center n.d.h)

```
var ensimmainen_otsikko = $("h1").get(0);
```

Tällöin ”ensimmainen_otsikko” on natiivi DOM-elementti, joka tarkoittaa että sillä on DOM-ominaisuuksia kuten ”.innerHTML” muttei jQuery-metodeita kuten ”.html()” tai ”.after()”. Tällöin ”ensimmainen_otsikko”-elementin kanssa työskentely on vaikeampaa, mutta on joitain tilanteita jolloin tämän kaltainen toiminta on pakollista joista yksi on elementtien vertailu. (jQuery Learning Center n.d.h)

Kun elementtejä sisällytetään jQuery-objektiin, on jokainen objekti uniikki, vaikka valitsin olisi ollut täysin sama ja ne olisi luotu täysin samalla tavalla. Ne kuitenkin sisältävät samat DOM-elementit, joiden vertailu on mahdollista käyttäen ”.get()”-metodia. (jQuery Learning Center n.d.h)

```
var logo1 = $("#logo"),
    logo1elementti = logo1.get(0);

var logo2 = $("#logo"),
    logo2elementti = logo2.get(0);

alert(logo1 === logo2); // "false", epätosi!
alert(logo1elementti === logo2elementti); // "true", tosi!
```

On tärkeää erottaa jQuery-objektit ja natiivit DOM-elementit toisistaan. Natiivien DOM-elementtien metodit ja ominaisuudet eivät kuulu jQuery-objektiin ja päinvastoin. Virheilmoitukset kuten ”event.target.closest is not a function” ja ”TypeError: Object [object Object] has no method 'setAttribute'" viittaavat että edellä mainittu yleinen virhe on tapahtunut. (jQuery Learning Center n.d.h)

Toinen yleinen väärä johtopäätelmä on olettaa että jQuery-objektit olisivat ”eläviä”, eli että niiden sisällöt päivittyisivät automaattisesti kun sivua päivitetään. jQuery-objektit eivät kuitenkaan toimi näin, vaan muuttujat tulee päivittää käsin jos dokumentti muuttuu. Tämä tapahtuu yksinkertaisesti kuin päivittämällä muuttuja ajamalla sama valitsin uudestaan. (jQuery Learning Center n.d.h)

```
var otsikot = $("h1");
// myöhemmin koodissa kun otsikoita on lisätty
otsikot = $("h1");
```

5.7 Kokoelmien läpikäyminen

Kun alustava valinta on tehty jQueryllä, voi valittua kokoelmaa läpikäydä syvemmälle kuin alkuperäinen valinta. Läpikäyminen voidaan jakaa kolmeen osa-alueeseen: vanhempiin (parents), lapsiin (children) ja sisaruksiin (siblings). jQuery tarjoaa runsaan valikoiman metodeja joiden avulla näitä osa-alueita voidaan läpikäydä. Joillekin näistä metodeista voidaan antaa argumentiksi valitsin ja osa voi myös käyttää toista jQuery-objektia argumenttina valinnan filtteröimiseksi. (jQuery Learning Center n.d.i)

Vanhempien, eli elementtien joihin alkuperäinen elementti sisältyy, etsimiseen on olemassa metodeja kuten ”.parent()”, ”.parents()”, ”.parentsUntil()” sekä ”.closest()”. Näistä ensimmäinen palauttaa elementin ensimmäisen vanhemman elementin, toinen kaikki vanhemmat ja kolmas kaikki vanhemmat elementit kunnes jokin elementti täyttää valinnan kriteerit. Metodi ”.closest()” vastaa ”.parentsUntil()”-metodia, mutta sisältää myös itsensä hakuun. (jQuery Learning Center n.d.i)

Lapsielementtejä, eli valintaan sisältyviä elementtejä, voidaan läpikäydä ”.children()” sekä ”.find()” -metodeilla. Näiden erona on, että ”.children()” etsii tuloksia vain yhtä tasoa alempana olevista elementeistä kun taas ”.find()” voi hakea elementtejä rekursiivisesti kaikista lapsi-elementeistä, niiden lapsista ja niin edelleen. (jQuery Learning Center n.d.i)

Sisarelementtien, eli elementtien jotka sijaitsevat samalla tasolla kuin valinta, voidaan myös läpikäydä monella tavalla. Metodit ”.next()” ja ”.prev()” palauttavat seuraavan tai edellisen sisarelementin kun taas ”.siblings()” palauttaa nämä molemmat (ja kaikki muut saman tason elementit). On myös muutamia funktioita jotka rakentuvat näiden pohjalle, kuten ”.nextAll()”, ”.nextUntil()”, ”.prevAll()” sekä ”.prevUntil()”. (jQuery Learning Center n.d.i)

Vaikka jQueryllä voi käytännössä navigoida mihin tahansa kohtaan dokumenttia, on suositeltavaa välttää pitkiä ja monimutkikkaita läpikäymisiä. Monimutkaiset haut vaativat että dokumentin rakenne pysyy aina samana mikä on monesti vaikeaa varmistaa. Muutaman askeleen läpikäyminen on hyväksyttävää, mutta yhdestä komponentista toiseen hyppäämistä tulisi välttää. (jQuery Learning Center n.d.i.)

5.8 CSS tyylit, elementtien sijainnit sekä dimensiot

jQuery mahdollistaa CSS-tyylien hakemisen ja asettamisen käyttäen ”.css()” -metodia. CSS -arvot, jotka sisältävät väliviivan, pitää yleensä muuttaa ”camelCase” -muotoon jossa sanat ovat kirjoitettu yhteen siten että ensimmäisen sanan ensimmäinen kirjain on pieni ja muiden kapitalisoitu.

Tämä ei kuitenkaan päde jos CSS-arvon nimi on kirjoitettu sulkuihin. Arvoja voidaan myös asettaa useita samanaikaisesti, jos nimi-arvo-parit kirjoitetaan objekteiksi. (jQuery Learning Center n.d.j)

```
// CSS-arvojen hakeminen
$(”h1”).css("fontSize"); // palauttaa fonttikoon kuten
"19px";
$(”h1”).css("font-size"); //tämäkin toimii.

// CSS arvojen asettaminen
$("h1").css("fontSize", "100px"); // fonttikoon asettaminen
$("h1").css({"fontSize: "100px", color: "red"}); //usean
arvon muuttaminen
```

Vaikka arvojen hakijana ”.css()”-metodi on erittäin käytännöllinen, sen käyttämistä arvojen asettajana valmiissa koodissa tulisi yleensä välttää. Arvojen käsin asettamisen sijasta olisi hyvä käyttää CSS-luokkaa, joka määrittää kaikki muutettavat arvot. Tämän jälkeen luokka voidaan lisätä tai poistaa elementille tarpeen mukaan. Luokkia voidaan myös käyttää esittämään elementin tilaa, kuten ilmoittamaan onko kyseinen elementti valittu. (jQuery Learning Center n.d.j)

```
var h1 = $("h1");

h1.addClass("iso"); // lisää luokan "iso"
h1.removeClass("iso"); //poistaa luokan "iso"
h1.toggleClass("iso"); //lisää tai poistaa luokan "iso";

if (h1.hasClass("iso") {
  //tämä ajetaan jos elementillä on luokka "iso"
}
```

Ulkonäön lisäksi voi jQueryllä tarkastella ja asettaa elementtien sijainteja sekä dimensioita. Näitä metodeja ovat esimerkiksi ”.width()”, ”.height()”, ”.position()” ja ”.offset()”. jQuery sisältää myös monia variaatioita näistä, kuten pelkästään sisä- tai ulkoleveyden (ilman marginaaleja tai niiden kanssa) valitsemisen. Metodien ”.position()” ja ”.offset()” ero on pieni mutta merkittävä. Metodi ”.position()” palauttaa elementin etäisyyden suhteessa elementtiin jossa se sijaitsee, kun taas ”.offset()” palauttaa elementin etäisyyden dokumentin yläkulmasta. (jQuery Learning Center n.d.j)

5.9 Datan hallinta

Usein on olemassa elementtiin liittyvää dataa jota kehittäjä haluaa sisällyttää elementteihin. Tavallisella JavaScriptillä tämä voidaan tehdä lisäämällä data DOM-elementin ominaisuudeksi, mutta tämä aiheuttaa helposti muistivuotoja joidenkin selainten kanssa. jQuery mahdollistaa yksinkertaisen tavan tallettaa dataa elementteihin samalla hoitaen mahdolliset muistivuodot. Elementteihin voidaan tallettaa minkä tahansa tyyppistä dataa. (jQuery Learning Center n.d.k)

```
$("#elementti").data("datanNimi", {foo: "bar"});
$("#elementti").data("datanNimi") // palauttaa {foo: "bar"}
```

