OULUN AMMATTIKORKEAKOULU

Joni Erkkilä

SALES-SUPPORT

Koodiviidakon sisäinen dokumentaatiosivusto

SALES-SUPPORT

Koodiviidakon sisäinen dokumentaatio sivusto

Joni Erkkilä
Opinnäytetyö
Lukukausi vuosi (Syksy 2015)
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Web sovelluskehitys

Tekijä(t): Joni Erkkilä
Opinnäytetyön nimi: Sales-support
Työn ohjaaja: Matti Viitala
Työn valmistumislukukausi- ja vuosi: Syksy 2015 Sivumäärä: 39

Projektin tarkoituksena oli rakentaa ja testata toimiva sivusto toimeksianto projektina. Toimeksianto saatiin Koodiviidakko Oy:ltä.

Sivuston oli tarkoitus tulla myynnin henkilöiden sisäiseksi dokumentaatiosivustoksi, jonne voidaan kootusti lisätä työhön liittyvää tietoa, sekä muita tärkeitä dokumentteja. Tämä projekti keskittyi ainoastaan sivuston rakentamiseen. Sisällön tuottaminen sekä suunnittelu rajattiin tämän projektin työn osuudesta.

Tietoperusta rakentuu vain niihin tekniikoihin joita sivuston rakentaminen vaati. HTML5 ja CSS3-tekniikat olivat keskeisessä osassa tämän projektin kannalta. Osa toiminnallisuudesta luotiin JavaScript-ohjelmointikielellä, ja loppuosa luotiin valmiiden lisäosien ja Bootstrap-sovelluskehyksen avulla. Puhdasta CSS-ohjelmointia ei harrastettu, vaan se korvattiin SASS-ympäristöllä. Sekä projektin hallinnan kannalta alkuun otettiin käyttöön git-versionhallintajärjestelmän, jonka käyttöönotto vaati oman hetkensä, jotta opittiin käyttämään järjestelmää.

Työ keskittyi ainoastaan sivuston rakentamiseen. Jatkotoimina jäädään jatkokehittämään sivustoa opinnäytetyöprosessin ulkopuolisena työnä.

Asiasanat: HTML, CSS, JavaScript

ABSTRACT

Oulu University of Applied Sciences
Business Information Systems, Web-design

Author(s): Joni Erkkilä
Title of thesis: Sales-support
Supervisor(s): Matti Viitala
Term and year when the thesis was submitted: Autumn 2015 Number of pages: 39

The goal of the project was to build website commissioned by Lianatechnologies ltd.

Website is to become an internal documentation site used by the sales team. Site is to become a collective place where all the relevant data regarding work can be stored and accessed by everyone. Projects main purpose was only build the site writing and collecting the data for the site was the excluded. Same policy was used on designing of the website.

Information used for the thesis reflects the technologies used in this projects. HTML5 and CSS3 where used to create the outlook of the site and JavaScript to add some functionality. And some functionality where able to obtain using several plugins and Bootstrap framework. Pure CSS programming wasn't used as much, since there is nowadays a better alternative for it called SASS. Especially in the beginning of building process git project management was in great help. Although this required first getting to know the and how it can be used.

After the project was finished content management could begging, which means some bugs are likely to emerge. These must be dealt with then and also there is a chance that some new development ideas are likely to emerge.

Keywords: HTML, CSS, JavaScript

SISÄLLYS

1 JOHDANTO

Opinnäytetyö toteutettiin toimeksiantona Koodiviidakko Oy:lle. Kehitystehtävänä oli luoda myynnin ryhmälle sisäiseen käyttöön verkkosivusto, johon kasataan kuvaukset yrityksen tuotteista. Sivustolle kootaan, tuotteiden ominaisuudet ja kuvaukset mitä voidaan toteuttaa ja mitä ei. Myös ryhmän työlle tärkeät lomakkeet ja lomakepohjat kasataan tälle sivustolle.

Vastaavanlainen sivusto on kasattuna jo tuotannon ryhmälle, mutta dokumentaatio tällä sivustolla keskittyy vastaamaan kysymyksiin miten toteutetaan. Tältä olemassa olevalta sivustolle saatiin suunnittelu, joten sitä ei tarvinnut toteuttaa erikseen ja voitiin ottaa mallia mitä ominaisuuksia sivustolle tulee. Toteutustekniikka on kummallakin sivustolla sama, kumpikin sivusto kasattiin käyttämällä yrityksen omaa tuotetta Sivuviidakkoa ja Bootstrap-sovelluskehystä.

Sivuston toteutus oli ollut suunnitteilla jo jonkin aikaa, mutta projektille ei ole pystytty suuntaamaan aikaa eikä resursseja, joten sen toteutus oli siirtynyt. Opinnäytetyön myötä sivuston toteutukselle pystyttiin ohjaamaan resursseja ja saatiin tekijä, jolla on aikaa.

2 TEKNIIKAT

2.1 HTML

2.1.1 Versiointi

Ensimmäinen version HTML:stä tuli ulos 1990. Tämä ensimmäinen versio oli tarkoitettu ainoastaan sähköisten dokumenttien ylläpitoon ja esittämiseen. Vasta kolmannen version HTML 3.0 kohdalla kieltä päivitettiin ja selkeää eroa saatiin aikaan verrattuna aikaisempiin versioihin. World Wide Web Consortium tai kuten se paremmin tunnetaan W3C, otti haltuun HTML-kielen kehityksen. 1998 HTML:stä valmistui versio 4.0, joka oli selvää kehitystä kohti nykyistä ohjelmointityyliä kohti. Suurin parannus, mitä neljänteen versioon lisättiin, oli ulkoisten tyylitiedostojen käyttö. Tyylitiedostoista puhutaan enemmän kappaleessa 2.2 CSS.

HTML5 on viimeisin versio, minkä virallinen julkaisu tapahtui vuoden 2014 puolella. Kuitenkin web-ohjelmoijat ovat käyttäneet uusinta versiota ennen, sen virallista julkaisua.

2.1.2 Kirjoittaminen

HTML ei ole ohjelmointikieli, vaan merkintäkieli. Koodia kirjoitetaan HTML-tiedostoon, jonka selain sitten muuntaa käyttäjälle ymmärrettävään muotoon. Tiedosto sisältää HTML-standardien mukaisia elementtejä ja näiden elementtien sisään voidaan lisätä normaalia tekstiä, kuvia tai viitteitä muihin tiedostoihin.

Muita paikkoja mihin kieltä voidaan käyttää, kuin nettisivujen luontiin on sähköposti kirjeet. Näiden kanssa on kuitenkin turvauduttava HTML-taulukkoelementtiin, jonka avulla voidaan rakentaa toimiva kirje. Sivustojen kanssa ei ole suositeltua, että rakenne tulee taulukoista. Myös puhelinsovelluksia voidaan rakentaa käyttämällä HTML-kieltä.

