OULUN AMMATTIKORKEAKOULU

Irina Korva

2D-GRAFIIKAN PIIRTÄMINEN JAVASCRIPTILLÄ

jQuery-kirjaston toteuttaminen grafiikan piirtämiseen

2D-GRAFIIKAN PIIRTÄMINEN JAVASCRIPTILLÄ

jQuery-kirjaston toteuttaminen grafiikan piirtämiseen

Irina Korva
Opinnäytetyö
Kevät 2016
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Web-sovelluskehitys

Tekijä(t): Irina Korva
Opinnäytetyön nimi: 2D-grafiikan piirtäminen JavaScriptillä
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi- ja vuosi: Kevät 2016 Sivumäärä: 28

Opinnäytetyön aihe tuli ohjaavan opettajan kautta. Opinnäytetyön toimeksiantaja on startup-yritys. Opinnäytetyön tavoitteena on luoda selainpohjainen sovellus, joka näyttää tehtaiden prosessi-instrumenttikarttoja. Sovellus tehdään toimimaan selainpohjaisesti käyttämällä uusia web-teknologioita, kuten HTML5-kieltä, CSS3-kieltä sekä erilaisia JavaScript-kirjastoja. Opinnäytetyö tulee keskittymään sovelluksen front-end –kehitykseen.

Opinnäytetyöprosessin alussa otetaan selvää erilaisista JavaScript-kirjastoista, joita voitaisiin hyödyntää opinnäytetyön aikana. Edellytyksinä kirjastolle ovat sen helppokäyttöisyys, maksuttomuus sekä muokattavuus, sillä työ on tarkoitus tehdä käyttäen avoimeen lähdekoodiin pohjautuvia kirjastoja.

Sovellus toimii demona, jonka pohjalta tullaan myöhemmin kehittämään ohjelmisto. Ohjelmisto osaa lukea prosessi-instrumenttikarttoja suoraan CAD-tiedostoista sekä pilkkoa ne dataksi. Ohjelmiston tarkoitus on edistää kunnossapitotöiden turvallisuussuunnittelua raskaan kemianteollisuuden alalla. Ohjelmisto parantaa prosessilaitteistossa tehtävien kunnossapitotöiden suunnittelua ja dokumentointia sekä luo järjestelmällisen ja yhdenmukaisen tavan valmistella turvallisuuden kannalta kriittisiä kunnossapitotöitä. Sovelluksen tehtävänä on näyttää kaavioita ja demonstroida sitä, kuinka ohjelmisto lukee prosessi-instrumenttikarttoja.

Asiasanat: 2D-grafiikka, JavaScript, jQuery, jQuery-kirjasto, JointJS, Kaaviot

ABSTRACT

Oulu University of Applied Sciences
Business Information Systems, Web application development

Author(s): Irina Korva
Title of thesis: Drawing 2D-graphics with JavaScript
Supervisor(s): Jouni Juntunen
Term and year when the thesis was submitted: Spring 2016 Number of pages: 28

The topic for the thesis was acquired through the thesis supervisor. The primary aim is to develop a web application that displays the process instrument charts for factories. The application will be cross-browser using the newest web technologies, such as HTML5, CSS3 and JavaScript. The focus of the thesis is in the front-end development.

In the beginning of the thesis, different JavaScript libraries are compared to find out which libraries can be used during the development. The requirements for the library are that it's easy to use, free of change and its license allowes it to be modified. The application will be made using open source libraries.

The application will serve as a demo for a software which will be developed later. The software will analyze process instrument charts from CAD-files and convert them to data. The purpose of the software is to help the planning maintenance work in the chemical industry. The software will improve the planning and documenting the equipment maintenance. It will also create a systematic and organised way to prepare the maintenance work which are critical to safety. The function of the application is to display charts and demonstrate how the software will analyze process instrument charts.

Keywords: 2D-graphics, JavaScript, jQuery, jQuery library, JointJS, Charts

SISÄLLYS

1 JOHDANTO

JavaScript on viime vuosina noussut yhdeksi suosituimmista ohjelmointikielistä verkkoympäristöissä. JavaScript on tuettuna kaikissa uusimmissa verkkoselaimissa – se ei tarvitse minkään lataamista tai asentamista sekä se toimii myös mobiiliympäristössä. JavaScriptin pohjalta on rakennettu useita kirjastoja, jotka helpottavat ohjelmistokehittäjien työtä. Tässä opinnäytetyössä perehdytään tarkemmin jQuery-nimiseen kirjastoon, joka on yksi suosituimmista sekä helppokäyttöisimmistä JavaScript-kirjastoista. Myös jQuery-kirjastolle on tehty omia kirjastoja, jotka nopeuttavat ohjelmistokehittäjän työtä pakkaamalla yleisimmin käytetyt toiminnot lyhyisiin metodeihin. Yksi osa opinnäytetyötä on luoda jQuery-kirjasto, jolla toteutetaan tarvittavan grafiikan piirtäminen.

Opinnäytetyön tavoitteena on luoda selainpohjainen sovellus, joka näyttää tehtaiden prosessi-instrumenttikarttoja. Prosessi-instrumenttikarttoja käytetään, kun suunnitellaan huoltotoimenpiteitä kemianteollisuuden tuotantolaitoksissa. Sovellus tulee havainnollistamaan, kuinka PI-karttoja voidaan katsoa suoraan selaimella. Mikäli tuotantolaitoksessa tulee tukos, lähes koko tuotantolaitos joudutaan sulkemaan sekä käymään manuaalisesti läpi suuria PI-karttoja, jotta tukos löydetään. Sovelluksessa näkyy hyvin yksinkertainen PI-kartta, jossa näkyy putkia sekä venttiilejä. Mahdollisen tukoksen sattuessa sovelluksen avulla nähdään, missä kohtaa putkistoa tukos on sekä mitkä venttiilit tulee sulkea, jotta tukos voidaan avata turvallisesti.

Sovellus on tarkoitus tehdä toimimaan selainpohjaisesti käyttämällä uusia web-teknologioita, kuten HTML5-kieltä, CSS3-kieltä sekä erilaisia JavaScript-kirjastoja. HTML5 sekä JavaScript-kirjastot tekevät sovelluksesta nopean sekä sujuvan käyttää, sillä ne ovat keveitä ladata. Nämä uudet teknologiat syrjäyttävät vanhempia teknologioita sekä mahdollistavat sen, että PI-karttoja voidaan katsoa suoraan selaimella ilman erityisiä liitännäisiä, kuten Flash playeria. Opinnäytetyöprosessin alussa otetaan selvää erilaisista JavaScript-kirjastoista, joita voitaisiin hyödyntää opinnäytetyön aikana. Edellytyksinä kirjastolle ovat sen helppokäyttöisyys, maksuttomuus sekä muokattavuus, sillä työ on tarkoitus tehdä käyttäen avoimeen lähdekoodiin pohjautuvia kirjastoja.