5.10 Ajax

Internetin alkuvaiheilla internetsivut tuli päivittää lataamalla sivut uudelleen. Esimerkiksi sähköpostin käyttäjälle tämä tarkoitti sivun lataamista uudelleen, jotta käyttäjä näki, oliko uutta sähköpostia tullut. Tämä oli hidasta ja se vaati käyttäjän tekevän manuaalista työtä. Samalla palvelimen täytyi rakentaa koko sivusto uudestaan ja lähettää sivuston HTML-rakenne, CSS-tiedostot, JavaScript-tiedostot sekä käyttäjän sähköpostit takaisin käyttäjälle. Ideaalisesti käyttäjälle olisi täytynyt lähettää vain uudet sähköpostit, ei koko sivua. Vuoteen 2003 mennessä kaikki suosituimmat selaimet ratkaisivat tämän ongelman implementoimalla XMLHttpRequest (XHR) -objektin, joka mahdollisti selaimen kommunikoinnin palvelimen kanssa ilman sivun uudelleenlatausta. (jQuery Learning Center n.d.l)

XMLHttpRequest-objekti on osa teknologiaa nimeltä Ajax (Asynchoronous JavaScript and XML). Ajaxin avulla dataa voidaan lähettää palvelimen ja selaimen välillä, käyttäen XMLHttpRequest API:a. Ajax-pyynnöt saadaan aikaan käyttämällä JavaScriptia: koodi lähettää pyynnön URL:ille eli verkko-osoitteeseen, joka saadessaan vastauksen voi ajaa niin kutsutun callback-funktion. Koska nämä pyynnöt ovat asynkronisia, muu sivuston koodi jatkaa toimintaansa sillä aikaa kun pyyntöjä prosessoidaan, jolloin callback-funktioiden käyttäminen on välttämätöntä. (jQuery Learning Center n.d.l)

Harmillisesti eri selaimet implementoivat Ajax API:n eri tavoilla. Tavallisesti tämä tarkoittaa, että ohjelmoijien tulee ottaa huomioon kaikki eri selaimet, jotta voidaan varmistaa että Ajax toimisi universaalisti. jQuery tarjoaa Ajax -tuen, jonka avulla selainten eriäväisyydet voidaan jättää jQueryn huoleksi. jQuery tarjoaa kattavan ”$.ajax()”-funktion, sekä useita helppokäyttöisyysmetodeita, kuten ”$.get()”, ”$.getScript()”, ”$.getJson()”, ”$.post()” sekä ”$.load()”. Monet jQuery-applikaatiot eivät käytännössä käytä XML:ää Ajax-termistä huolimatta, vaan dataa siirretään HTML- tai JSON (JavaScript Object Notation) -muodoissa. (jQuery Learning Center n.d.l)

JSON-objekti muistuttaa JavaScript-objektia. Se koostuu kahdesta rakenteesta: järjestelemättömistä nimi-avain-pareista eli objekteista sekä järjestellyistä arvojoukoista (array). Objektit erotellaan tosistaan aaltosulkeilla ({ ja }) ja nimi-arvo-parit pilkulla. Arvojoukot kirjoitetaan hakasulkujen ([ja]) sisään ja niiden arvot erotetaan toista pilkuilla. Nimet ovat merkkijonoja (string) ja arvot voivat olla joko merkkijonoja, numeroita, objekteja, arrayjonoja, boolean-arvoja tai null-arvoja. (JSON n.d.)

5.10.1 Ajaxin avainkonseptit

jQueryn tarjoamien Ajax-metodeiden hyväksikäyttämiseksi tulee ymmärtää joitain avainkonsepteja, kuten GET ja POST-metodien eroavaisuuden, datatyyppien eroavaisuudet sekä asynkronisuuden tarkoituksen. (jQuery Learning Center n.d.m)

GET ja POST-metodit ovat kaksi yleisintä tapaa lähettää palvelimelle pyyntöjä. GET-metodia käytetään turvallisiin operaatioihin, eli pyyntöihin joissa

vain haetaan dataa palvelimelta, palvelimen dataa muuttamatta. Esimerkiksi hakumoottorille voidaan lähettää kysely käyttäen GET-metodia. Selaimet saattavat asettaa GET-pyynnön välimuistiin, joka saattaa johtaa epäjohdonmukaiseen käyttäytymiseen. GET-pyynnöt lähettävät yleensä pyyntönsä kyselymerkkijonona. POST-metodia käytetään vastaavasti operaatioihin, joissa palvelimen dataa muutetaan. Esimerkiksi blogi-kirjoituksen tallentaminen tulisi lähettää POST-pyyntönä. POST-pyyntöjä ei yleensä myöskään aseteta välimuistiin. POST-pyynnöt voivat olla myös kyselymerkkijonoja osana URL:lia, mutta data lähetetään yleensä erikseen POST-datana. (jQuery Learning Center n.d.m)

jQuery yleensä vaatii jonkinlaisen viittauksen siihen, minkälaista dataa tulisi Ajax-pyynnön palauttaa. Jossain tapauksissa datatyyppi määrittyy käytetyn metodin mukaan, toisissa tapauksissa se tulee määrittää osana konfiguraatio-objektia. Vaihtoehtoja ovat:

- text – yksinkertaisten merkkijonojen lähettämiseen
- html – HTML-merkintöjen lähettämiseen
- script – uuden JavaScriptin lisäämiseksi sivustolle
- json – JSON-formatoidun datan lähettämiseen
- jsonp – JSON-datan lähettämiseen eri sivustojen välillä
- xml – XML-datan lähettämiseen

(jQuery Learning Center n.d.m)

Ajaxin asynkronisuus voi yllättää monet uudet jQueryn käyttäjät. Koska Ajax-pyynnöt ovat asynkronisia, vastaukset eivät ole heti käytettävissä. Vastauksia tulee siis käsitellä käyttäen ”callback”-funktioita. Esimerkiksi seuraava koodi ei toimi:

```
var vastaus;
$.get("foo.php", function (r) {
  vastaus = r;
});
console.log(vastaus) // undefined, ei määritetty
```

Sen sijaan dataa tulee käsitellä ”callback”-funktion sisällä

```
$.get("foo.php", function (vastaus) {
  console.log(vastaus) // palvelimen vastaus
});
```

(jQuery Learning Center n.d.m)

Yleisesti Ajax-pyynnöt ovat rajoitettu saman protokollan (http:// tai salattu https://), portin sekä verkkonimen välille jolla pyynnön lähettänyt sivu on. Tätä kutsutaan same-origin policy:ksi, saman lähteen periaatteeksi. Tämä ei vaikuta skripteihin jotka ladataan käyttäen jQueryn Ajax-metodeja. Toinen poikkeus tähän ovat pyynnöt jotka lähetetään JSONP -palveluun joka pyörii eri verkkonimen alla. JSONP:n tapauksessa palveluntarjoaja on hyväksynyt vastaamaan muista verkkonimistä tulleisiin pyyntöihin käyttäen skriptiä joka ladataan sivulle käyttäen <script>-tagia, jolloin vältetään saman lähteen periaatteen rajoitteet. Ladattu skripti sisältää lähetetyt datat sekä käyttäjän antaman ”callback”-funktion. (jQuery Learning Center n.d.m)

5.10.2 jQueryn Ajax-metodit

Vaikka jQuery tarjoaa monia helppokäyttöisyysmetodeja Ajax:in hyödyntämiseen, ydinmetodi ”$.ajax()” toimii niiden kaikkien pohjana. Se tarjoaa helpon ja kattavan tavan Ajax-pyyntöjen luomiselle. Se ottaa vastaan konfiguraatio-objektin, joka sisältää kaikki määritteet jotka jQuery vaatii pyynnön suorittamiseksi. ”$.ajax()” mahdollistaa myös ”callback”-funktioiden luomisen sekä pyynnön onnistumis- että epäonnistumistilanteisiin. Seuraavassa esimerkissä luodaan yksinkertainen pyyntö käyttäen ”$.ajax()”-metodia. Kattavat konfiguraatio-optiot löytyvät osoitteesta http://api.jquery.com/jQuery.ajax/. (jQuery Learning Center n.d.n)

```
// $.ajax()-metodin käyttäminen
$.ajax({
  // URL johon pyyntö lähetetään
  url: "post.php",

  // data joka lähetetään (muutettuna kyselymerkkijonoksi)
  data: {
    id: 123
  },

  //pyynnön tyyppi
  type: "GET",

  // palautettavan datan oletettu tyyppi
  dataType : "json",

  // koodi joka ajetaan jos pyyntö onnistuu
  success: function( json ) {
    alert(json);
  },

  // koodi joka ajetaan jos kysely epäonnistui, funktion parametreiksi on annettu alkuperäinen kysely, statuskoodi sekä lähetetty virheilmoitus
  error: function(xhr, status, virhe) {
    alert( "Kysely epäonnistui" );
    console.log("Virhe: " + virhe);
    console.log("Status: " + status);
    console.dir( xhr);
  },

  // koodi joka ajetaan joka tapauksessa
  complete: function(xhr, status) {
    alert("Kysely toteuttin");
  }
});
```