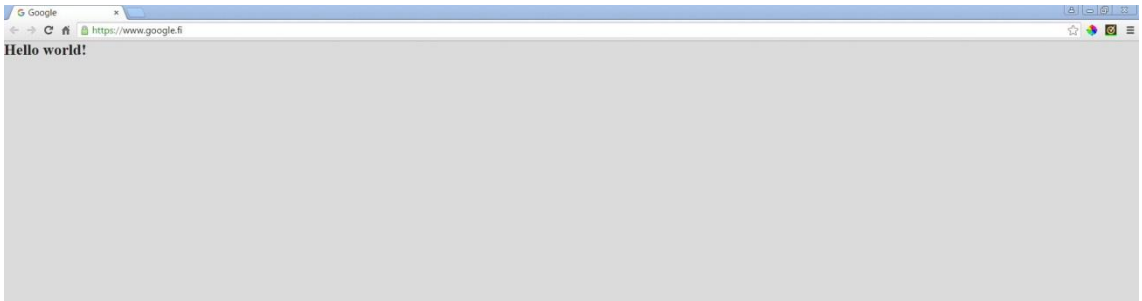
Yleensä elementit kirjoitetaan pareiksi, elementti saa aloitus- ja lopetustagin, esimerkiksi jos kirjoitetaan HTML-elementti. Aloitustag olisi <html> ja lopetustag olisi </html>. Lopetustag erotetaan /-merkillä, joka asetetaan aina ensimmäisen <-merkin jälkeen. Tähän sääntöön on kuitenkin olemassa poikkeuksia, kuten kuvaelementti joka ei saa erillistä lopetustagia.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tämä on tiedoston otsikko</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

KUVIO 1. Hello world –sivusto

Jokaisen tiedoston ensimmäisen rivin täytyy määritellä, mitä versiota HTML:stä tiedosto edustaa. Ylläkuvatussa esimerkissä oleva tiedosto on HTML5-standardin mukainen. Standardi määritteen jälkeen määritetään HTML-elementti, joka määrittelee koko dokumentin. Kaikki mitä tiedostoon kirjoitettava merkistö on oltava HTML-tagin sisällä, muutoin selain ei kykene kääntämään tiedostoa oikein.

Muut pakolliset elementit, jotka nettisivuilla vaaditaan ovat <head> ja <body>. Ensimmäisenä mainitun tarkoitus on antaa ohjeita selaimelle. Lukuisia tiedoston ominaisuuksia määritetään sinne, muun muassa otsikko, kieli, merkistö. Tyylitiedostot otetaan myös käyttöön <head>-tagin sisällä. Jos <head>-tag kertoo selaimelle, minkälaisesta tiedostosta on kyse, niin <body>-tag on se mihin määritellään sivun ulkoasu. Kuviossa kaksi HTML-tiedosto on esitetty Google Chrome selaimessa.



KUVIO 2. Hello world -sivusto selaimessa

Ennen HTML5:sta suositeltu standardi oli lisätä tarvittavat JavaScript-tiedostot käyttöön head-tagin sisällä. Nykyisin suositellaan niiden mukaan oton vasta body-tagin lopussa. Tämä järjestys takaa tehokkaamman sivun latauksen ja on myös huomattavasti loogisempi tapa. On huomattavasti järkevämpää ladata ensin tyylit, sitten elementit joiden avulla JavaScript kommunikoi tiedoston kanssa ja vasta näiden jälkeen itse JavaScript-tiedostot. JavaScriptin ja HTML:n välinen

kommunikaatio tapahtuu elementtien attribuuttien avulla, joten on tehokkaampaa ladata ensin HTML rakenne ja vasta sen jälkeen JavaScript. Lisää JavaScript ohjelmoinnista kappaleessa 4.

2.1.3 Attribuutit

Elementeille HTML:ssä voidaan antaa lisäarvoja, jotka muuttavat niiden käytöstä haluttuun suuntaa. Attribuutit ovat elementtikohtaisia, mikä tarkoittaa että tietyille elementeille voidaan antaa vain tiettyjä attribuutteja. Joidenkin elementtien kanssa on pakko käyttää attribuutteja, toimiakseen oikein. Esimerkiksi kuville täytyy aina määritellä src-attribuutti. Kun taas, teksti elementille ei voida määritellä src-attribuuttia.

```
<img src="kuva-1.jpg" alt="Tekstiä">
```

KUVIO 3. HTML-kuvaelementti

Esimerkissä on HTML-kuvaelementti, jolle on annettu kaksi attribuuttia. Ensimmäisenä kuvalle määritetään src, johon kirjoitetaan absoluuttinen polku mistä kuva löytyy ja kuvan nimi mukaan lukien kuvan tiedosto tyyppi. On vastoin HTML-standardeja jättää kuvan alt-attribuutti kirjoittamatta. Tällä attribuutilla ei ole mitään tekemistä kuvan toiminnallisuuden kanssa, kuva näkyy oikein vaikka alt-attribuuttia ei ole. Tämä toimii kuitenkin eräänlaisena kuvatekstinä, jota selain käyttää tilanteissa kun se ei kykene löytämään tai jaksa ladata kuvaa.

Kielen kehityksen myötä useat attribuutit ovat jääneet standardien ulkopuolelle. Niiden tuomat ominaisuudet otetaan käyttöön tyylitiedostoissa. Tällaisia attribuutteja ovat muun muassa:

- style
- bgcolor
- width
- height
- border
- color

Nämä attribuutit antavat elementille erilaisia tyyliasetuksia, kuitenkin nykyään on mahdollista lisätä elementeille id -tai class-attribuutti ja viitata sen avulla elementtiin ulkoisessa tiedostossa.

Syy miksi suositaan erillisiä tiedostoja, sekä id -ja class-attribuutteja on sen niiden vaikutus HTML-koodin luettavuuteen.

Attribuutit jakautuvat kahteen ryhmään normaaleihin ja tapahtuma-attribuutteihin. Tähän asti on kirjoitettu normaaleista attribuuteista. Tapahtuma attribuutit kirjoitetaan samalla lailla aloitus tagin sisään, niiden toiminta perustuu enemmänkin seuraamaan mitä kohtaa sivulla on klikattu, mitä sivulla näkyy tai näytön kokoa. Jos verrataan class-attribuuttia, jonka avulla HTML-tiedosto kykenee kommunikoimaan tyylitiedostojen kanssa, niin tapahtuma attribuutit kommunikoivat JavaScript-tiedostojen kanssa.