Sovelluksen tehtävänä on näyttää kaavioita ja demonstroida sitä, kuinka ohjelmisto lukee prosessi-instrumenttikarttoja. Se tulee piirtämään näytölle yksinkertaisen putkistokartan, jossa näkyy putkistoja sekä venttiilejä. Se myös näyttää, mitkä putket ovat kytköksissä mihinkin venttiileihin värjäämällä ne punaiseksi. Grafiikan piirtäminen toteutetaan luomalla putkille sekä venttiileille oma

jQuery-kirjasto, josta niitä voidaan kutsua ja niiden parametreja voidaan päivittää tarvittaessa. Sovelluksen on tarkoitus toimia jokaisella modernilla verkkoselaimella, myös mobiililaitteilla. Opinnäytetyö tulee keskittymään sovelluksen front-end –kehitykseen.

Sovellus toimii demona, jonka pohjalta tullaan myöhemmin kehittämään ohjelmisto. Ohjelmisto osaa lukea prosessi-instrumenttikarttoja suoraan CAD-tiedostoista sekä pilkkoa ne dataksi. Ohjelmiston tarkoitus on edistää kunnossapitotöiden turvallisuussuunnittelua raskaan kemianteollisuuden alalla. Ohjelmisto parantaa prosessilaitteistossa tehtävien kunnossapitotöiden suunnittelua ja dokumentointia sekä luo järjestelmällisen ja yhdenmukaisen tavan valmistella turvallisuuden kannalta kriittisiä kunnossapitotöitä. Sovellus tulee esittelemään ohjelmiston toimivuutta sekä havainnollistamaan sen tarpeellisuus.

2 JAVASCRIPT

JavaScript on kevyt järjestelmäriippumaton ja oliopohjainen ohjelmointikieli, jonka avulla verkkosivustot saadaan luotua interaktiivisiksi. Järjestelmäriippumattomuuden ansiosta JavaScript toimii samalla lailla käyttöjärjestelmästä sekä laitteesta riippumatta. JavaScript mahdollistaa HTML-elementtien hallitsemisen ja muokkaamisen verkkosivustolla. JavaScript on sisäänrakennettu kaikkiin suosituimpiin moderneihin verkkoselaimiin, kuten Internet Exploreriin, Mozilla Firefoxiin sekä Safariin. Tämä mahdollistaa sen, että JavaScript ei vaadi käyttäjältä minkään asentamista tai ulkopuolisia ohjelmistoja. (Mozilla Developer Network, viitattu 4.4.2016)

JavaScriptin yleisimpänä tuotantoympäristönä toimii verkkoselain, mutta sitä voidaan myös käyttää hybridisovelluksissa. Hybridi mobiilisovellus on web-teknologioita käyttäen toteutettu web-sovellus, joka ajetaan mobiililaitteessa omana erikseen asennettavana sovelluksenaan. (Kolme tapaa kehittää mobiilisovellus, viitattu 4.4.2016)

JavaScript on noussut viime aikoina erittäin suosituksi web-teknologiaksi nopeutensa sekä muokattavuutensa ansiosta. Se on syrjäyttänyt paljon vanhempia teknologioita, kuten Flashin, ja edelleen se on käytetyimpien teknologioiden joukossa, vaikka uusia web-teknologioita koitetaan tuoda julkisuuteen lähes päivittäin. (Why JavaScript Will Become The Dominant Programming Language Of The Enterprise, viitattu 9.5.2016)

JavaScriptin pohjalta on rakennettu useita kirjastoja (Kuva 1). JavaScript-kirjastot ovat erillisiä JS-tiedostoja, joihin on kirjoitettu erilaisiin käyttötapoihin soveltuvia hyödyllisiä toimintoja. Näitä funktioita voi sitten kutsua omassa tiedostossaan. Tämä säästää aikaa, sillä suurin osa toiminnoista on jo koodattu valmiiksi – kehittäjän täytyy vain löytää oikea JS-kirjasto, jota hyödyntää omassa projektissaan. JS-kirjastojen käyttö mahdollistaa sen, että koodi on toimivaa ja oikeaoppisesti kirjoitettua sekä sen lataaminen verkkoselaimessa on nopeaa. Kirjastojen lisäksi JavaScriptin pohjalta on rakennettu frameworkeja (ohjelmistokehys), kuten AngularJS sekä React. Nämä JS-frameworkit on yleensä suunniteltu helpottamaan kehittäjiä web-sovellusten suunnittelussa ja rakentamisessa. Ne ovat suuria kokonaisuuksia, jotka sisältävät useampia tiedostoja kuin normaali JS-kirjasto. (Mozilla Developer Network, viitattu 4.4.2016)

Name	10k	100k	Million	Entire Web
jQuery	↓6,907	↑70,446	↑936,149	↓51,962,923
Facebook for Websites	↑4,007	↑36,216	↑349,046	↑6,928,797
Google Hosted Libraries	↓3,711	↓28,352	↑203,774	↓10,381,935
Facebook SDK	↑3,230	↑21,614	↑287,250	↑4,966,372
html5shiv	↓2,993	↓21,353	↑247,885	↓9,424,447
Modernizr	↓2,874	↓19,368	↑172,782	↑5,504,267
jQuery UI	↓2,771	↑22,508	↓272,153	↓7,412,171
Twitter Platform	↓2,409	↓15,222	↓203,141	↓3,175,712
Google API	↑1,759	↑21,235	↓124,043	↓2,890,252
jQuery Cookie	↓1,554	↓9,769	↑116,458	↑2,649,312
SWFObject	↓1,433	↓8,894	↓88,700	↓2,307,938
yepnope	↓1,167	↑7,542	↑69,944	↓1,546
Fancybox	↓1,075	↑8,115	↓118,376	↓4,611,348
jQuery Form	↑925	↑7,892	↓119,839	↑5,302,741
jQuery NoConflict	–875	↓6,143	↓75,184	↓2,978,528
jQuery Cycle	↓861	↑7,507	↓82,652	↓3,554,964
jQuery Easing	↓832	↓7,136	↓104,586	↓4,871,233
jQuery Validate	↑777	↓5,553	↓64,219	↑1,306,043
Yahoo User Interface	↓760	↑4,633	↑48,898	↑1,102,994
Lightbox	↑739	↑6,372	↓103,224	↓3,223,375

Kuva 1 Käytetyimmät JavaScript-kirjastot 10 000 suosituimmalla verkkosivustolla (JavaScript Usage Statistics, viitattu 21.3.2016)

2.1 jQuery

JQuery on kevyt JavaScript-kirjasto, jonka ensimmäinen versio luotiin vuonna 2006 amerikkalaisen John Resigin toimesta. JQueryn tarkoitus on helpottaa sekä nopeuttaa JavaScriptin käyttöä verkkosivustoilla. JQuery nopeuttaa JavaScriptin kirjoittamista, sillä se pakkaa JavaScriptin yleisimmät toiminnot metodeihin, joita voi kutsua yhdellä rivillä koodia (Kuva 2). Se myös yksinkertaistaa JavaScriptin monimutkaisia ominaisuuksia, kuten Ajax-kutsuja sekä DOM:n, dokumenttioliomallin, käsittelyä. (Why jQuery is the Most Popular JavaScript Library, viitattu 9.5.2016)

jQuery

```
$ ('body') .css ('background', '#ccc');
```

JavaScript

```
Function changeBachground(color) {

    Document.body.style.background = color;

}

Onload="changeBackground ('red');"
```