Datatyypin määrittämisessä on huomioitava, että väärän datatyypin käyttäminen saattaa johtaa koodin epäonnistumiseen eikä syy ole aina selvää, koska HTTP -vastauskoodi ei näytä virhettä. Kun Ajax-pyyntöjä tehdään, tulee varmistaa että selain vastaa datatyypillä joka siltä on pyydetty. Esimerkiksi JSON-datan sisältötyypin tulisi olla ”application/json”. (jQuery Learning Center n.d.n)

Jos ”$.ajax()”-metodin kattavaa konfiguroitavuutta ei tarvita, voidaan käyttää jQueryn tarjoamia helppokäyttöisyysmetodeja. Nämä metodit ovat ”wrappereita” ”$.ajax()”-metodin ympärillä, jotka yksinkertaisesti asettavat tiettyjä asetuksia ”$.ajax()”-metodiin. Näitä helppokäyttöisyysmetodeja ovat:

- $.get – GET pyyntöjen tekeminen
- $.post – POST pyyntöjen tekeminen
- $.getScript – Skriptin lisääminen sivustolle
- $.getJSON – GET pyynnön tekeminen joka olettaa vastauksen olevan JSON-muotoista dataa.
  (jQuery Learning Center n.d.n)

jQuery sisältää myös erityisen ”.load()”-metodin, joka kutsutaan valitsimella valittuun elementtiin. Tämä metodi lataa valittuun elementtiin vastauksena tulleen HTML-sisällön tai osia siitä, jos pyyntöön on lisätty valitsimia. Seuraavassa esimerkissä on esitetty molempien tapojen käyttötavat.

```
// ".load()"-metodin käyttäminen sisällön lataamiseen
$("#uusiSisalto").load("/foo.html");

// Load-metodin käyttäminen käyttäen valitsimia
$("#uusiSisalto").load("/foo.html #omaDiv h1:first", func-
tion(html) {
  alert("Sisaltö päivitetty!");
});
```

(jQuery Learning Center n.d.n)

5.11 Deferred-objektit

jQueryn deferred-objekteja voidaan pitää tapana esittää asynkronisia operaatioita, jotka saattavat kestää pitkään. Ne ovat asynkronisia vaihtoehtoja niin kutsutuille estäville funktioille, jotka estävät tiettyjä tapahtumia sillä aikaa kun jokin asynkroninen operaatio on käynnissä. Estämisen sijaan funktiot palauttavat deferred-objektin välittömästi, mihin taas liitetään ”callback”-funktioita. Nämä ”callback”-funktiot ajetaan, kun operaatio on valmis. (jQuery Learning Center n.d.o)

Deferred-objektit perustuvat niin kutsuttuun Promise-malliin, jonka konsepti on sama kuin ylläkuvattu deferred-objektien toiminta: palautetaan ”lupaus” tulevasta arvosta. Kuvitellaan tilanne jossa web-applikaatio pohjautuu kolmannen osapuolen API:iin. Yleinen ongelma on, että kolmannen osapuolen palvelimien toimintaa ei voida aina varmistaa. Esimerkiksi palvelussa saattaa olla ruuhkaa, joka voi johtaa webapplikaation jumiutumiseen, kunnes kolmannen osapuolen palvelin jälleen vastaa. Deferred-objektit tarjoavat ratkaisun tähän ongelmaan. (jQuery Learning Center n.d.o)

CommonJS:n, JavaScript ekosysteemin luomiseen tähtäävän ryhmän, ehdotus Promise/A määrittää metodin ”then”. ”Then”-metodin avulla voidaan

rekisteröidä ”callback”-funktioita lupauksiin joiden avulla voidaan käsitellä myöhemmin saatavia arvoja. Esimerkiksi koodi voi näyttää tältä:

```
var lupaus = APIkutsu(argumentti1, argumentti);

lupaus.then(function (saatuArvo) {
  /* käsitellään saatu arvo */
});

lupaus.then(function (saatuArvo) {
   /* tehdään jotain muuta samalla arvolla */
});
```

(jQuery Learning Center n.d.o)

Lupaus voi päättyä kahteen eri tilaan: ”resolved”-tilaan, jossa data oli saatavilla, sekä ”rejected”-tilaan, jossa palvelu tai arvo ei ollut saatavilla. Tämän vuoksi ”then”-metodi ottaa vastaan kaksi argumenttia: toinen tilanteelle jossa pyyntö onnistui, toinen tilanteelle jossa pyyntö epäonnistui. Tällöin funktiot kirjoitetaan toistensa perään.

```
lupaus.then(function (saatuArvo) {
  /* käsitellään arvo */
}, function () {
  /* mitä tehdään, jos kysely epäonnistui */
});
```

(jQuery Learning Center n.d.o)

Jossain tapauksissa vaatimuksena on, että palvelimelta saadaan vastaus useaan kyselyyn, ennen kuin ohjelma voi jatkaa toimintaansa. Näissä tapauksissa voidaan käyttää metodia ”when”, jonka avulla voidaan ”then”-metodi ajaa vasta kun kaikkiin kyselyihin on saatu vastaus.

```
when(
  lupaus1,
  lupaus2,
  …).then(function (arvo1, arvo, …) {
    * kaikki lupaukset on palautettu */
  });
```

(jQuery Learning Center n.d.o)

Kuvitellaan esimerkiksi tilanne, jossa käyttäjälle näytetään useaa samanaikaista animaatiota. Ilman jokaisen animaation ”callback”-tilanteiden tarkkailua voi olla hankalaa havaita milloin kaikki animaatiot ovat päättyneet. Lupauksia ja ”when”-metodin käyttäminen on yksinkertainen tapa saavuttaa tämä: animaatiot voivat käytännössä sanoa ”me lupaamme ilmoittaa kun animaatio on valmis”. Tällöin voidaan helposti ajaa yksi ”callback”-funktio kun kaikki animaatiot ovat valmiita.

```
var lupaus1 = $( "#id1" ).animate().promise();
var lupaus2 = $( "#id2" ).animate().promise();
when(
  lupaus1,
  lupaus2
).then(function() {
  /* tämä ajetaan kun molemmat animaatiot ovat valmiita*/
});
```

Tämä tarkoittaa, että kehittäjä voi helposti kirjoittaa toimintaa estämätöntä logiikkaa, joka voidaan suorittaa ilman synkronoimista. (jQuery Learning Center n.d.o)

Deferred-objektit lisättiin jQueryn Ajax-moduulin uudelleenkirjoittamisen yhteydessä, seuraten edellä mainittua CommonJS Promises/A-mallia. Näin ollen jQueryn versiot 1.5:stä ylöspäin sisältävät deferred-ominaisuudet, kun taas vanhemmat versiot käyttivät jQueryn Ajax-”callback”-funktioita toteuttamaan nämä toiminnollisuudet. Tämä johti funktioiden pariutumiseen, tilanteisiin joissa eri funktiot olivat riippuvaisia toisistaan. (jQuery Learning Center n.d.p)

jQueryn Promises/A-implementaation ydin on jQuery.Deferred – ketjutettava rakentaja joka pystyy luomaan uusia deferred-objekteja. Nämä objektit voivat tarkistaa lupauksien olemassaolon saadakseen selville onko tarkkailtavaa objektia olemassa. jQuery.Deferred pystyy kutsua ”callback”-jonoja ja määrittää asynkronisten että synkronisten tapahtumien tiloja. On huomattava, että deferredin alkutila on ”unresolved”, selvittämätön. Callback-funktiot, jotka voidaan liittää deferred:iin ”.then()” ja ”.fail()” funktioiden kautta, asetetaan jonoon ja suoritetaan myöhemmin prosessin aikana. (jQuery Learning Center n.d.p)

Deferred-objekteja voidaan käyttää yhdessä lupaus-konseptin when()-metodin kanssa, joka on implementoitu jQueryyn ”$.when()”-metodina, odottamaan kaikkien deferred-objektien pyyntöjen valmistuista. Käytännössä ”$.when()” on tapa suorittaa callback-funktoita jotka pohjautuvat rajaamattomaan määrään lupauksia jotka esittävät asynkronisia tapahtumia. (jQuery Learning Center n.d.p)

Esimerkki ”$.when()”-metodin käyttämisestä: odotetaan monen Ajax-pyynnön valmistumista, joidenka jälkeen suoritetaan joko onnistumiselle tai epäonnistumiselle asetettu funktio.

```
function onnistui() {
  console.log("onnistui!");
}

function epaonnistui() {
  console.log("epäonnistui")
}

$.when(
  $.ajax( "/index.html" ),
  $.ajax( "/sivu1.html" ),
  $.ajax( "/sivu2.html" )
).then(onnistui, epaonnistui);
```