Uusimman standardin mukaan on liitetty data-* attribuutit. Nämä toimivat normaalien attribuuttien tavoin, mutta niiden käyttö on tilanteeseen riippuvainen ja niitä voidaan antaa mille tahansa elementille. Niiden käyttö on yksinkertaista, mikä tahansa attribuutti jonka nimi alkaa sanalla data on data-attribuutti. Tarkoitus on säilöä elementtiin tärkeää tietoa, joka voidaan hake JavaScriptillä ohjelman pyörittämistä varten. Data-attribuutteja ei käytetä vain tiedon tallentamiseen, niitä voidaan käyttää tyylitiedoston valitsijana. Valitsijoista enemmän kappaleessa 2.2.2.

2.1.4 HTML5

Julkaisun jälkeen HTML5:sta tuli W3C:n asettama standardi HTML-koodille, mikä tarkoittaa että kaikki ohjelmointi tulee tapahtua HTML5:lla. Kun standardi siirtyi kolmosesta neloseen mukaan ulkoiset tyylitiedostot, mikä oli iso muutos. Mitä siis viides versio tarjoaa uutena, joka tekee siitä paremman kuin edeltäjänsä.

- Uusia elementtejä
    - <article>
    - <section>
    - <header>
    - <nav>
    - <footer>
    - <iframe>
- Poistaa käytöstä vanhoja elementtejä
    - <font>
    -
    - <big>

- Uusia attribuutteja
    - Required
    - Placeholder
    - Autofocus
- Muuntaa ja korjaa vanhoja attribuutteja
- Poistaa käytöstä vanhoja attribuutteja

Mikäli sivun halutaan käyttävän utf-8 merkistö koodausta, merkitään se seuraavalla tavalla HTML5-standardin mukaan.

```
1
2       <meta charset="utf-8">
3
```

KUVIO 4. Merkistökoodaus lyhyellä mallilla

Sama merkistö koodaus otetaan käyttöön HTML4:ssa.

```
1
2       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
3
```

KUVIO 5. Merkistökoodaus pidemmällä mallilla

Kumpikin ratkaisu toimii nykyisten standardien mukaan, mutta HTML5:n mukainen ratkaisu vaati vain yhden attribuutin määrittämisen meta tag:lle pitäen koodin siistimpänä.

Ulkoinen tyylitiedosto otetaan käyttöön käyttämällä link-attribuuttia. HTML5:n ansiosta attribuutille pitää määrittää vain kaksi pakollista määrettä, aikaisemmin oli pakko lisätä vielä kolmas. Kolmas attribuutti, jota HTML5-standardi ei enää tarvitse on type=”text/css”. Tällä arvolla vielä alleviivattiin minkä tyyppinen on tiedoston sisältö.

```
<head>
    <meta charset="utf-8">
    <link href="css/style.css" rel="stylesheet">
</head>
```

KUVIO 6. Ulkoisen tyylitiedoston käyttöönotto

Ensimmäinen attribuutti href, kertoo mistä tyylitiedosto haetaan. Tämän esimerkin tapauksessa HTML-tiedoston vieressä on kansio css, jonka sisällä käyttöönotettava tiedosto. Rel-attribuutti määrittää link-attribuutille minkä tyylinen link-elementti on, tässä tapauksessa kyseessä on tyylitiedosto linkki. Mikäli link-elementti olisikin kirjoitettu seuraavalla tavalla.

```
<head>
    <link rel="canonical" href="http://www.sivu.fi/ensisijainen-url">
</head>
```

KUVIO 7. Kanooninen linkki

Olisi kyseessä hakukone optimointiin tarkoitettu elementti. Rel määrittelee, että kyseessä sivuston ensisijainen osoite, jota tulee käyttää kun hakukoneet etsivät sivustoja.

2.2 CSS

2.2.1 Tarkoitus

Sama järjestö, joka vastaa HTML-kielestä julkaisi ensimmäisen version CSS-kielestä vuonna 1996 hieman ennen kuin HTML siirtyi toisesta versiosta kolmanteen. Tämän jälkeen on tapahtunut kaksi suurempaa versio päivitystä, 1998 CSS2 ja seuraavana vuonna CSS3. Heti julkaisunsa jälkeen CSS:stä tuli osa ohjelmoinnin standardia ja siitä syystä kantaa isoa osaa jokaisella verkkosivulla, joka on netissä julkaistu.

CSS on lyhenne sanoista Cascading Style Sheets, jonka tarkoitus on lisätä muotoilua verkkosivulle. Muotoilun suhteen HTML on erittäin rajoittunut kieli etenkin jos ajatellaan sivuston ulkoasua. Attribuutteja käyttämällä voidaan esimerkiksi muuttamaan p-elementin tekstin tasausta, kuitenkin CSS tuo tähän paljon enemmän mahdollisuuksia. Alun perin CSS antoi mahdollisuuksia vaihtaa sivuston fontteja, taustavärejä tai ennalta mainittuja tasauksia. Ajan kuluessa ulkoasulliset mahdollisuudet ovat lisääntyneet. CSS:n vaikutus on niin suuri, että se on muuttanut koko tavan miten HTML-koodia kirjoitetaan. Samoin kuin HTML, CSS ei ole myöskään ohjelmointikieli.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <table width="900" border="1">
            <tr>
                <td width="450">
                    <h1>Vasen palsta</h1>
                </td>
                <td width="450">
                    <h1>Oikea palsta</h1>
                </td>
            </tr>
        </table>
    </body>
</html>
```

KUVIO 8. Kaksi palstainen sivu, taulukko rakenne

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>


        <div>
            Vasen palsta
        </div>

        <div>
            Oikea palsta
        </div>


    </body>
</html>
```

```css
div {

    display: inline-block;
    width: 450px;

}
```

Kuvio 9. Kaksi palstainen sivu, tyylejä hyväksi käyttäen

Kuviossa 8 tehdään taulukko ja käyttämällä taulukkoa luomaan ulkoasua, sekä attribuutteja saatiin luotua kaksi palstaa vierekkäin. Tämä on vanha tapa luoda sivustolle ulkoasu, ja vaatii myös huomattavasti enemmän kirjoittamista. Yllä olevassa kuviossa on moderni tapa rakentaa sivuston ulkoasua. Ensimmäisenä tiedoston head-elementin sisällä on link-elementti, joka hakee tyyli tiedoston sivulle. Body:n sisään on tehty kaksi div-elementtiä eikä mitään muuta. Tyylitiedosto määrittelee että kaikki div-elementti esitetään (display) yhdellä rivillä (inline-block) ja että elementtien leveys on 450 pikseliä. Tämä on huomattavasti siistimpi tapa esittää rakentaa ja esittää sivuja, sillä koodin jako on selkeä. Toinen etu CSS-ratkaisussa on mahdollisuus uudelleen käyttöön. HTML on vain sivuston rakenne ja tyylit sekä ulkoasu rakennetaan tyylien avulla.