Kuva 2 Verkkosivuston taustavärin muuttaminen jQueryllä sekä JavaScriptillä (JQuery vs. JavaScript: What's the Difference Anyway?, viitattu 20.5.2016)

JQueryn ominaisuuksia ovat HTM-kielen ja DOM:n käsittely, CSS-tyylien muokkaaminen, HTML-kielen tapahtumametodit, efektit ja animaatiot sekä Ajax. JQuery on vain yksi olemassa olevista JavaScript-kirjastoista, mutta tällä hetkellä se on suosituin sekä eniten laajennettavissa. Useat internetin suurimmista yrityksistä, kuten Google, Microsoft, IBM sekä Netflix, käyttävät jQueryä palveluissaan. (Why jQuery is the Most Popular JavaScript Library, viitattu 9.5.2016)

2.2 jQuery-kirjastot

JQuery-kirjastoja on tehty useaan eri tarkoitukseen ja ne ovat erittäin suosittuja web-kehittäjien joukossa, sillä suurin osa niistä perustuu avoimen lähdekoodin lisenssiin. Lisenssistä riippuen kehittäjällä on yleensä vapaat kädet sekä mahdollisuus muokata kirjastoja omaan käyttötarkoitukseen sopivaksi. Lisensseissä on kuitenkin eroja ja osa lisensseistä on tiukempia kuin toiset. Kehittäjän tulee olla tarkkana ja ottaa huomioon, että miten eri kirjastojen lisenssit toimivat toistensa kanssa. (JQuery vs. JavaScript: What's the Difference Anyway?, viitattu 20.5.2016)

Suosituimpia jQuery-kirjastoja ovat erilaiset kuvasliderit, kuvagalleriat, kalenterit, mobiilivalikot, lomakkeet sekä niiden validointi ja erilaiset interaktiiviset toiminnot sivustoilla. Tällaisia ovat mm. popup-mainokset sekä sivusta tulevat valikot. Lähes jokaisella nykyaikaisella verkkosivustolla on käytetty jotain jQuery-kirjastoa. (JQuery vs. JavaScript: What's the Difference Anyway?, viitattu 20.5.2016)

3 JQUERY-KIRJASTON LUOMINEN

JQuery-kirjaston virallinen nimi on jQuery Plugin (jQuery-lisäosa), mutta yleensä siitä puhutaan nimellä jQuery-kirjasto. On useita syitä, miksi kehittäjät luovat omia jQuery-kirjastojaan. Suurin syy on, että ei ole vielä kehitetty sellaista kirjastoa, josta löytyisi kaikki tarvittavat toiminnot omaan käyttötarkoitukseen. Joissain tapauksissa kehittäjät luovat kirjastoja myös tavanomaisiin tarkoituksiin. Tällöin syynä on yleensä se, että projekti halutaan pitää DRY-periaatteen mukaisena. DRY on lyhenne englanninkielisestä lauseesta "Don't repeat yourself", joka ohjelmistokehityksessä tarkoittaa toistuvien rakenteiden yhdistämistä ja lähdekoodin pitämisenä mahdollisimman lyhyenä sekä helposti luettavana. (Writing your own jQuery plugins, viitattu 22.3.2016)

Yksinkertaisen jQuery-kirjaston luominen on helppoa. Ensimmäiseksi luodaan normaali JavaScript-tiedosto. Hyvä käytäntö jQuery-kirjaston nimeämiselle on muotoilla nimi siten, että kirjaston tarkoitus tulee ilmi tiedostonimestä. Esimerkkinä opinnäytetyössä luotava kirjasto jquery.elements.js, jossa tullaan luomaan opinnäytetyössä tarvittavia elementtejä (Kuva 3). Näin jQuery-kirjaston erottaa heti eri JavaScript-tiedostojen listasta. (Writing your own jQuery plugins, viitattu 22.3.2016)

```
        <script src="js/jquery.min.js"></script>
        <script src="js/lodash.min.js"></script>
        <script src="js/backbone.min.js"></script>
        <script src="js/joint.min.js"></script>
        <script src="js/jquery.elements.js"></script>
        <script src="js/scripts.js"></script>
    </body>
</html>
```

Kuva 3 Projektin JavaScript-tiedostot lueteltuna. JQuery-kirjaston tiedostossa käytettyä oikeaoppista nimeämistapaa.

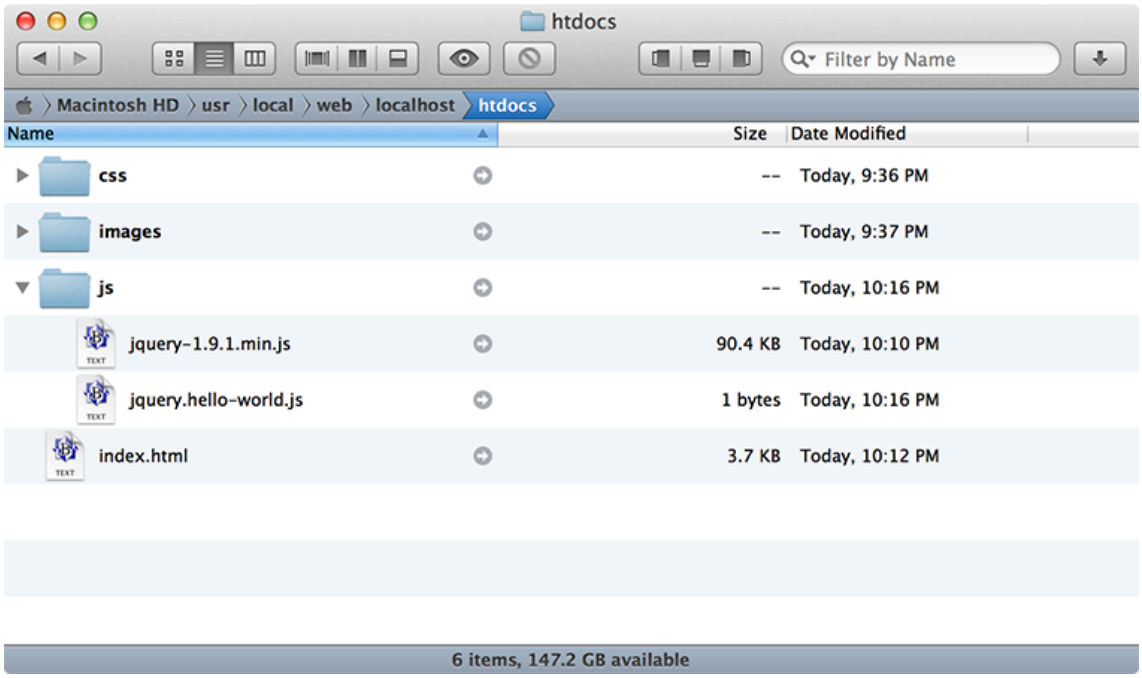
Omaa jQuery-kirjastoa tehtäessä kaikki kirjaston koodit on hyvä laittaa yhden funktion sisään. Tämä mahdollistaa sen, että kaikki kirjaston muuttujat pysyvät vain tämän funktion sisällä eivätkä aiheuta päällekkäisyyksiä muiden JavaScript-tiedostojen muuttujien kanssa. Näin koodi ei aiheuta turhia virheitä muualla sivustolla. Kirjaston koodeissa ei käytetä $-merkkiä, joka on yleensä käytössä jQuery-kirjastoissa. Sen sijaan lisäosassa voidaan käyttää $.fn.lisaosanNimi, esimerkiksi $.fn.helloWorld-yhdistelmää, jolla voidaan määritellä lisäosa (Kuva 4). Myös tämä on tärkeää, jotta

lisäosan koodit eivät sekoitu muiden JavaScript-tiedostojen koodien kanssa. (Writing your own jQuery plugins, viitattu 22.3.2016)

```
(function($) {

    $.fn.helloWorld = function() {

        // Future home of "Hello, World!"

    }

}(jQuery));
```