Jos kaikki sivut ladattiin onnistuneesti, ilmoittaa kehittäjän konsoli ”onnistui!”. Jos yksi tai useampi pyyntö epäonnistui, ilmoittaa konsoli taas ”epäonnistui”. Huomioitavaa on, että ”onnistui” ja ”epaonnistui”-funktiot ajetaan maksimissaan kerran, vaikka deferred-objekteja olisi kuinka monta tahansa. (jQuery Learning Center n.d.p)

5.12 Tapahtumat

Tapahtumat (events) ovat tilanteita, joita käyttäjä käynnistää käyttäessään sivuja, esimerkiksi kirjoittaessaan tekstikenttään tai siirtäessään hiiren kursoria. Jossain tapauksissa, kuten sivuja avattaessa tai sulkiessa, selain itsessään laukaisee tapahtuman. (jQuery Learning Center n.d.q)

jQuery tarjoaa helppokäyttöisyysmetodeja suurimmalle osalla selainten natiiveja tapahtumia. Nämä metodit, kuten ”.click()”, ”.focus()”, ”.blur()”, ”.change()”, ovat oikoteitä jQueryn ”.on()”-metodille. Metodi ”.on()” on käytännöllinen, kun halutaan liittää sama tapahtumien käsittelijä (event handler) monelle tapahtumalle tai kun halutaan antaa jotain dataa tapahtuman käsittelijälle. Se on myös käytännöllinen kun käsitellään käyttäjän itse määrittämiä tapahtumia tai kun halutaan lähettää eteenpäin objekti joka sisältää useita tapahtumia ja käsittelijöitä. (jQuery Learning Center n.d.q)

Seuraava esimerkki esittää samaan tapahtumaan liitetyn käsittelijän käyttäen sekä helppokäyttöisyysmetodia että ”.on()”-metodia.

```
// Helppokäyttöisyysmetodi
$("p").click(function () {
  console.log("Klik!");
});

// Sama tapahtuma käyttäen ".on()"-metodia
$("p").on("click", function() {
  console.log("Klik!");
});
```

(jQuery Learning Center n.d.q)

On tärkeää huomata, että “.on”-metodi voi luoda tapahtumien kuuntelijoita (event listeners) vain elementteihin, jotka ovat olemassa niiden luomisen aikaan. Myöskään elementit, jotka on luotu vasta tapahtuman kuuntelijan luomisen jälkeen, eivät automaattisesti peri muihin vastaaviin elementteihin liitettyjä tapahtumia. (jQuery Learning Center n.d.q)

Ennen “.on()”-metodia, joka lisättiin jQueryyn versiossa 1.7., oli jQueryssä useita eri tapahtuma-metodeita kuten ”.bind()”, ”.live()”, sekä ”.delegate()”. Nämä metodit ovat vanhentuneita, eikä niitä tule käyttää kuin tilanteissa, joissa ainoa käytössä oleva versio on vanhempi kuin versio 1.7. (jQuery Learning Center n.d.t.)

5.12.1 ”Event handler”-funktion toiminta

Jokainen ”event handler”- eli tapahtumankäsittelijä-funktio vastaanottaa tapahtumaobjektin, joka sisältää monia ominaisuuksia ja metodeja. Sitä käytetään tavallisesti estämään elementille asetettu oletustapahtuma käyttäen ”.preventDefault()”-metodia. Se sisältää kuitenkin monia muitakin hyödyllisiä ominaisuuksia kuten esimerkiksi:

- pageX, pageY – hiiren sijainti suhteessa selaimen ikkunan yläkulmaan

- type – tapahtuman tyyppi, esimerkiksi ”click”
- data – data joka liitettiin tapahtumaan sitä liittäessä
- target – DOM elementti joka käynnisti tapahtuman
- namespace – tapahtumaan liitetty namespace
- timeStamp – ajan erotus, jona tapahtuma tapahtui, laskettuna millisekunteina 1.1.1970:stä
- preventDefault() – Oletustapahtuman estäminen, esimerkiksi linkin avaaminen
- stopPropagation() – Tapahtuman kuplimisen, eli esimerkiksi klikatun elementin alla olevan elementin klikkaamisen estäminen

(jQuery Learning Center n.d.q)

Tapahtumaobjektin lisäksi tapahtuman käsittelijä-funktiolla on myös pääsy DOM -elementtiin, johon käsittelijä oli liitetty ”this”-avainsanan kautta. Tämän elementin muuttamiseksi jQuery-objektiksi, voidaan ”this” yksinkertaisesti kirjoittaa muotoon ”$(this)”, yleensä samalla tallentaen se muuttujaan. (jQuery Learning Center n.d.q)

```
var elementti = $(this);
```

Kokonainen esimerkki yhdestä tapahtuman liittämisestä, sen käsittelijästä että tapahtuma-objektista näyttää seuraavalta:

```
$("a").on("click", function (evt) {
  //evt on tapahtuman tapahtumaObjekti
  //estään oletustapahtuma
  evt.preventDefault();
  // tallennetaan klikattu elementti muuttujaan
  var elem = $(this);
  // liitetään klikattuun elementtiin luokka "klikattu"
  elem.addClass("klikattu");
});
```

(jQuery Learning Center n.d.q)

5.12.2 Useiden tapahtumien käsittelijät, tapahtumien poistaminen, tapahtumien nimeäminen sekä tapahtumien kuuntelijoiden poistaminen

Monesti samoihin elementteihin liitetään monia eri tapahtumia. Jos moneen eri tapahtumatyyppiin voidaan liittää sama käsittelijäfunktio, voidaan tapahtumatyypit kirjoittaa peräkkäin välilyönnillä eroteltuina.

```
// sama kuuntelija monelle eri tapahtumalle
$(”input").on(
  "click change",
  function() {
    console.log( "Input elementtiä painettiin tai sen arvo
vaihtui" )
  });
```

(jQuery Learning Center n.d.q)

Jos kaikille tapahtumilla on oma käsittelijänsä, voidaan “.on()”-metodille antaa argumentiksi yhdestä tai useammasta koostuva avain-arvo-pari, jossa avain on tapahtuman nimi ja arvo on tapahtuman käsittelevä funktio.

```
$("p").on({
  "click": function() { console.log( "klikattu" ); },
  "mouseover": function() {
      console.log("hiiri elementtin päällä!");
  }
});
```
(jQuery Learning Center n.d.q)

Elementtiin liitetyt tapahtumat voidaan poistaa käyttämällä ”.off()”-metodia.

```
$(”p”).off(”click”);
```

Monimutkaisien applikaatioiden tai liitännäisten kanssa saattaa tapahtumat olla käytännöllistä nimetä, eli asettaa ne tiettyyn namespaceen, jotta vältytään poistamasta tapahtumia, joiden olemassa ei oltu tietoisia.

```
$("p").on("click.minunNamespace", function () { /* ... */ });
$("p").off("click.minunNamespace");
$("p").off(".minunNamespace"); // poistetaan kaikki tapahtumat, jotka liittyvät "minunNamespace"-namespaceen
```
(jQuery Learning Center n.d.q)

Tietyissä tilanteissa on suotavaa, että tietty tapahtuman käsittelijä ajetaan vain kerran. Tällöin voidaan käyttää ”.one()”-metodia. On kuitenkin huomattava, että ”.one()”-metodin kanssa käsittelijä ajetaan ensimmäisen kerran jokaista elementtiä kohden, sen sijaan että käsittelijä poistettaisiin kaikista valituista elementistä ensimmäisen kerran jälkeen. (jQuery Learning Center n.d.q)

5.12.3 Tapahtumien laukaisu käsin

jQuery mahdollistaa myös tapahtumien laukaisun ilman käyttäjänvuorovaikutusta käyttäen “.trigger()”-metodia. Sillä ei voida kuitenkaan laukaista mitä tahansa tapahtumia. jQueryn tapahtumien hallintasysteemi on rajapinta selainten natiivien tapahtumien päällä. Jos tapahtumienkäsittelijä lisätään käyttäen ”.on(”click”, function () {…})”-metodia, se voidaan laukaista käyttämällä ”.trigger()”-metodia, koska jQuery säilöö viittauksen alkuperäiseen käsittelijään. Tämän lisäksi ”.trigger()” ajaa myös kaiken JavaScript-koodin, joka on asetettu ”onclick”-attribuuttiin. (jQuery Learning Center n.d.r)

```
$("a").on("click", function (evt) {
  evt.preventDefault();
  console.log("Klik!");
});
//tapahtuman laukaiseminen
$("a").trigger("click");
```

“.trigger()”-metodia ei voida kuitenkaan käyttää imitoimaan selainten natiiveja tapahtumia, kuten linkin klikkaamista. Tämä johtuu siitä, ettei tapahtumien käsittelijää ole asetettu jQueryn kautta, jolloin jQueryllä ei ole linkin

painamistapahtumaa vastaavaa käsittelijää. Seuraavassa esimerkissä ”.trigger()”-metodin käyttäminen ei tee mitään, koska linkin avaaminen on selaimen tapahtuma jota ei ole liitetty jQueryn avulla.

```
<a href="http://learn.jquery.com">Learn jQuery</a>

$("a").trigger("click"); //mitään ei tapahdu
```
(jQuery Learning Center n.d.r)