2.2.2 Valitsijat

Valitsija	Määre	Arvo
p	color:	black;

Taulukko 1. CSS-valitsijan määritys

Yllä on esitetty esimerkki, miten valitsijoita voidaan käyttää. Esimerkki vaikuttaa kaikkiin p-elementteihin. Valitsija säätää elementin väri arvoa ja määrittelee, että kaikkien p-elementtien sisältö on väriltään musta. Jos verrataan style-attribuutin käyttöön, tarvitsee tällöin väriarvo kirjoittaa jokaiselle p-elementille erikseen. Määre erotetaan kaksoispisteellä arvosta, joka annetaan ja arvon jälkeen kirjoitetaan puolipiste, joka merkitsee selaimen kääntäjälle rivin vaihtoa.

Attribuutit kappaleissa puhuttiin id -ja class-attribuuteista, näitä käytetään erottelemaan elementtejä toisistaan. Ohjelmoija voi antaa joillekin p-elementeille id:n ja sille arvoksi, minkä tahansa valitsemansa nimen, vaikka ”sininen”. Tällöin tyyli tiedostoon voidaan kirjoittaa:

Valitsija	Määre	Arvo
#sininen	color:	blue;
.sininen	color:	blue;

Taulukko 2. Id ja class selectori

Kuviossa esitetty p-elementin valinta on vielä olemassa ja nyt sen jälkeen kirjoitetaan yllä esitetty valitsija. Risuaita nimen edessä kertoo kääntäjälle, että sen täytyy hakea HTML-elementtiä jolle on annettu id arvolla ”sininen”. Tällöin kaikki p-elementit pysyvät mustina paitsi, se mihin id ”sininen” muuttaa tekstin siniseksi. Jos taas nimen edessä on piste, on kyseessä class valitsija. Id ja class valitsijoilla ei varsinaisesti toiminnallisuuden puolesta mitään eroa, kumpikin toimii samoin. Pääero niiden välillä W3C:n mukaan, on että id on yksilöllinen jokaisella sivulla. Toisin sanoen sivua kohden saisi olla vain yksi id=”sininen”, mutta samoja class:ja voi olla useita.

Valitsijoita käytetään myös JavaScript-ohjelmoinnissa, ja jotkin ohjelmoijat ohjeistavat että id:tä käytettäisiin yksilöimään JavaScriptiä varten ja class:ja tyylejä varten. Kyse on kuitenkin yksilöllisistä tottumuksista ja jos id:tä ei käytetä tyyliohjelmoinnissa voi tyylitiedostosta tulla vaikea lukuista.

Tähän mennessä esitettyjä valitsijoita yhdistää yksi piirre, ne eivät tarjoa dynaamisuutta elementeille. Mitä jos halutaankin tekstin vaihtavan väriä kun hiiri on sen päällä. Tämän kaltainen tilanne vaatii pseudo-luokan käyttöä. Näille luokille on varattu omat nimensä, jolloin ne erotetaan kaksoispisteellä ja kirjoitetaan valitsijan perään. Toinen pseudo-ominaisuus on nimeltään pseudo-elementti. Tätä ominaisuutta voidaan käyttää esimerkiksi värjäämään tekstin ensimmäinen kirjain eri värillä.

Valitsija	Määre	Arvo
p:hover	color:	red;
p:first-letter	color:	red;

Taulukko 3. CSS pseudo-elementti

2.2.3 SASS

Yleinen ongelma CSS:n kirjoittamisessa on pitää tiedoston rakenne selkeänä ja siistinä. Mitä laajempi sivusto on kyseessä, sitä enemmän CSS-koodia sivustolle tulee, voidaan puhua useista tuhansista riveistä. On olemassa monia kehitelmiä pitää koodi selkeänä, yleinen neuvo on kommentoida koodia ja jäsentää se loogisesti. Tällöin on jatkokehityksenkin aikana suhteellisen helppoa ymmärtää tiedoston rakenne. BEM on ohjesääntö joka tarttuu tähän ongelmaan. BEM on lyhenne sanoista Block Elmenent Modifier ja se ohjeistaa tietynlaisen syntaksin miten nimetä CSS-valitsijoita.

```
/* BLOCK */
.block {
  font-family: sans-serif;
  font-size: 16px;
  padding: 50px 10px;
  margin: auto;
  max-width: 1560px;
  width: 100%;
}
/* ELEMENT */
.block__red { background-color: red; }
/* MODIFIER */
.block--padding0 { padding: 0; }
```

KUVIO 10. BEM-mallin mukainen nimeäminen

Jos sovelletaan BEM nimeämistä, ensimmäisenä luodaan block niminen luokka. Tätä voidaan ajatella perusluokkana joka annetaan kaikille HTML-elementeille joille tämä ominaisuus halutaan. Seuraava oleva block__red luokan tarkoitus on lisätä jotain uutta alkuperäiseen luokkaan. BEM erottelee tällaiset luokat nimeämällä ne seuraavasti:

- Ensimmäinen sana on sama, kuin perusluokka johon lisätään uusi ominaisuus
- Ensimmäisen sanan jälkeen lisätään kaksi ”_” – merkkiä
- Viimeisen sanan tarkoitus on kuvata uuta ominaisuutta, joka lisätään alkuperäiseen luokkaan.

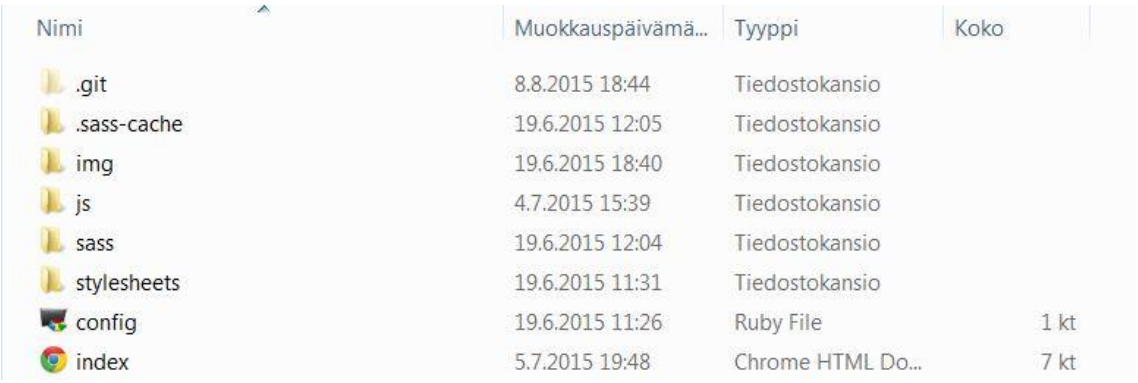
Viimeisenä on niin kutsutut modifier-luokat, joilla muutetaan jotain alkuperäistä ominaisuutta.