Kuva 4 Esimerkki jQuery-kirjaston koodirakenteesta (Writing your own jQuery plugins, viitattu 22.3.2016)

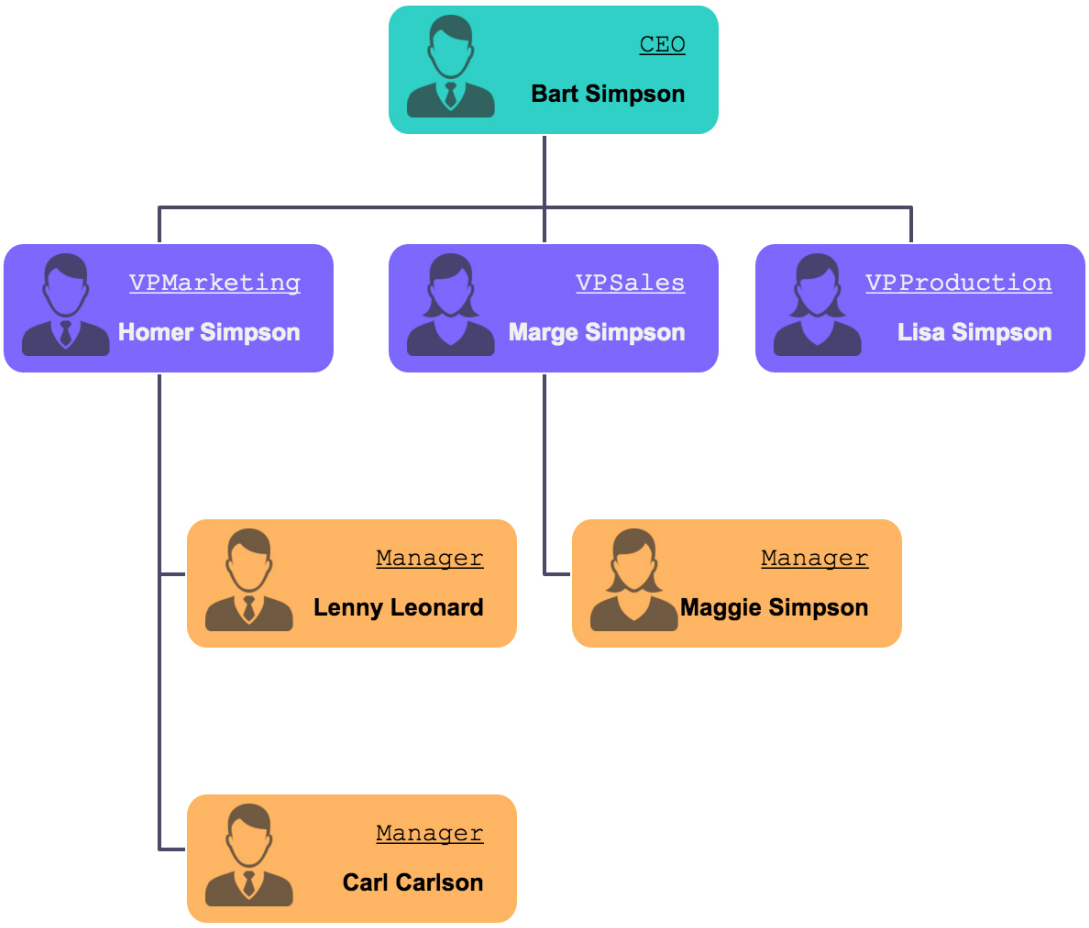
JQuery-kirjaston kansiorakenteen tulee olla selkeä. Suuremmissa jQuery-kirjastoissa on valtava määrä erilaisia tiedostoja, joilla jokaisella on oma tarkoituksensa. Kehittäjä ei aina tarvitse näitä kaikkia tiedostoja projektissaan vaan joissain tapauksissa hänen pitää saada helposti valita, mitkä tiedostot hän kirjastosta ottaa. Tämän vuoksi selkeä kansiorakenne sekä tiedostojen nimeäminen on tärkeää ja se on hyvä tehdä yleisten käytäntöjen mukaan. Yleinen käytäntö on se, että erityyppiset tiedostot laitetaan omiin kansioihinsa – JS-tiedostot ovat omassa kansiossaan sekä tyylitiedostot omassaan (Kuva 5). On myös hyvä luoda omat kansiot muille lisäosan erilaisille tiedostoille, kuten kuville sekä fonteille. (Writing your own jQuery plugins, viitattu 22.3.2016)



Kuva 5 jQuery-kirjaston oikeaoppinen kansiorakenne (Writing your own jQuery plugins, viitattu 22.3.2016)

4 JOINTJS

Työhön valittiin käytettäväksi JointJS-niminen kirjasto, jolla on mahdollisuus tehdä erilaisia interaktiivisia 2D-kaavioita (Kuva 6). JointJS on moderni HTML5-pohjainen JavaScript-kirjasto kuvaajien ja kaavioiden visualisointiin. Sitä voidaan käyttää niin staattisten kuin kokonaan interaktiivisten diagrammien luontiin. (JointJS, viitattu 10.3.2016)



Kuva 6 JointJS-kirjastolla luotu kaavio (JointJS, viitattu 10.3.2016)

JointJS on hyvä pienille avoimen lähdekoodin projekteille. Suurempiin projekteihin kirjaston kehittäjät suosittelevat Rappid HTML5 diagramming UI framework –kirjastoa, joka on rakennettu JointJS-kirjaston pohjalta. Rappid laajentaa JointJS-kirjastoa sekä tuo siihen 40 erilaista komponenttia, kuten käyttöliittymävimpaimia, jotka nopeuttavat diagrammi- ja kaavio-ohjelmistojen kehitystä. (JointJS, viitattu 10.3.2016)

JointJS mahdollistaa usean eri back-end –teknologian yhdistämisen kirjastoon. JointJS on kehitetty käyttämällä sellaista ohjelmointitapaa, jossa ohjelman keskeiset toiminnot perustuvat käyttäjän interaktiivisuuteen. Tällaisia ovat muun muassa käyttäjän aiheuttamat tapahtumat, kuten hiiren sekä näppäimistön klikkaukset. (JointJS, viitattu 10.3.2016)