Vaikka ”.trigger()”-metodilla on käyttötarkoituksensa, ei sitä tulisi käyttää kutsumaan funktiota joka on liitetty esimerkiksi ”click”-tapahtuman käsittelijään. Sen sijaan funktio tulisi tallentaa muuttujaan, joka taas tulisi asettaa tapahtuman käsittelijäksi. Tällöin tarvittaessa voidaan kutsua myös itse funktiota, ilman ”.trigger()”-metodin käyttämistä. (jQuery Learning Center n.d.r)

Sama koskee myös elementtejä, joita ei ole omassa ennen tapahtumankäsittelijän liittämistä. Jos sivulle lisätään uusia HTML-elementtejä, tulee niihin erikseen liittää tapahtumankäsittelijät vaikka elementtiä vastaavia tapahtumankäsittelijöitä sivulla olisi jo ennestään. Toinen tapa on käyttää tapahtumien delegointia, esimerkiksi tapahtuman käsittelijän liittämistä elementin sijasta dokumenttiin ja määrittämällä sille argumentiksi tarkkailtavan elementin.
(jQuery n.d.f.)

```
$(document).on("click", "a",  function () {
  console.log("Klik”);
});
```

5.12.4 Kustomoidut tapahtumat

Jos selaimen tarjoamat tapahtumat eivät riitä, voi käyttäjä luoda myös omia, kustomoituja tapahtumia. Kustomoidut tapahtumat ovat tapa siirtää tapahtumat pois elementistä, joka laukaisee tapahtuman, siihen elementtiin johon tapahtuma itse asiassa vaikuttaa. Tämä tarjoaa monia etuja kuten

- Kohde-elementin toimintoja voidaan helposti laukaista monien eri elementtien kautta käyttäen samaa koodia
- Toimintoja voidaan laukaista moniin samankaltaisiin elementteihin samanaikaisesti
- Toiminnot liittyvät selvemmin kohde-elementtiin, jolloin koodin ylläpitäminen on helpompaa.

(jQuery Learning Center n.d.s)

Kustomoituja tapahtumia voidaan luoda samalla tavalla kuin tavallisia tapahtumia: kiinnittämällä ne elementtiin käyttämällä ”.on()” -metodia ja laukaisemalla ne käyttäen ”.trigger()” -metodia. Seuraavassa esimerkissä luodaan ”lamppu”, jonka tilaa hallitaan kahdella kytkimellä. Lampulle luodaan kustomoitu tapahtuma ”vaihdaTilaa”, joka laukaistaan kun jompaakumpaa kytkintä painetaan.

```
<!-- HTML koodi -->
```

```
<div class="huone">
    <div class="lamppu pois-paalta"></div>
    <div class="kytkin"></div>
    <div class="kytkin"></div>
</div>

//JavaScript
// lampun kytkeminen päälle tai pois päältä
$(".lamppu").on("vaihdaTilaa", function () {
  var lamppu = $(this);
  if (lamppu.hasClass("pois-paalta")) {
    lamppu.hasClass("paalla").removeClass("pois-paalta");
  } else {
    lamppu.hasClass("pois-paalta").removeClass("paalla");
  }
});

//kytkinten toiminta: kytkin laukaisee painettaessa "vaihda-
Tilaa" tapahtuman.
$(".kytkin").on("click", function () {
  $(".lamppu").trigger("vaihdaTilaa");
});
```
(jQuery Learning Center n.d.s)

5.13 Efektit

jQueryn avulla voidaan sivuille luoda monenlaisia efektejä, kuten animaatioita ja häivytyksiä. Siinä on sisäänrakennettuna muutamia useinten käytettyjä animaatioita, mutta se tarjoaa myös kattavan tavan luoda omia, monimutkaisia animaatiosekvenssejä. (jQuery Learning Center n.d.u)

Efektejä käytetään kuten mitä tahansa muita jQuery-metodeja: valitaan elementti johon kohdistetaan efektifunktio. Sisäänrakennetuista efekteistä yksinkertaisimpia ovat ”.hide()” sekä ”.show()”, jotka vain piilottavat tai näyttävät valitun elementin muutamalla sen ”display” -arvon ”none”:ksi.

```
$("p").hide();
$("p").show();
```

Efekti-metodeille voidaan usein antaa argumentiksi myös tapahtuman kesto millisekunteina. Millisekuntien lisäksi jQueryssä on muutama esiasetettu ajan määrite, ”slow”, ”normal” ja ”fast”, jotka vastaavat 600, 400 ja 200 millisekuntia. Esimerkiksi jos ”.hide()”-metodille annetaan argumentiksi ”500”, metodi muuttaa piilottaminen lisäksi myös elementin korkeutta, leveyttä ja läpinäkyvyyttä puolen sekunnin ajan, jolloin näiden kaikkien arvo on nolla.

```
$(”.p”).hide(500);
```
(jQuery Learning Center n.d.u)

Jos kuitenkin halutaan vain häivyttää elementti ilman koon muuttamista, voidaan käyttää ”.fadeOut()”-metodia, jonka vastineena on metodi ”.fadeIn()”. Päinvastoin jos elementti halutaan ”liu’uttaa” pois näkyvistä ilman häivyttämistä, voidaan käyttää ”.slideDown()”-metodia ja tuoda elementti takaisin käyttäen ”.slideUp()”-metodia. Metodin ”.toggle()”-avulla voidaan

elementti tuoda näkyviin tai poistaa näkyvistä sen nykyisen tilan perusteella. (jQuery Learning Center n.d.u)

Yleinen väärinkäsitys on, että animaation kanssa ketjutetut metodit ajetaan vasta animaation loppumisen jälkeen. On huomattava, että animaatiometodi vain käynnistää animaation. Kun animaatio alkaa, vaihtaa animaatiometodi elementin CSS-arvoja käyttäen JavaScriptin ”setInterval”-looppia. Jos halutaan, että elementtiin kohdistetaan muita metodeja vasta animaation loppumisen jälkeen, tulee käyttää animaatioiden ”callback”-funktioita. Seuraavassa esimerkit molemmista tilanteista:

```
//Luokka "valmis" lisätään elementtiin virheellisesti heti
animaation alkaessa
$("p").hide(500).addClass("valmis");

//Luokka "valmis"-lisätään vasta animaation loputtua käyttäen
callback-funktiota.
$("p").hide(500, function () {
   $(this).addClass("valmis");
});
```

Huomaa että jälkimmäisessä esimerkissä voidaan callback-funktion sisällä viitata elementtiin käyttäen ”this”-avainsanaa. (jQuery Learning Center n.d.u)

Animaatioita voidaan kontrolloida myös käyttäen metodeja ”.stop()” sekä ”.delay()” että käyttäen jQueryn ”jQuery.fx”-objektia. Metodi ”.stop()”-pysäyttää välittömästi käynnissä olevan animaation, kun taas ”.delay()”-metodilla voidaan viivästyttää animaatioiden alkamista annetun arvon verran. ”jQuery.fx” sen sijaan on objekti, joka määrittää efektien implementoinnin. Esimerkiksi ”jQuery.fx.speeds”-aliobjekti sisältää määritykset aiemmin mainituille ”fast”, ”normal” sekä ”slow”-arvoille. Jos kehittäjä haluaa poistaa kaikki animaatiot käytöstä tai antaa käyttäjälle mahdollisuuden valita tämän, voi ”jQuery.fx.off”-arvolle antaa arvon ”true”. (jQuery Learning Center n.d.u)

5.13.1 Kustomoidut efektit

Jos edellisessä luvussa mainitut metodit eivät mahdollista haluttua lopputulosta, voidaan lähestulkoon mitä tahansa CSS-arvoja animoida käyttäen ”.animate()”-metodia. Metodin avulla voidaan animoida arvoja tiettyn arvoon asti tai suhteessa nykyiseen arvoon.

```
$("#animoitavaElementti").animate({
    left: "+=50",
    opacity: 0.25
  },
  300, // Kesto millisekunteina
  function() { // Callback-funktio
    console.log("valmis");
  }
);
```

Yleisin CSS-arvo, jota ”.animate()”-funktiolla ei voida animoida, on elementin väri. Värin animoiminen on mahdollista kuitenkin mahdollista käyttäen esimerkiksi jQuery Color-liitännäistä. (jQuery Learning Center n.d.v)

jQueryn versiosta 1.4-lähtien on arvoille voitu antaa myös ”easing”-arvoja, joiden avulla voidaan säädellä esimerkiksi siirtymisen kiihtyvyyttä. jQuery sisältää vain kaksi mahdollista ”easing”-tyyppiä, ”swing” sekä ”linear”. jQuery UI sen sijaan lisää monia ”easing”-tyyppejä näiden kahden rinnalle, joiden avulla voidaan luoda vaikkapa pomppiva pallo. (jQuery Learning Center n.d.v)