- Ensimmäinen sana on sama, kuin perusluokka jota muutetaan
- Ensimmäisen sanan jälkeen tulee kaksi ”-” – merkkiä
- Viimeinen sana kuvaa tätä ominaisuutta

Nämä ovat kuitenkin vain ohjeita, kuinka valitsijat tulisi nimetä ja jäsentää, mitään automaatiota tai erillistä sovellusta BEM:lle ei ole joten se on ohjelmoijan käsissä käyttääkö tätä järjestelmää.

Vaikka BEM tarjoaa selkeän tavan pitää CSS-koodin selkeänä on olemassa tehokkaampia tapoja hallita koodin syöttöä. SASS on CSS-koodin esikääntäjä, jonka avulla koodista pystyy kirjoittamaan selkeää ja helppolukuista, sekä turhien rivien määrä minimoituu.

Esikääntäjä toimii samalla periaatteella kuin selaimen kääntäjä. Kääntäjä tulkitsee tyylitiedoston koodin ja esittää ne valittuihin HTML-elementteihin. SASS ohjelmoinnissa luodaan omat scss-tiedostot, jotka käännetään CSS-koodiksi, jonka selain osaa tulkita.

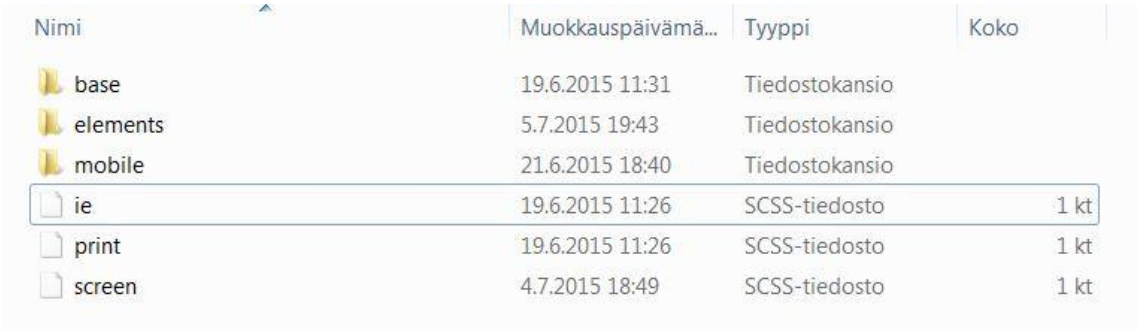


KUVIO 11. SASS-projektin tiedosto rakenne

Projektin kansio rakenne näyttää tältä. SASS:in toiminnan kannalta tärkeitä kansioita ovat:

- .sass-cache
- sass
- stylesheets

Ensimmäinen kansio, jota SASS käyttää välimuisti kansio tai kuten nimetty, sass-cache. SASS jäsentää luodut tiedostot tänne uudelleen käyttöä varten, tämä kansion päätarkoitus on nopeuttaa tiedostojen kasausta ja suositellaan isoissa projekteissa. Kansion luonnin sen päivityksen voi kytkeä pois päältä. Toinen tärkeä kansio on sass-kansio, tänne tehdään kaikki scss-tiedostot, jotka käännetään stylesheets-kansioon CSS-tiedostoiksi. HTML-tiedossa ei käytetä sass-kansion tiedostoja vaan stylesheets-kansion tiedostoja.



KUVIO 12. Tiedostot jaettuna kansioihin

Pääidea SASS:ssa on pilkkoa ohjelmoitavat tyylit pienempiin osiin, joita tässä tapauksessa ovat kolme kansiota esitettynä kuviossa, tuoda ne yhteen tiedostoon joka on tässä projektissa screen.scss tiedosto. Screen.scss on se tiedosto joka käännetään CSS-koodiksi. Esimerkkinä jos

sukelletaan elements-kansion sisään, löydettäisiin sieltä omat SASS-tiedostot sivuston yläosa -, navigaatio -, sisältö -, footer – osioille.

```
/* BASE FILES */
@import 'base/_variables.scss';
@import 'base/_base.scss';

/* ELEMENT FILES */
@import 'elements/_header.scss';
@import 'elements/_nav.scss';
@import 'elements/_content.scss';
@import 'elements/_footer.scss';

/* MEDIA QUERIES */
@import 'mobile/_media991.scss';
```

KUVIO 13. SASS tuo eri paikkoihin kirjoitetut tiedostot yhteen

Kaikki tiedostot tuodaan yhteen lopulta screen.scss-tiedostossa johon on kasattuna import-lauseilla kaikki scss-tiedostot jokaisesta kansiosta. Tiedostojen nimistä on tärkeää huomata kuinka ne alkavat alaviiva-merkillä. Tämä on tärkeä osa tätä toimintoa, koska ilman alaviivaa esikääntäjä tulkitsee tiedoston erilliseksi tyylitiedostoksi joten syntyy kokonaan uusi tyylitiedosto. Alaviiva kertoo että tiedosto on vain osa tulevaa CSS-tiedostoa, joten siitä ei luoda omaa CSS-tiedostoa vaan se otetaan osana suurempaa kokonaisuutta.

Yksi suosituimmista ja parhaimmista toiminnoista, mitä SASS:n kaltainen ympäristö tarjoaa on muuttujat. Kuten normaalissa ohjelmoinnissa voidaan monessa kohdassa toistuvaa tietoa tallentaa muuttujiin, jolloin tiedon käyttö on vaivattomampaa. SASS mahdollistaa samankaltaisen toiminnallisuuden. Kuviossa 15 huomataan että ensimmäinen tiedosto, joka importataan screen.scss tiedostoon on nimeltään variables. Kyseinen tiedosto sisältää kaikki projektissa tarvittavat muuttujat, joista sitten arvo haluttuun määritykseen kun sitä tarvitaan. Jos vielä tarkistellaan aikaisempaa kuviota, ensimmäinen import-lause hakee tiedoston nimeltään _variables.scss. SASS mahdollistaa tallentaa CSS:ssä toistuvia arvoja muuttujiin, jotka tässä projektissa on kirjoitettu kaikki tähän tiedostoon. Sivustolla on tietty pääväri, joka toistuu useassa elementissä sivustolla. Tämä on helppo tallentaa omaan muuttujaan jolloin se on helppo ottaa käyttöön tarvittuihin elementteihin. Muuttujiin kannattaakin tallentaa sellaisia arvoja jotka toistuvat useasti sivustoilla väriarvo, fontti koko, väri tai ihan mitä tahansa.

```
3
4  $brand-color: #4c8930;
5
```