Tällä hetkellä JointJS tukee Safaria, Google Chromea, Firefoxia, Internet Explorer 9:ää ja sitä uudempia versioita sekä Opera 15 ja sitä uudempia versioita. JointJS tukee sekä Safarin että Google Chromen mobiiliversioita. (JointJS, viitattu 10.3.2016)

4.1 JointJS-kirjaston lisenssi

JavaScript, jQuery sekä suurin osa eri JavaScript-kirjastoista on julkaistu avoimen lähdekoodin lisenssillä. Lisenssistä riippuen tämä tarkoittaa sitä, että kuka tahansa saa käyttää sekä muokata kirjastoja omaan käyttötarkoitukseensa sopivaksi. Yleensä niitä voi myös käyttää kaupalliseen tarkoitukseen ilman erillisiä lisenssimaksuja. Avoimien lähdekoodien lisensseissä on kuitenkin eroja. Osassa lisensseistä lähdekoodien käyttö sallitaan ainoastaan jos julkaisee valmiin tuotoksen lähdekoodit näkyville. (Creative Commons, viitattu 29.5.2016)

JointJS-ydinkirjaston lisenssi on Open Source Mozilla Public License 2.0. Ydinkirjasto sisältää kaiken, jonka saa ladattua JointJS-kirjaston verkkosivujen Download-osiosta. Ydinkirjasto ei sisällä Rappidia, jolla on kaupallinen lisenssi. (JointJS License, viitattu 10.3.2016)

Mozilla Public License, eli MPL, on yksinkertainen tekijänoikeuslisenssi. MPL on tiedostotason tekijänoikeus ja se on suunniteltu kannustamaan osallistujia jakamaan muokkaamiaan koodejaan, mutta samalla mahdollistaen oman lähdekoodin yhdistämisen muihin avoimiin tai suljettuihin lähdekoodeihin vähäisillä rajoituksilla. (MPL 2.0 FAQ, viitattu 10.3.2016)

4.2 JointJS-kirjaston käyttöönotto

JointJS-kirjaston käyttöönotto on nopeaa sekä yksinkertaista. JointJS-kirjaston verkkosivuilla on dokumentaatio, jossa on latauslinkit tarvittaviin tiedostoihin. Tarvittavia tiedostoja ovat JointJS-kir-

jaston tyylitiedosto sekä JavaScript-koodit, jQuery-kirjasto sekä Backbone ja Lodash –nimiset JavaScript-kirjastot, joita JointJS käyttää toimintoihinsa. Pienimmissä JointJS-projekteissa HTML-rakenne on hyvin yksinkertainen ja siihen tarvitsee lisätä tiedostojen lisäksi vain yksi sisältö-div, jonka tunniste on myholder (Kuva 7).

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" type="text/css" href="css/joint.css" />
        <link rel="stylesheet" type="text/css" href="css/styles.css" />
        <script src="js/jquery.min.js"></script>
        <script src="js/lodash.min.js"></script>
        <script src="js/backbone.min.js"></script>
        <script src="js/joint.min.js"></script>
        <script src="js/scripts.js"></script>
    </head>
    <body>
      <div id="myholder">

      </div>
</html>
```

Kuva 7 Yksinkertaisen JointJS -projektin HTML-tiedosto

JointJS-kirjaston verkkosivuilla on muutamia esimerkkejä hyvin yksinkertaisista kaavioista. Tässä kaaviossa (Kuva 8) on käytetty kahta erilaista elementtiä, suorakulmiota sekä linkkiä. Jotta JointJS-kirjaston avulla luodut elementit saadaan näkymään, tulee JS-tiedoston alkuun alustaa graph-niminen muuttuja, joka tarkoittaa diagrammia, eli kaaviota. Tämän lisäksi luodaan paper-niminen muuttuja, johon alustetaan paperi, jonka sisään kaavio piirretään. Paperille annetaan muutamia parametreja, jotka määrittävät sen koon.

Kuva 8 Esimerkki yksinkertaisesta JointJS-kirjastolla luodusta kaaviosta (JointJS, viitattu 10.3.2016)

Seuraavaksi luodaan kaaviossa näytettävät elementit, eli suorakulmiot sekä niiden välillä oleva linkki. Nämä luodaan käyttämällä JointJS-kirjaston oletuselementtejä, joille tulee määritellä tiettyjä

tietoja, jotta ne saadaan näkyville. Suorakulmiolle tulee määritellä sijainti asettamalla x- ja y-koordinaatit. Nämä koordinaatit asetetaan pikseleinä. Esimerkiksi jos x-koordinaatti on 100, elementti on 100 pikseliä paperin yläreunasta. Koordinaattien lisäksi suorakulmiolle asetetaan koko pikseleinä sekä halutessaan sille voi määrittää muita ominaisuuksia. Tässä esimerkissä suorakulmioille on asetettu niiden taustaväri, sisältöteksti sekä tekstin väri.

JointJS-kirjastossa on sisäänrakennettu toiminto, jonka avulla samanlaisia elementtejä voi luoda useampaan kertaan. Tässä esimerkissä rect2-nimiseen muuttujaan on kloonattu ensimmäisen suorakulmion, rect-muuttujan, kaikki tiedot. Kloonaamisen jälkeen toisen suorakulmion sijaintia on muutettu 300 pikselillä käyttämällä translate-toimintoa, jotta suorakulmiot eivät ole päällekkäin. Kloonaamisen jälkeen suorakulmiot yhdistetään toisiinsa käyttämällä linkkiä. JointJS luo linkin automaattisesti kahden eri pisteen välille. Alkupisteeksi, eli sourceksi, on asetettu ensimmäinen suorakulmio ja päätepisteeksi, eli targetiksi, toinen suorakulmio. Lopuksi elementit lisätään sivustolle lisäämällä ne JointJS-kirjaston luomaan kaavioon käyttämällä kirjaston omaa addCells-funktiota (Kuva 9).

```
var graph = new joint.dia.Graph;

var paper = new joint.dia.Paper({
    el: $('#myholder'),
    width: 600,
    height: 200,
    model: graph,
    gridSize: 1
});

var rect = new joint.shapes.basic.Rect({
    position: { x: 100, y: 30 },
    size: { width: 100, height: 30 },
    attrs: { rect: { fill: 'blue' }, text: { text: 'my box', fill: 'white' } }
});

var rect2 = rect.clone();
rect2.translate(300);

var link = new joint.dia.Link({
    source: { id: rect.id },
    target: { id: rect2.id }
});

graph.addCells([rect, rect2, link]);
```

Kuva 9 Yksinkertaisen JointJS-kaavion JavaScript-koodi

4.3 JointJS-kirjaston turhien toimintojen poistaminen

JointJS on interaktiivinen JavaScript-kirjasto, joten siinä on paljon ominaisuuksia sisäänrakennettuna. Oletuksena JointJS-kirjaston avulla piirretyt kaaviot ovat klikattavia sekä niiden elementtejä pystyy raahaamaan. Mikäli näille toiminnoille ei ole tarvetta, ne on helppo poistaa yksinkertaisella CSS-koodilla (Kuva 10).