Esimerkki ”easing”-arvon asettamisesta:

```
$("#animoitavaElementti").animate({
  left: ["+=50", "swing"]
  opacity: [0.25, "linear"]
}, 300);
```

5.14 Ristiriitojen välttäminen käyttäessä muita kirjastoja

Kuten luvussa 5.1 mainittiin, jQuery-kirjasto (ja käytännössä kaikki sen liitännäiset) käyttävät jQuery-namespacea. Tästä syystä yhteensopivuusongelmia muiden JavaScript-kirjastojen kanssa ei pitäisi juurikaan ilmetä. Tähän on kuitenkin yksi poikkeus: luvussa 5.1 mainittiin, myös että jQuery käyttää $-symbolia aliaksena funktiolle jQuery. Jos jokin toinen kirjasto käyttää $-symbolia muuttujana, voi ongelmia syntyä. Tällöin jQuery tulee asettaa niin kutsuttuun ”no-conflict”-tilaan heti kun jQuery-kirjasto ladataan sivulle käyttäen jQuery.noConflict()-metodia. jQuery voidaan asettaa ”no-conflict”-tilaan kolmella tavalla. (jQuery Learning Center n.d.w)

Ensimmäinen näistä on jQuery-aliaksen vaihtaminen. Sen sijaan että sivusto käyttäisi $-symbolia jQueryn aliasta, voidaan se asettaa miksi tahansa muuksi. Esimerkiksi asetetaan jQuery;n oikotiesymboliksi ”$jq”. Tämän jälkeen tulee ”$jq”-symbolia käyttää aina viitattaessa jQuery-funktioon (jQuery Learning Center n.d.w)

```
var $jq = jQuery.noConflict();
$jq(”#nappi”).html();
```

Toinen tapa on käyttää välittömästi käytettävää funktiolauseketta. Välittömästi kutsuttava funktiolauseke on JavaScript-tekniikka, jossa funktiota kutsutaan heti sen määrittämisen jälkeen. Tämä tapahtuu kirjoittamalla funktion määritelmä sulkeiden sisälle. Samalla tekniikalla voi jatkaa jQueryn käyttämistä $-symbolin avulla myös konfliktitilanteissa: vapautetaan $-symboli muille kirjastoille, luodaan välittömästi ajettava funktiolauseke joka ottaa vastaan argumenttina jQuery-funktion ja sen sisälle kirjoitetaan jQuery-koodi. (jQuery Learning Center n.d.w)

```
jQuery.noConflict();
function ($) {
  // jQuery-koodi tulee aaltosulkeiden sisälle
}(jQuery));
```

Kolmas tapa on käyttää ”$(document).ready()”-funktion argumenttia. ”$(document).ready()”-funktio ottaa vastaan jQuery-funktion argumenttina, jolloin $-symboli voidaan asettaa jQueryn käyttöön funktiota ajettaessa. (jQuery Learning Center n.d.w)

```
jQuery(document).ready(function ($) {
  // jQuery koodi tulee tähän, käyttäen $-symbolia viitta
maan jQueryyn
});
```

6 LIITÄNNÄISET

jQuery-liitännäisten ideana on tehdä jotain ryhmälle elementtejä. Liitännäisten toteutustavasta johtuen voidaan myös jQueryn ytimeen kuuluvia metodeita, kuten ”.fadeOut()” tai ”.addClass()” pitää myös liitännäisenä. Liitännäisiä voi tehdä itse mutta niitä on myös saatavilla tuhansia kappaleita internetissä. (jQuery Learning Center n.d.x)

jQuery-liitännäinen on yksinkertaisesti uusi metodi, joka jatkaa jQueryn prototyyppiobjektia. Prototyyppiobjektia jatkaessa kaikki jQuery objektit saavat kaikki uudet, lisätyt metodit. Aina kun jQuery()-funktiota kutsutaan, luodaan uusi jQuery -objekti joka perii kaikki jQueryn sisäiset metodit. (jQuery Learning Center n.d.x)

jQuery-liitännäisten laatu voi vaihdella paljon. Monet liitännäisistä on laaja-alaisesti testattuja ja hyvin ylläpidettyjä, toiset taas nopeasti tehtyjä ja sitten hylättyjä. Monet myöskään eivät noudata liitännäisiin liittyviä parhaita toimintatapoja. Osaa liitännäisistä, pääasiassa jQueryn käyttöliittymäkirjastoa jQuery UI:ta (http://jqueryui.com/), ylläpitää jQueryn kehittäjät. Näiden liitännäisten laatua voidaan pitää jQueryä vastaavina. (jQuery Learning Center n.d.y)

Helpoin tapa löytää liitännäisiä on yksinkertaisesti etsiä niitä hakumoottorilla, kuten Googlella tai käyttäen jQueryn omaa liitännäisrekisteriä-rekisteriä osoitteessa http://plugins.jquery.com/. Liitännäisiä etsiessä on hyvä pitää mielessä muutamia asioita: kehittäjän tulee tarkastaa että liitännäinen on hyvin dokumentoitu ja että sen tekijä antaa tarpeeksi esimerkkejä liitännäisen käyttämisestä. Joskus liitännäiset tekevät paljon enemmän kuin mitä on tarpeen, mikä luo tarpeetonta raskautta sivustolle. (jQuery Learning Center n.d.y)

jQuery-liittännäisen lataamisen jälkeen se yksikertaisesti lisätään projektiin kopioimalla liitännäinen projektikansioon ja lisätään se sivustolle ”<script>”-tagin avulla jQueryn jälkeen. (jQuery Learning Center n.d.y)

```
<script src="js/jquery-min.js"></script>
<script src="js/jquery-liitannainen.js"></script>
```

6.1 Liitännäisten luominen

Liitännäinen luodaan lisäämällä funktio jQueryn $.fn-metodiin. Tämä tapahtuu käyttämällä piste-merkintätapaa. Seuraavassa esimerkissä luodaan jQuery-funktio ”muutaPunaiseksi”, joka muuttaa valitut elementit punaisiksi.

```
$.fn.muutaPunaiseksi = function () {
  this.css("color", "red");
}

$("a").muutaPunaiseksi() // muuttaa kaikki linkit punaisiksi
```

Huomaa että ”.css()”-metodin käyttämiseen käytetään viittausta ”this” ”$(this)”-viittauksen sijaan, koska ”muutaPunaiseksi”-funktio on osa samaa objektia kuin ”.css()”. (jQuery Learning Center n.d.z)

jQueryn yksi tärkeimmistä ominaisuuksista on ketjuttaminen joka käytiin lävitse luvussa 5.4.2. Kuten luvussa mainittiin, jQuery-metodit palauttavat yleensä alkuperäisen jQuery-objektin metodin päättyessä. Edellisen ”muutaPunaiseksi”-funktion muuttaminen ketjutettavaksi on yksinkertaista: funktion päättyessä palautetaan jQuery-objekti lisäämällä ”return this”. (jQuery Learning Center n.d.z)

```
$.fn.muutaPunaiseksi = function () {
  this.css("color", "red");
  return this;
};

$(“a”).muutaPunaiseksi().addClass(“esimerkki”)
```

Luvussa 5.14 läpikäyty konfliktien välttäminen pätee myös jQuery-liitännäisiin. Vaikka ”jQuery.noConflict()”-metodia käytettäisiin, ovat liitännäiset yleensä kirjoitettu olettaen, että $-muuttuja on alias jQuerylle. Jotta jQuery-liitännäiset toimisivat yhdessä muiden liitännäisten kanssa (samalla käyttäen $-aliasta), tulee koodi kirjoittaa välittömästi kutsuttavan funktiolausekkeen sisälle, kuten luvussa 5.14 esitettiin. (jQuery Learning Center n.d.z)

```
(function ($) {
  $.fn. muutaPunaiseksi = function () {
    this.css("color", "red");
    return this;
  };
}(jQuery));
```

Tämä myös mahdollistaa niin kutsuttujen privaattimuuttujien käyttämisen. Esimerkiksi seuraavassa liitännäiselle annetaan privaattimuuttuja ”vari”, jota ei pysty muokkaamaan liitännäisen ulkopuolelta. (jQuery Learning Center n.d.z)

```
(function ($) {
  $.fn.esimerkki = function () {
```

```
    var vari = "#123456";
    this.css("color", vari);
    return this;
  };
}(jQuery));
```

Hyvänä toimintatapana on käyttää vain yksi $.fn-funktiopaikka yhtä liitännäistä kohden. Tämä vähentää tilanteita, joissa liitännäinen ajaa toisen liitännäisen yli. Jos liitännäinen voi suorittaa useita eri toimintoja, tulee nämä erotella toistaan käyttäen parametreja sen sijaan että liitännäinen loisi näille useita $.fn-funktioita. (jQuery Learning Center n.d.z)

```
(function ($) {
  $.fn.esimerkki = function (savy) {
    var tumma_vari = "#006400",
        vaalea_vari = "#98FB98";

    if (savy === "tumma") {
      return this.css("color", tumma_vari);
    }

    if (savy === "vaalea") {
      return this.css("color", vaalea_vari);
    }
  };
}(jQuery));
```