KUVIO. 14 SASS-muuttuja

Muuttujan määritys on hyvin yksinkertaista, kääntäjä tietää kyseessä olevan muuttuja kun sen ensimmäinen merkki on $. Tämän jälkeen seuraa muuttujan nimi ja viimeisenä arvo. Muuttujan tallennettava arvo voi olla mitä vain, tähän ei ole asetettu rajoituksia. Muuttujia ei ole pakko määrittää omaan tiedostoon vaan niitä voidaan alustaa missä tahansa kohtaa koodia, ja mikäli useita samannimisiä muuttujia löytyy aina viimeisin määre jää voimaan. Ajatellaan tilanne, sivustolla toistuu useassa kohtaa kahdenkymmenen pikselin marginaali. Joka tallennetaan muuttujaan. Tämä sama marginaali pysyy samana, mutta sivustolta löytyy yksi sisältö kohta jossa kun näytön leveys on 450 pikseliä marginaalin tulisi olla nolla. Tällaisessa tilanteessa voidaan alustaa muuttujan arvo uudestaa media queryn sisällä. Arvon muutos ei vaikuta muuhun koodin, koska muutos tapahtuu media queryn sisällä ja valitsijalla määritetään mihin elementtiin muuttujan uusi arvo menee.

```
21 nav[role="main"] {
22   padding: 0;
23   background-color: #fafafa;
24 }
25
26 /* line 7, ../sass/elements/_nav.scss */
27 nav[role="main"] #main-nav {
28   padding: 0;
29 }
30 /* line 8, ../sass/elements/_nav.scss */
31 nav[role="main"] .row {
32   border-bottom: 1px solid #e5e5e5;
33   padding: 15px 10px;
34   margin-left: 0px;
35   margin-right: 0px;
36 }
37 /* line 14, ../sass/elements/_nav.scss */
38 nav[role="main"] .row.current {
39   border-left: 5px solid #4c8930;
40 }
41 /* line 15, ../sass/elements/_nav.scss */
42 nav[role="main"] .row .subOpen {
43   display: none;
44 }
45 /* line 16, ../sass/elements/_nav.scss */
46 nav[role="main"] .col-md-12 ul li a {
47   border-bottom: none;
48 }
49 /* line 17, ../sass/elements/_nav.scss */
50 nav[role="main"] li.level2.current {
51   font-weight: 900;
52 }
```

KUVIO 15. Tyypillinen CSS-toisto

Tyypillisessä CSS koodissa tulee kirjoitettua paljon toistoa. Tässä tapauksessa sivustolla on nav-elementti, jonka sisälle lukuisia lapsi elementtejä. Tyypillisesti kaikki CSS nav-elementille kirjoitetaan yllä olevan kuvion mukaisesti, koska se hahmottaa HTML-koodin rakennetta. Kuitenkin tässä projektissa kirjoitettiin SASS-koodia, joten yllä olevan kuvan mukainen on SASS-kääntäjän tekemää.

```
nav[role="main"] {
    padding: 0;
    background-color: #fafafa;
}

nav[role="main"] {
    #main-nav { padding: 0; }
    .row {
        border-bottom: 1px solid $line-color;
        padding: 15px 10px;
        margin-left: 0px;
        margin-right: 0px;
    }
    .row.current { border-left: 5px solid $brand-color; }
    .row .subOpen { display: none; }
    .col-md-12 ul li a { border-bottom: none; }
    li.level2.current { font-weight: 900; }
}
```

KUVIO 16. SASS nesting

Aikaisemman kuvion CSS tulostuu tämän kuvion SASS-koodin perusteella. Ensimmäinen huomio on rivi määrässä, CSS-koodia on kirjoitettu navigaatiolle kolmenkymmentäyksi riviä, kun taas SASS suorittaa saman kahdeksallatoista rivillä. Koodilla määritetään elementin valitsija ja sille annetaan avain normaalisti aaltosulut, mutta sulkujen sisään ei kirjoitetta pelkkää CSS-määreitä vaan uusia valitsijoita ja niiden määreet.

Tyypillisesti CSS-ohjelmoinnissa tulee paljon samojen määreiden toistoa, sekä erityisesti prefix toistoa. Prefix:t eivät kuuluu CSS-standardeihin, mutta joidenkin määreiden kohdalla niitä on pakko käyttää jotta selain tuntee ne. Vanhentunut esimerkki näistä on määre border-radius, tälle ei enää tarvitse antaa prefix:jä, mutta se toimii hyvänä esimerkkinä. Jokaisella selaimella on oma prefix, joten jos selaimena on Internet Explorer ja halutaan varmistua että border-radius toimii. Tulisi määreen eteen kirjoittaa -ms-. Tämän kaltaisen toiston vähentämiseen SASS tarjoaa niin kutsuttua mixin toimintoa. Hyvänä esimerkkinä, voidaan ajatella että mixin toimii samoin kuin ohjelmointikielissä puhutut setteri funktiot.

```
@mixin border-radius($param) {
    border-radius: $param;
    -moz-border-radius: $param;
    -ms-border-radius: $param;
    -o-border-radius: $param;
    -webkit-border-radius: $param;
}
```

KUVIO 17. SASS mixin määritys

Kaikki tämänkaltaiset toiminnot voidaan määrittää omaan tiedostoon ja ottaa samoin käyttöön kuin muuttuja esimerkissä. Määritys alkaa @mixin, joka spesifioi kyseessä olevan mixin toiminto. Jos ajatellaan ohjelmointi setteriä, vaikka JavaScript-ohjelmointikielessä, @mixin vastaisi JavaScript funktion määrettä, jota kummassakin seuraa vapaasti määriteltävä nimi. Tässä tapauksessa border-radius ja sitten sulut jonka sisään voidaan määrittää parametrit.

```
.box {

    @include border-radius(10px);


}
```

KUVIO 18. Mixin käyttö

Include lause kutsuu mixin:ä nimellä ja sulkuihin määritetään arvo, jonka toiminto asettaa elementille. Tässä tapauksessa elementti box saisi kaikille kulmille kymmenen pikselin pyörityksen.

On myös mahdollista jakaa elementtien ominaisuuksia toistensa kesken. Jos tarkoitus on luoda laatikoita, jotka ovat muuten identtisiä mutta omaavat jokainen erilaisen taustavärin.

```
.box {
    font-family: sans-serif;
    color: white;
    margin-right: 20px;
    padding: 10px;
}

.red {

    @extend .box
    background: red;


}

.blue {

    @extend .box;
    background: blue;

}

.green {

    @extend .box;
    background: green;
}
```

KUVIO 19. SASS-perintä

Kuviossa ensimmäisenä määritetään elementti box, joka saa kaikki yhteiset arvot. Sen jälkeen tulee elementit nimeltään blue, red ja green joista jokaisessa ensimmäinen määre on @include. Mitä tämä määre tekee on se perii suoraan siinä määritetyn elementin arvot. Tässä määritetty elementti on box ja sitten sille lisätään omat määreet, jotka voivat lisätä alkuperäiseen elementtiin jotain, tai ylikirjoittaa alkuperäisestä elementistä jotain.