```
.element,
.element.basic.Rect,
.connection-wrap {
    cursor: default;
}

.link-tools .tool-remove,
.link-tools .tool-options,
.link .marker-arrowheads {
    display: none;
}

.marker-vertex-group {
    visibility: hidden;
}

.connection-wrap:hover {
    opacity: 0;
}
```

Kuva 10 Turhien toimintojen poistaminen CSS-tiedostossa

5 TOTEUTUS

Sovellusta varten on luotu oma jQuery-kirjasto, jonne on talletettu grafiikan luomiseen liittyvät koodit. Sovelluksessa on käytössä kahta erilaista elementtiä – putkia sekä venttiilejä.

5.1 Sovelluksen elementit

Venttiilin tehtävänä on näyttää siihen liittyvät putket. Koska putket ja venttiilit ovat eri elementtejä, toisiinsa liittyville putkille ja venttiileille on määritelty yhdistävä tunniste. Venttiili voi toimia niiden elementtien kanssa, jotka jakavat saman tunnisteen. Yhdessä nämä putket sekä venttiilit muodostavat putkiston. Venttiiliä klikatessa tarkastetaan sen tunnisteet sekä verrataan niitä muiden elementtien tunnisteisiin. Näin saadaan selville, mihin putkistoon venttiili kuuluu. Klikattu venttiili sekä kaikki sen putkistoon liittyvät putket ja venttiilit värjätään punaiseksi. Tarkoituksena on esitellä koko putkisto, johon klikattu venttiili on jollain tavalla osallinen.

Putken tehtävänä on yhdistää venttiilejä toisiinsa. Toisin kuin venttiili, putkea klikatessa se värjää vain itsensä sekä alku- ja päätepisteensä. Tämän tarkoituksena on esitellä vaihtoehtoa, jossa tiettyyn putkeen on tullut tukos ja halutaan tietää, mitkä venttiilit tulee sulkea, jotta tukos voidaan avata turvallisesti.

5.2 Elementtien luominen

Sovelluksen elementit luodaan jQuery-kirjastossa. Kirjastossa on alustettu oletusarvoinen putki ja venttiili, joiden sijainti- sekä tunnistetietoja muutetaan jokaiselle elementille tapauskohtaisesti. Sovelluksen päätiedostoon on luotu taulukot sekä putkille että venttiileille, jotka tallennetaan objekteina. Taulukot sisältävät muuttuvat tiedot jokaiselle yksittäiselle elementille. Päätiedosto välittää taulukot kirjastolle, joka käy läpi taulukoiden objektit luoden niistä joko oletusarvoisen tai taulukon objektin tietojen mukaisen elementin.

Kirjastossa venttiilille on asetettu koko pikseleinä sekä venttiilin kuva ja kuvan koko. Nämä tiedot ovat kaikilla venttiileillä samat. Näiden lisäksi venttiilille on asetettu oletusarvoinen sijainti- ja tunnistetieto. Oletusarvoiset tiedot voidaan ylikirjoittaa sovelluksen päätiedoston lähettämän taulukon mukaisesti (Kuva 11).

```
var valveParams = [
    { },
    {
        position: { x: 300, y: 30 },
        valvePoint: { 0:'A', 1:'B' }
    },
    {
        position: { x: 100, y: 200 },
        valvePoint: { 0:'B' }
    },
    {
        position: { x: 400, y: 200 },
        valvePoint: { 0:'B' }
    },
    {
        position: { x: 500, y: 400 },
        valvePoint: { 0:'C', 1:'D' }
    },
    {
        position: { x: 500, y: 0 },
        valvePoint: { 0:'C' }
    },
    {
        position: { x: 500, y: 240 },
        valvePoint: { 0:'D' }
    }
];
```

Kuva 11 Venttiili-elementtien taulukko

Putkille on asetettu putken väri sekä putken leveys. Nämä ovat putkien ainoat muuttumattomat tiedot. Putkille on myös alustettu oletusarvoinen tunnistetieto sekä kulmakoordinaatit. Oletuksena putki piirretään elementtien välille suorana viivana, mutta kulmakoordinaattien avulla sille voidaan asettaa mutkakohtia. Koska putki luodaan jo olemassa olevien elementtien välille, sen alku- ja päätepisteitä ei voida alustaa.

Mikäli kirjaston asettamia arvoja ei ylikirjoiteta, elementti luodaan oletusarvojen mukaisesti. Tästä syystä ensimmäistä elementtiä ei tarvitse muokata päätiedoston lähettämässä taulukossa.

5.3 Elementtien piirtäminen

5.3.1 Venttiilien piirtäminen

Venttiilien piirtäminen tapahtuu jQuery-kirjaston drawValves-funktiossa (Kuva 12). Päätiedostossa alustetussa taulukossa on määritelty venttiilien tiedot. DrawValves-funktiossa käydään läpi jokainen taulukon sisäinen objekti ja tarkistetaan, onko sillä muuttuvia tietoja. Mikäli muuttuvia tietoja ei ole, venttiili piirretään oletusarvoisesti. Jos venttiilillä on muuttuvia tietoja, ne käydään läpi ja oletusarvoiset tiedot ylikirjoitetaan niiden mukaisesti.

JointJS sisältää tarvittavat funktiot elementtien piirtämiseen sekä erilaisten muotojen ja kuvien alustamiseen. JointJS muodostuu useista luokista, jotka ohjeistavat sitä luomaan oikeanlaisen kuvion tai kuvan. Koska venttiili on kuva, käytetään JointJS-kirjaston Image-luokkaa. Varsinaisen piirtämisen JointJS toteuttaa addCells-funktiossa, johon välitetään aikaisemmin mainitusta Image-luokasta luotu olio.

Jokainen piirretty elementti on yksilöity omalla uniikilla tunnistetiedollaan, jonka JointJS luo elementin piirtämisen yhteydessä. Tätä kyseistä tunnistetietoa tulee käyttää aina kun halutaan tehdä viittauksia tai muutoksia piirrettyyn elementtiin. JointJS generoi tunnistetiedon aina sivunlatauksen yhteydessä. Tämän vuoksi se on erilainen jokaisella kerralla eikä sitä voi alustaa itse.

Putket yhdistetään venttiileihin JointJS-kirjaston antaman tunnistetiedon mukaisesti. Koska tunniste luodaan joka sivunlatauksella uudestaan, se tulee tallettaa uuteen taulukkoon, joka palautetaan päätiedostoon kaikkien venttiilien piirtämisen jälkeen.

```
$.fn.drawValves = function(graph, valveParams) {
    var valveIds =[];
    $(valveParams).each(function(k, valve) {
        var newValue = $.extend({}, defaultValve, valve);
        var valveImage = new joint.shapes.basic.Image(newValue);
        newValue.id = valveImage.id;

        // adds valves to paper
        graph.addCells([valveImage]);

        // This is the current valve unique id which we need when mapping pipe endpoints to valves.
        valveIds[k] = newValue.id;
    });
    return valveIds;
}
```

Kuva 12 Funktio venttiilien piirtämiseen

Venttiilien piirtämisen kutsuminen tapahtuu sovelluksen päätiedostossa (Kuva 13). Venttiilien piirtämisen jälkeen drawValves-funktio palauttaa piirrettyjen venttiilien tallennetut tunnistetiedot päätiedostoon, joka otetaan vastaan valveIDs-nimiseen taulukkoon, johon on lueteltu JointJS-kirjaston luoma tunnistetieto jokaiselle venttiilille. Taulukon tunnistetiedot ovat samassa järjestyksessä kuin venttiilitaulukon venttiilit.