Kuten luvussa 5.6 todettiin, tyypillinen jQuery-objekti sisältää viittauksia useisiin DOM-elementteihin jonka vuoksi jQuery-objekteja kutsutaan yleensä kokoelmiksi. Kun halutaan tehdä muutoksia tietyille elementeille, tulee ”.each()”-metodia käyttää näiden läpikäymiseen.

```
$.fn.toinenEsimerkki = function () {
  return this.each(function () {
    //jotain toiminnollisuutta
  });
}
```

Huomaa että esimerkissä palautetaan “.each()”-metodin tulokset ”this”:n sijaan. Koska ”.each()”-metodi on jo valmiiksi ketjutettava, palauttaa se ”this”:n jonka taas esimerkkifunktio palauttaa eteenpäin. Tämä on parempi tapa ylläpitää ketjutettavuutta. (jQuery Learning Center n.d.z)

Kun liitännäiset tulevat monimutkaisimmiksi, on yleensä hyvä tapa mahdollistaa liitännäisen kustomointi hyväksymällä asetuksia käyttäjältä. Helpoin tapa mahdollistaa tämä on käyttäen objekteja sekä ”$.extend()”-metodia. Seuraavassa esimerkissä ajetaan esiasetettu väri käyttäjän värivalinnalla. (jQuery Learning Center n.d.z)

```
(function ($) {
  'use strict';
  $.fn.esimerkki = function (kayttajanvalinnat) {
    var asetukset = $.extend({
      //esiasetettu väri
      color: "#006400"
```

```
    }, kayttajanvalinnat);

    return this.css("color", asetukset.color);
  };
}(jQuery));

$("a").esimerkki({color: "#123456"});
```

Tällöin täytetään myös kaksi muuta hyviin toimintatapoihin liittyviin periaatteisiin. Muuttujan nimi ”color” vastaa CSS-arvoa ”color”, jolloin käyttäjä tietää välittömästi mihin asetus vaikuttaa. Tämän lisäksi asetus tarjoaa juuri oikean määrän kustomoimista: yksinkertainen syntaksi antaa täyden hallinnan elementtejä kohtaan. (jQuery Learning Center n.d.å)

Jos edellistä konseptia halutaan laajentaa, voisi esiasetetut muuttujat asettaa myös julkisiksi. Tällöin tavalliset muuttujat voidaan helposti yliajaa pysyvästi. Jos näin halutaan, lisätään asetukset funktio-objektiin käyttäen pisteesitystapaa.

```
(function ($) {
  'use strict';
  $.fn.esimerkki = function (kayttajanvalinnat) {
    var asetukset = $.extend({ }, $.fn.esimerkki.asetukset,
kayttajanvalinnat);
    return this.css("color", asetukset.vari);
  };

  $.fn.esimerkki.asetukset = {
    vari: "#006400"
  };
}(jQuery));
```

Tällöin esimerkkiväri voidaan ajaa ylitse pysyvästi kirjoittamalla seuraava rivi koodia mihin tahansa kohtaan ohjelman JavaScript-tiedostoa.

```
$.fn.esimerkki.asetukset.vari = ”#654321”;
```

Huomaa että ”$.extend()”-funktio yhdistää funktion esiasetetut asetukset sekä käyttäjän valinnat uuteen tyhjään objektiin, jotta alkuperäisiä ei ajeta ylitse. Tällöin on saatu yksi askel lisää kustomoituvuutta liitännäiseen ilman että liitännäisen koko on kasvanut juuri yhtään. Samaa periaatetta voi käyttää myös sisäisten funktioiden muokkaamiseen. On kuitenkin muistettavaa, että kaikille funktioille ei välttämättä ole järkevää antaa muokkaamismahdollisuutta, esimerkiksi tilanteissa joissa liitännäisen toiminta riippuu kyseisestä metodista. Nyrkkisääntönä voidaan pitää seuraavaa: jos et ole varma tulisiko funktio olla ulkopuolelta muokattavissa, ei sen luultavasti kannata olla. (jQuery Learning Center n.d.å)

Liitännäinen on aina kompromissi juostavuuden, tiedostokoon sekä performanssin välillä. Yksi liitännäinen ei pysty vastaamaan jokaiseen vaatimukseen mutta liitännäinen ei myöskään ole hyödyllinen jos se sisältää vähän tai ei yhtään muokkausmahdollisuuksia. Liitännäisen koon tulisi vastata sen toiminnollisuutta: yksinkertaisen liitännäisen tulisi olla hyvin pienikokoi-

nen. Optimoinnin kannalta tulisi miettiä, ovatko toiminnollisuuden performanssikustannukset niin pieniä, että niiden vaikutukset ovat loppukäyttäjälle hyväksyttävää. (jQuery Learning Center n.d.å)

Yleensä liitännäiset ovat tilattomia: niitä kutsutaan kokoelmaan elementtejä ja tämä on kaikki mitä liitännäinen tekee. On kuitenkin olemassa paljon toiminnollisuutta joka ei mahdu tämän kaltaisen konseptin sisälle. jQuery UI tarjoaa monipuolisemman liitännäissysteemin nimeltä Widget Factory. Vaikka Widget Factorya on voinut jQueryn versiosta 1.8. lähtien käyttää itsenäisesti ilman jQuery UI:ta, on se silti osa jQuery UI-kirjastoa eikä näin kuulu tämän opinnäytetyön piiriin. (jQuery Learning Center n.d.ä)

7 YHTEENVETO

jQuery on kirjastona erittäin käytännöllinen työkalu, joka helpottaa monen webkehittäjän elämää. Sen tarjoamat metodit paikkaavat mainiosti selainten sekä JavaScript-standardin puutteita, auttavat reunatapauksista seuraavien ongelmien välttämistä sekä yksinkertaistavat kirjoitettua koodia lähestulkoon poikkeuksetta. Sen massiivinen osuus kaikista käytetyistä JavaScript kirjastoista takaa koodiesimerkkien löytymisen, kattavan dokumentaation sekä monien hyödyllisten liitännäisten olemassa olon.

On kuitenkin muistettava, että jQueryn tehokas käyttäminen edellyttää myös kattavaa ymmärrystä natiivin JavaScriptin toiminnasta ja sen periaatteista. Voisi jopa sanoa, että jQuery tekee JavaScript-ohjelmoinnista liian helppoa: jQueryn yksinkertaisuuden takia kehittäjä saattaa huomaamattaan kirjoittaa epätehokasta, hankalasti muokattavaa koodia joka ulkoisesti näyttää hyvältä ja toimii oikein.

Yleisimpinä virheinä ovat liian laajojen valitsimien käyttö, koodin turha toistaminen, pitkät ja vaikeasti seurattavat metodi-ketjut ja epätehokas DOM:in muokkaaminen. Koska jQuery tarjoaa monia tapoja toteuttaa sama lopputulos, kuten esimerkiksi luvussa 5.1.1 mainittu elementtien siirtäminen, voi kehittäjä helposti erehtyä käyttämään ”väärää” metodia, joka tuottaa kuitenkin oikean lopputuloksen. jQueryn laajuuden takia kehittäjä ei välttämättä ole edes tietoinen paremmin ongelmaan soveltuvan metodin olemassa olosta.

Tämän takia kehittäjän tulisi aina epävarmoissa tilanteissa varmistaa metodien toiminnollisuus jQuery API:in dokumentaatiosta. Tällöin hyvä perustietämys JavaScriptin toimintamalleista takaa myös tehokkaan jQueryn käytön. DRY (Don’t Repeat Yourself, älä toista itseäsi)-periaatteen seuraaminen estää koodin turhan toistamisen ja auttaa luomaan yksinkertaista, helposti seurattavaa ja muokattavaa koodia.

jQueryn käyttäminen ei aina ole myöskään välttämätöntä. Yksinkertaiset koodinpätkät on monesti helppo kirjoittaa käyttämättä jQueryä lainkaan. Toisaalta jQuery ei myöskään ole paras tapa tehdä tiettyjä asioita. Se tukeutuu vahvasti DOM:in rakenteeseen, jolloin yksinkertaisen, useasti käytettävän koodin kirjoittaminen on monesti vaikeaa, jos ylläpidettävien komponenttien määrä on suuri tai niiden rakenne on monimutkainen.