Viimeisenä toiminto, minkä SASS:stä voisi mainita on kuinka Bootstrap-sovelluskehys on ottanut sen käyttöön. Tämä nopeuttaa projektien kehittämistä, sillä voidaan avata sovelluskehyksen oma muuttujat-tiedosto ja muokata se vastaamaan kyseisen projektin määreitä. Bootstrap on tässä vain mainintana, mutta monet muutkin sovelluskehykset tarjoavat perinteisten tyylitiedostojen lisäksi mahdollisuutta ladata SASS-tiedostot. Tämä on kuitenkin täysin kiinni järjestö kehyksen taustalla.

2.3 JavaScript

Ennen tätä kappaletta on puhuttu HTML- ja CSS-kielistä, kuitenkin nämä kaksi eivät ole ohjelmointikieliä vaan merkistökieliä. Merkistökielen tarkoitus on esittää grafiikkaa käyttäjälle, ohjelmointikielen tarkoitus on luoda ja hallinnoida ohjelman logiikkaa. JavaScript on Netscapella 1990-luvulla kehitetty ohjelmointikieli, jonka tarkoituksena on luoda toiminnallisuutta sivustoille. Sen jälkeen JavaScript on kehittynyt valtavan laajuiseksi kieleksi sekä yhdeksi maailman käytetyimmäksi ohjelmointikieleksi. Tiobe listaa, JavaScriptin sijalle yhdeksän ja se on noussut sijalta kymmenen vuoden aikana (TIOBE 2015, viitattu 10.10.2015). Tulevaisuuden kannalta voidaan odottaa JavaScriptin nousevan korkeammalle koko ajan. JavaScriptiä ei käytetä vain nettisivujen kanssa, sitä käytetään myös peliohjelmointiin, sovellusohjelmointiin sekä sen uusin aluevaltaus on tietokantaohjelmointi. Tähän aluevaltaukseen perustuen voidaan odottaa JavaScriptin nousua, sillä tähän asti PHP on ollut kieli, jolla hallinnoidaan tietokantoja.

Nettisivut eivät voi enää tänä päivänä luottaa ulkoasuun. Ulkoasun lisäksi tarvitaan toiminnallisuutta. Tällä viitataan että sivusto reagoi käyttäjän klikkauksiin, sekä yleisesti hiiren paikaan. Toiminnallisuutta voidaan toteuttaa myös CSS3:lla, mutta se ei ole luotettavin ratkaisu selain yhteensopivuuden takia, JavaScriptin kanssa selain yhteensopivuus ei ole ongelma. Yksinkertainen ja yleinen esimerkki toiminnallisuudesta on kuvagalleria.

2.3.1 DOM

DOM (Document Object Model) ohjelmointirajapinta, jota HTML käyttää. Sen avulla määritetään HTML:n rakenne ja se tarjoaa mahdollisuuden JavaScriptin kaltaisille kielille vaikuttaa sen toimintaan. JavaScript ohjelmoinnissa haetaan DOM tapahtumia ja niillä vaikutetaan sivuston toiminnallisuuteen.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body onload="window.alert('Hello World!');">


    </body>
</html>
```

KUVIO 20. Hello world

Kommunikaatio JavaScriptin ja HTML:n välillä tapahtuu tapahtuma-attribuuttien avulla. Esimerkin yksinkertaisuuden takia window.alert -kirjoitettiin suoraan onload:in sisään. Tavallisesti olisi kirjoitettu attribuutin sisään funktion nimi, joka olisi ajettu sitten omassa JavaScript-tiedostossa. Muita tapahtuma attribuutteja on:

- onclick, ajetaan kun kohdetta klikataan
- onmousedown, kun hiiren painike on painettu alas kohteen päällä
- onmouseup, kun hiiren painike on noussut ylös kohteen päällä
- onmouseenter, kun hiiren osoitin on kohteen päällä

Yllä on kuvattu vain osaa hiiri tapahtumista, näppäimistölle on omia tapahtumia ja näytölle on omia joista esimerkkinä on esitelty jo onload. Tapahtumia on valtava määrä, mutta ne kaikki toimivat samalla tavalla. Omien funktioiden määritys on yksinkertaista, kuten alla on esitetty. Hakasulkujen sisään voidaan kirjoittaa niin yksinkertaista, kuin monimutkaista koodia, jota sitten kutsutaan kun sille määritetty tapahtuma tapahtuu.

```
function tamaOnFucktionNimi() {
    //Ohjelmakoodi kirjoitettaan tänne
}
```

KUVIO 21. Funktiomääritys

2.3.2 Muuttujat

Kaikissa ohjelmointikielissä tarvitaan muuttujia. Niiden tarkoitus on varastoida data myöhempää käsittelyä varten. Joissain kielissä muuttujilla on käytössä tietotyyppejä, joiden tarkoitus on määrittää muuttujalla, minkä tyyppistä tietoa siihen voidaan varastoida. Verrattuna Java ohjelmointikieleen, jokaiselle muuttujalle määritetään tietotyyppi ja yleensä vielä näkyvyysmääre. Kuitenkin JavaScriptissä ei tällaisia määreitä käytetä.

```
int foo = 1;