```
// call valves from elements jQuery library
var valveIDs = $.fn.drawValves(graph, valveParams);
```

Kuva 13 Venttiilien piirtofunktion kutsuminen päätiedostossa

5.3.2 Putkien piirtäminen

Päätiedostossa alustetaan taulukko putkille (Kuva 14). Koska venttiilien tunnistetietoja ei tiedetä, käytetään niiden sijasta taulukon avaimia yhdistävinä tekijöinä. Taulukon avain on kasvava kokonaisluku, joka lähtee aina luvusta nolla. Se määrittää elementin sijainnin taulukossa. Esimerkiksi ensimmäisen elementin avain on nolla sekä toisen elementin avain on yksi.

```
var pipeParams = [

    {
        source: { id: 0 }, // From which valve the pipe starts
        target: { id: 1}, // To which valve the pipe ends
    },
    {
        source: { id: 1 },
        target: { id: 2 },
        vertices: [ { x: 390, y: 50 }, { x: 390, y: 20 }, { x: 90, y: 20 }, { x: 90, y: 100 },
                    { x: 30, y: 100 }, { x: 30, y: 440 }, { x: 350, y: 440 }, { x: 350, y: 220 } ],
        pipePoint: { 0:'B' }
     },
     {
        source: { id: 3 },
        target: { id: 1 },
        vertices: [ { x: 440, y: 110 }, { x: 290, y: 110 }, { x: 290, y: 50 } ],
        pipePoint: {0:'B'}
     },
     {
        source: { id: 4 },
        target: { id: 5 },
        vertices: [ { x: 50, y: 420 }, { x: 50, y: 120 }, { x: 540, y: 120 } ],
        pipePoint: {0:'C'}
     },
     {
        source: { id: 6 },
        target: { id: 4 },
        vertices: [ { x: 400, y: 260 }, { x: 400, y: 500 }, { x: 540, y: 500 } ],
        pipePoint: {0:'D'}
     }
];
```

Kuva 14 Putki-elementtien taulukko

DrawValves-funktion palauttaman valveIDs-taulukon arvoja käytetään yhdistämään putkille asetettujen alku- ja päätepisteiden, eli venttiilien, tunnistetiedot. Esimerkkinä putkitaulukon ensimmäisen objektin alkupiste, eli source, on nolla sekä sen päätepiste, eli target, on yksi. Näitä vastaavat venttiilin tunnistetiedot poimitaan valveIDs-taulukon kohdista nolla ja yksi. Tämä tehdään jokaiselle putkelle (Kuva 15).

```
$(pipeParams).each(function(key, pipe) {
    pipe.source.id = valveIDs[pipe.source.id];
    pipe.target.id = valveIDs[pipe.target.id];
});
```

Kuva 15 Venttiilien tunnistetietojen päivittäminen putkien taulukkoon

Tunnistetietojen päivittämisen jälkeen voidaan kutsua drawPipes-funktiota, joka on toiminnaltaan samanlainen kuin drawValves-funktio (Kuva 12). Funktiossa käydään läpi jokainen putki ja sen mahdollisesti ylikirjoitettavat tiedot. Putki käyttää JointJS-kirjaston Link-luokkaa. Varsinaisen piirtämisen JointJS toteuttaa addCells-funktiossa, johon välitetään aikaisemmin mainitusta Link-luokasta luotu olio.

```
$.fn.drawPipes = function(graph, pipeParams) {

    // overwrite = updates the default values
    // newValue = overwritten value
    $(pipeParams).each(function(k, pipe) {
        var newValue = $.extend({}, defaultPipe, pipe);
        var pipeLink = new joint.dia.Link(newValue);
        graph.addCells([pipeLink]);
    });

}
```

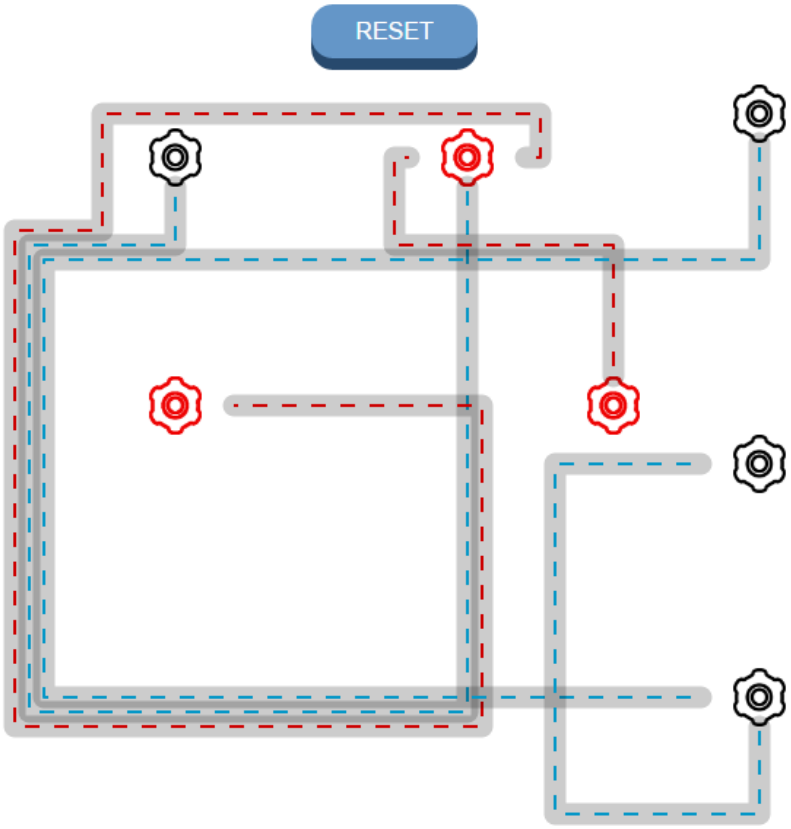
Kuva 16 Funktio putkien piirtämiseen

5.4 Sovelluksen toiminnot

JointJS tarjoaa useita eri toimintoja kaavioilleen, mutta suurin osa niistä on sellaisia, joita tässä sovelluksessa ei ole tarvittu. JointJS-kirjastolla on useita eri apufunktioita, joita on käytetty apuna sovelluksen toimintojen tekemiseen. Tällaisia toimintoja ovat putkien sekä venttiilien värjääminen, kun ne ovat aktiivisia sekä aktiivisten elementtien nollaaminen reset-nappia painettaessa.

Tärkein sovelluksen toiminto on venttiilien sekä putkien värjäytyminen punaiseksi. Punainen väri tarkoittaa, että ne ovat aktiivisia sillä hetkellä. Venttiilit ja putket muuttuvat aktiiviseksi, kun kursorin

vie niiden päälle tai niitä klikkaa. Ne muuttuvat aktiiviseksi myös silloin kun niihin liittyvät muut elementit muuttuvat aktiiviseksi (Kuva 17).



Kuva 17 Aktiivinen putkisto

Sovelluksen venttiilit on tehty kuvana, joten kuvan väriä ei voi muokata sovelluksessa. Sen sijaan venttiilin ollessa aktiivinen, sen kuva muutetaan JavaScriptin avulla kuvaksi punaisesta venttiilistä. Tämä tapahtuu muuttamalla venttiilin kuvapolkua. Sovelluksen putket on tehty käyttämällä JointJS-kirjaston omaa linkkiä, joten niitä pystyy tyylittelemään päätiedoston koodissa. JointJS käyttää kaavioissaan SVG-elementtejä, joten niiden tyylittely on hieman erilaista kuin normaalin HTML-elementin tyylittely CSS- tai JavaScript-koodilla. Sovelluksessa se tapahtuu muuttamalla putken väri, eli linkin stroke-attribuutti, punaiseksi. Lisäksi elementeille lisätään tunniste active, joka tarkoittaa niiden olevan aktiivinen (Kuva 18).