jQuerylle on olemassa monia vaihtoehtoja natiivin JavaScriptin käyttämisen lisäksi, kuten ZeptoJS, MooTools ja Minified-ohjelmakirjastot, jotka tarjoavat osittain poikkeavia toiminnollisuuksia samankaltaisen syntaksin kautta. Jos halutaan siirtyä jQueryn DOM:in muokkaamiseen perustuvista periaatteista, voi kehittäjä käyttää esimerkiksi jotain kaavaimiin (template) perustuvaa ohjelmistokehystä kuten Googlen AngularJS:ää, BackboneJS:ää tai EmberJS:ää, joissa DOM:in muokkaamisen sijaan keskitytään mallin eli näkymän takana olevan datan muokkaamiseen. jQueryn käyttäminen yhdessä näiden kehysten kanssa on kuitenkin myös mahdollista tai joskus jopa pakollista.

jQueryn tarpeellisuus saattaa myös vähentyä tulevaisuudessa, kun ECMAScript-standardi, johon myös JavaScript perustuu, ja HTML5-standardin tuki yleistyvät ja selaimet kehittyvät. ECMAScriptin versio tuo jo monia jQuerystä löytyviä helpotuksia natiivisti selaimiin, kuten syntaksin jonojen yksinkertaiseen iterointiin ja webanimaatioiden luomiseen. Esimerkiksi HTML5-standardin Shadow DOM, joka ratkaisee DOM-puun kapselointiongelman, ja webkomponentit vähentävät edelleen tarvetta DOM:in raakaan muokkaamiseen kun HTML-elementit monipuolistuvat. Osaa näistä ominaisuuksista voi käyttää moderneissa selaimissa jo tänään, tai Polymer-kirjaston avulla. Selainten kehittyessä myös tarve tukea vanhoja, huonosti toimivia selaimia vähentyy.

Tällä hetkellä jQuery on kuitenkin ylivoimaisesti yksi käyttökelpoisimmista JavaScript-kirjastoista. Sen yleisyydestä johtuen sen ymmärtäminen on lähestulkoon pakollista jokaiselle JavaScript-kehittäjälle, vaikka kehittäjä itse ei jQueryä käyttäisikään. jQuery ei ole ratkaisu kaikkiin JavaScriptin ongelmiin mutta se tarjoaa helpon tavan lähestyä monia niistä. Se ei kuitenkaan tee automaattisesti koodista laadukasta vaan jQueryä käyttäessä tulee kehittäjän muistaa myös kaikki oikeaoppiset JavaScript-periaatteet.

LÄHTEET

Crockford, D. 2008. The world's most misunderstood programming language has become the world's most popular programming language. Viitattu 9.8.2014.
http://javascript.crockford.com/popular.html

Crockford, D. 2001a. The world’s most misunderstood programming language. Viitattu 9.8.2014.
http://javascript.crockford.com/javascript.html

Crockford, D. 2001b. Private members in JavaScript. Viitattu 9.8.2014.
http://www.crockford.com/javascript/private.html

Github n.d. jQuery-migrate: Migrate older jQuery code to jQuery 1.9+. Viitattu 23.8.2014.
https://github.com/jquery/jquery-migrate

jQuery Learning Center n.d.a. JavaScript 101 – Getting Started. Viitattu 9.8.2014.
http://learn.jquery.com/javascript-101/getting-started/

jQuery n.d.e. jQuery.noConflict(). Viitattu 23.8.2014.
http://api.jquery.com/jquery.noconflict/

jQuery Learning Center n.d.b. $ vs $(). Viitattu 23.8.2014
http://learn.jquery.com/using-jquery-core/dollar-object-vs-function/

jQuery Learning Center n.d.c. $(document).ready(). Viitattu 23.8.2014.
http://learn.jquery.com/using-jquery-core/document-ready/

jQuery Learning Center n.d.d. Selecting Elements. Viitattu 24.8.2014.
http://learn.jquery.com/using-jquery-core/selecting-elements/

jQuery n.d.e., Attribute Contains Selector [name*="value"]. Viitattu 24.8.2014.
http://api.jquery.com/attribute-contains-selector/

jQuery Learning Center n.d.e. Working with selections. Viitattu 24.8.2014.
http://learn.jquery.com/using-jquery-core/working-with-selections/

jQuery Learning Center n.d.f. Manipulating elements. Viitattu 24.8.2014.
http://learn.jquery.com/using-jquery-core/manipulating-elements/

jQuery Learning Center n.d.g. Reserved Words. Viitattu 24.8.2014.
http://learn.jquery.com/javascript-101/reserved-words/

jQuery Learning Center n.d.h. The jQuery-object. Viitattu 7.9.2014.
http://learn.jquery.com/using-jquery-core/jquery-object/

jQuery Learning Center n.d.i. Traversing. Viitattu 7.9.2014.

http://learn.jquery.com/using-jquery-core/traversing/

jQuery Learning Center n.d.j. CSS, styling & dimensions. Viitattu 7.9.2014
http://learn.jquery.com/using-jquery-core/css-styling-dimensions/

jQuery Learning Center n.d.k. Data methods. Viitattu 7.9.2014.
http://learn.jquery.com/using-jquery-core/data-methods/s

jQuery Learning Center n.d.l. Ajax. Viitattu 27.9.2014
http://learn.jquery.com/ajax/

jQuery Learning Center n.d.m. Key concepts. Viitattu 27.9.2014
http://learn.jquery.com/ajax/key-concepts/

jQuery Learning Center n.d.n. jQuery's Ajax-Related Methods. Viitattu 27.9.2014.
http://learn.jquery.com/ajax/jquery-ajax-methods/

jQuery Learning Center n.d.o. Deferreds. Viitattu 28.9.2014.
http://learn.jquery.com/code-organization/deferreds/

jQuery Learning Center n.d.p. jQuery Deferreds. Viitattu 28.9.2014.
http://learn.jquery.com/code-organization/deferreds/jquery-deferreds/

jQuery Learning Center n.d.q. jQuery Event Basics. Viitattu 5.10.2014
http://learn.jquery.com/events/event-basics/

jQuery Learning Center n.d.r. Triggering Event Handlers. Viitattu 5.10.2014
http://learn.jquery.com/events/triggering-event-handlers/

jQuery n.d.f. .on(). Viitattu 2.11.2014
http://api.jquery.com/on/

jQuery Learning Center n.d.s. Introducing Custom Events. Viitattu 25.10.2014
http://learn.jquery.com/events/introduction-to-custom-events/

jQuery Learning Center n.d.t. History of jQuery Events. Viitattu 25.10.2014
http://learn.jquery.com/events/history-of-events/

jQuery Learning Center n.d.u. Introduction to Effects. Viitattu 25.10.2014
http://learn.jquery.com/effects/intro-to-effects/

jQuery Learning Center n.d.v. Custom Effects with .animate(). Viitattu 25.10.2014
http://learn.jquery.com/effects/custom-effects/

jQuery Learning Center n.d.w. Avoiding Conflicts with Other Libraries. Viitattu 23.8.2014.
http://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/

jQuery Learning Center n.d.x. Plugins. Viitattu 6.9.2014.
http://learn.jquery.com/plugins/

jQuery Learning Center n.d.y. Finding & evaluating plugins. Viitattu 7.9.2014.
http://learn.jquery.com/plugins/finding-evaluating-plugins/

jQuery Learning Center n.d.z. Basic plugin creation. Viitattu 7.9.2014.
http://learn.jquery.com/plugins/basic-plugin-creation/

jQuery Learning Center n.d.å. Advanced plugin concepts. Viitattu 7.9.2014.
http://learn.jquery.com/plugins/advanced-plugin-concepts/

jQuery Learning Center n.d.ä. Writing stateful plugins with the jQuery UI Widget Factory. Viitattu 7.9.2014.
http://learn.jquery.com/plugins/stateful-plugins-with-widget-factory/

jQuery n.d.a. Viitattu 9.8.2014.
http://jquery.com/

jQuery n.d.b. Downloading jQuery. Viitattu 23.8.2014.
http://jquery.com/download/

jQuery n.d.c. License. Viitattu 23.8.2014.
https://jquery.org/license/

jQuery n.d.d jQuery API. Viitattu 23.8.2014.
http://api.jquery.com/

JSON n.d. Introducing JSON. Viitattu 17.08.2014.
http://json.org/

Methvin, D. 2013. jQuery 1.9 final, jQuery 2.0 beta, Migrate final released. Viitattu 23.8.2014.
http://blog.jquery.com/2013/01/15/jquery-1-9-final-jquery-2-0-beta-migrate-final-released/

Mozilla Developer Network n.d.a. EventListener.addEventListener. Viitattu 9.8.2014.
https://developer.mozilla.org/en/docs/Web/API/EventTarget.addEventListener

Mozilla Developer Network n.d.b. Introduction – What is the DOM? Viitattu 23.8.2014.
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Resig, J. 2006. jQuery 1.0. Viitattu 9.8.2014.
http://blog.jquery.com/2006/08/26/jquery-10/

W3Techs 2014. Usage statistics and market share of JQuery for websites. Viitattu 8.11.2014.
http://w3techs.com/technologies/details/js-jquery/all/all