```
if (cellView.model.isLink()) {
    var link = cellView.model;
    var source = link.getSourceElement();
    var target = link.getTargetElement();

        source.attr({
            image: { 'xlink:href': 'js/img/venttiiliClicked.png' }
        });
        target.attr({
            image: { 'xlink:href': 'js/img/venttiiliClicked.png' }
        });

        link.attr({
            '.connection': { stroke: 'red' }
        });

        if (active) {
            source.attr({'active' : 'true'});
            target.attr({'active' : 'true'});
            link.attr({'active' : 'true'});
       }

}
```

Kuva 18 Venttiilin sekä putken värin muuttaminen

6 POHDINTA

Opinnäytetyön aihe itsessään oli todella mielenkiintoinen, sillä se on erilainen sekä sellainen, jota en olisi ikinä lähtenyt esimerkiksi vapaa-ajalla toteuttamaan. Aluksi mietin, onko opinnäytetyö liian haastava, sillä siinä tuli paljon uutta asiaa. Muutaman kerran opinnäytetyöprosessin aikana olin jo lähellä luovuttaa, koska ajauduin useasti umpikujaan koodin kanssa. Onneksi en luovuttanut vaan uskalsin haastaa itseäni sekä jouduin oikeasti pistämään aivonystyräni koville miettiessäni mahdollisia toteutustapoja.

Jo heti alussa suureksi ongelmaksi muodostui sopivan jQuery-kirjaston löytyminen. Tiesin, ettei aikani ei riittäisi massiivisen kirjaston tekemiseen alusta asti. Löysin paljon hyviä sekä monipuolisia kirjastoja 2D-grafiikan luontiin, mutta mikään niistä ei ollut juuri sitä, mitä hain. Lähes kaikki oli tarkoitettu suurien ja interaktiivisten kaavioiden luontiin sekä suurimmassa osassa niistä oli liikaa turhia ominaisuuksia, joita en tarvinnut. Tämä sai ne näyttämään raskailta sekä monimutkaisilta. Kirjaston tuli olla maksuton sekä sen lisenssin sellainen, että sitä saa vapaasti käyttää omassa projektissaan ja nämä myös loivat rajoituksia valittavalle kirjastolle.

JointJS ei ollut optimaalisin kirjasto tämän tyyppiseen projektiin, sillä se on alun perin tarkoitettu erilaisten interaktiivisten kaavioiden luontiin sekä niiden muokkaamiseen. Se oli kuitenkin sopivan yksinkertainen ja sen perusominaisuuksia pystyi hyödyntämään tässä projektissa sekä käyttämään niitä perustana sovelluksen luomiseen. JointJS-kirjaston dokumentaatio oli todella laaja, mutta omasta mielestäni monimutkainen – sieltä oli hankala löytää suoria sekä selkeitä esimerkkejä, joissa olisi ollut koodia valmiina sekä selitys sille, miten koodi toimii ja miksi se on tehty juuri sillä tavalla. Kirjasto ei ole erityisen suosittu eikä internetistä oikein löytynyt suoraa tietoa siitä, miten tietty asia oli toteutettu juuri tämän kirjaston avulla.

Opinnäytetyöprosessin aikana opin paljon uutta. Olin koulussa opiskellut vain hieman JavaScriptiä sekä käyttänyt sitä töissä muutamassa nettisivuprojektissa, mutta en ollut alusta asti kirjoittanut pitkiä pätkiä koodeja. Myös jQuery-arrayt olivat kokonaan uusi tuttavuus sekä oman jQuery-kirjaston luominen ja tietojen välittäminen kahden eri JS-tiedoston välillä. Myös funktioiden luominen sekä niiden kutsuminen olivat sellaisia asioita, joihin en ole aiemmin kiinnittänyt kovin paljon huomiota – yleensä koodini on ollut niin lyhyttä, että olen vain kirjoittanut kaiken samaan tiedostoon sen kummemmin jaottelematta.

Aluksi kirjoitin kaikki sovelluksen koodit yhteen JS-tiedostoon ja vasta jälkikäteen loin jQuery-kirjaston grafiikkaa varten. Tämä auttoi itseäni ymmärtämään, kuinka tärkeää koodin jaottelu on, sillä koodini selkeytyi huomattavasti tämän jälkeen. Sovelluksen ideana on, että sitä voitaisiin käyttää mahdollisesti esimerkkinä lopullisen ohjelmiston luomiseen, joten koodin tuli olla selkeää sekä helposti luettavaa. Koitin erityisesti näihin seikkoihin kiinnittää paljon huomiota kirjoittaessani koodia ja käytin paljon aikaa koodin sekä sen funktioiden kommentointiin.

LÄHTEET

Mozilla Developer Network, Viitattu 4.4.2016, https://developer.mozilla.org/en-US/docs/Web/JavaScript

Vuorinen C., Kolme tapaa kehittää mobiilisovellus. Viitattu 4.4.2016, http://w3.fi/kolme-tapaa-kehittaa-mobiilisovellus/

Wright N., Why JavaScript Will Become The Dominant Programming Language Of The Enterprise. Viitattu 9.5.2016, http://readwrite.com/2013/08/09/why-javascript-will-become-the-dominant-programming-language-of-the-enterprise/

JavaScript usage statistcs, Build With. Viitattu 21.3.2016, http://trends.builtwith.com/javascript

Chalkley A., Why jQuery is the Most Popular JavaScript Library. Viitattu 9.5.2016, http://blog.teamtreehouse.com/jquery-popular-javascript-library

Wilde B., JQuery vs. JavaScript: What's the Difference Anyway? Viitattu 20.5.2016, https://blog.udemy.com/jquery-vs-javascript/

Rotton, T. 2013. Writing your own jQuery plugins. Viitattu 22.3.2016, http://blog.teamtreehouse.com/writing-your-own-jquery-plugins

JointJS. Viitattu 10.3.2016, http://jointjs.com/