var foo = 1;
```

KUVIO 22. Java muuttuja verrattuna Javascript muuttuja

Ylemmällä rivillä on tapa jolla määritetään numero tyyppinen muuttuja Javassa, ja sen alapuolella on tapa jolla alustetaan muuttuja JavaScriptissä. Sana var alustaa JavaScript muuttujan, mutta ei erottele mitä tyyppiä se edustaa. Sen jälkeen sekä Javassa, että JavaScriptissä tulee muuttujan nimi, minkä tarkoitus on kuvata jollain tasolla mitä muuttujaan varastoidaan, mutta on kuitenkin ihan vapaasti nimettävissä. Yhtäsuuruus merkki osoittaa, mikä arvo muuttujaan menee ja sen jälkeen varastoitava tieto. Merkittävä ero Javan ja JavaScriptin muuttujien välillä on, kuinka JavaScript käsittelee kaikkia numeropohjaisia muuttujia desimaaleina. Javassa on erikseen kokonaisluku muuttujat ja desimaali.

2.3.3 jQuery

Tähän asti on esitelty JavaScript-ohjelmointikieltä, mutta tämä on hieman ristiriitainen alue. Osa kehittäjistä pitää JavaScript-ohjelmointia heikkona vaihtoehtona ja osa pitää sitä parempana. Se osa, joka ei pidä JavaScript-ohjelmoinnista vetoaa kuinka HTML-koodin sekaan on kirjoitettava attribuutit, mikä lisää sekavuutta sillä kehittäjä joutuu liikkumaan vähintään kahden tiedoston välillä. Sekä koodin syntaksia pidetään rumana ja erittäin työläänä kirjoittaa.

Tähän on kehitetty vaihtoehto, nimeltään jQuery. Kyseessä ei ole lisäosa, mutta ei myöskään oma ympäristö. JQuery on JavaScriptillä rakennettu kirjasto, joka mahdollistaa koodin selkeämmän kirjoittamisen. Kuitenkin on hyvä huomauttaa että suurin osa JavaScript-ohjelmoijista aloittaa

ohjelmoinnin JavaScriptillä ja siirtyy sen jälkeen opettelemaan jQuery:n. Tämän jälkeen vasta tapahtuu valinta näiden kahden välillä.

JavaScript vaatii useita rivejä koodia toteuttaakseen toimintoja. Esimerkkinä sovellus, jonka halutaan vaihtavan taustaväriä kun sivu on ladattu, kirjoitetaan seuraavasti JavaScriptillä.

KUVIO 23. Javascript-sovellus, sivuston taustaväri vaihtuu kun se selain on ladannut sen

Ensin määritetään body-elementille onload-tapahtuma, joka kutsuu funktiota vaihdaVari. Tämän funktion sisään tulee koodi, joka muuttaa sivun taustavärin punaiseksi. Näinkin yksinkertaisen toiminnon luominen puhtaalla JavaScriptillä vaatii neljä riviä koodia, tapahtuman määritys HTML-tiedostoon, sekä funktio ja koodi sen sisään. JavaScript toteutuksessa ei ole mitään väärä ja on olemassa kehittäjiä jotka suosivat JavaScriptin kirjoittamista. JQuery on kuitenkin vaihtoehto, jolla voidaan toteuttaa kyseinen esimerkki yhdellä rivillä koodia, ilman tapahtuma attribuuttia HTML-koodissa.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>

        <h1>Taustaväri vaihtuu onload tapahtuman yhteydessä</h1>

        <script src="js/jquery-1.11.3.min.js"></script>
        <script src="js/main.js"></script>
    </body>
</html>
```

```javascript
$(function() {
    $('body').css('background', 'red');
});
```

KUVIO 24. Aiemmin kuvatun esimerkin toteutus jQuery:llä

HTML-tiedosto näyttää aivan samalta kuin aikaisemmassa esimerkissä, paitsi body-elementistä on poistettu onload-tapahtuma ja ennen main-tiedoston mukaan ottoa on otettu käyttöön jQuery-tiedosto. Tämä on pakko ottaa jokaiselle sivulle käyttöön jolla jQuery:ä tarvitaan, koska kyseessä ei ole oma ohjelmointikieli vaan JavaScript kirjasto. Koodi, joka vaihtaa taustavärin suoritetaan yhdellä rivillä, huomattavasti selkeämmällä syntaksilla. Ohjelma toimisi oikein vaikka koodista poistettaisiin ensimmäinen ja viimeinen rivi, mutta nämä rivit määrittävät onload-tapahumaa. Idea onkin jQuery:ssä, että tapahtumien kuuntelu ei tapahdu HTML-tiedostossa. Kirjaston sisään on kirjoitettu tapahtumat, jotka voidaan ajaa käyttämällä CSS-valitsija.

Yleisesti jQuery:a suositaan JavaScriptin sijaan juuri siitä syystä että koodin määrä vähenee huomattavasti, mikä on tärkeä piirre etenkin suurissa projekteissa.

3 LISÄOSAT JA SOVELLUSKEHYKSET

Kun kehittäjä on ollut tekemässä useampaa nettisivu projektia pienistä aina suuriin, huomaa hän että tietyt toiminnot toistuvat useilla sivustoilla. Toisin sanoen, hän joutuu tekemään samoja asioita, pienillä muutoksilla useaan kertaan. Samojen asioiden tekeminen jokaiseen projektiin uudestaan ja uudestaan ei ole tehokasta työskentelyä.

Tehokkuuden parantamiseksi kehittäjällä on mahdollisuus etsiä netistä valmiin lisäosan tai sovelluskehyksen, joilla näitä toistuvia piirteitä voidaan tehdä tehokkaasti. Yleisiä toimintoja, jotka toistuvat useilla sivuilla:

- Responssivisuus
- Kuvakarusellit
- Mobiilivalikko poikkeaa normaalista valikosta

Lisäosa on yleensä yksi yksittäinen tyylitiedosto ja JavaScript-tiedosto, tai yksi JavaScript tai tyylitiedosto. Näihin lisäosa-tiedostoihin kuuluu yksi toiminto mitä sivustokehittäjät tarvitset toistuvasti sivustoillaan, vastoin että he toistuvat tekevät samat toiminnot uudestaan ja uudestaan he voivat etsiä internetistä valmiiksi kehitetyn lisäosan jolla toteutetaan haluttu toiminto. Otetaan esimerkiksi toiminto, jossa sivulta löytyy tietty kuva. Tätä kuvaa klikkaamalla kuva aukeaa muun sivun sisällön päälle, tummentaa taustaa ja mikäli muita tämän kaltaisen toiminnallisuuden omaavia kuvia löytyy mahdollistaa siirtymisen seuraavaan kuvaan yhdellä klikkauksella. Tämän kaltainen toiminnallisuus vaatii oman JavaScript-tiedoston, joka hallinnoi kuvan avaamisen ja siirtymisen kuvien välillä. Sekä oma tyylitiedoston jolla saadaan toiminnon ulkoasu näyttämään miellyttävältä. Tässä projektissa tällaista toimintoa tarvittiin ja lisäosa otettiin käyttöön, magnific-popup. Lisäosien käytetään paljon sivusto projekteissa. Kehittäjä lataa lisäosan lähdekoodin, tässä tapauksessa tyylitiedoston ja JavaScript-tiedoston. Sitten hän yksinkertaisesti ottaa nämä tiedostot käyttöön. Viimeinen askel on alustaa lisäosa, joka tapahtuu kehittäjän omassa JavaScript-tiedoston.

```
$('.reference-item__images').magnificPopup({
    delegate: 'a',
    type: 'image',
    tLoading: 'Loading image #%curr%...',
    mainClass: 'mfp-img-mobile',
    gallery: {
        enabled: true,
        navigateByImgClick: true,
        preload: [0, 1]
    },
    image: {
        tError: '<a href="%url%">The image #%curr%</a> could not be loaded.',
        titleSrc: function(item) {

            var title = $(item.el).find('img').attr('title'),
            description = $(item.el).find('img').attr('alt');

            if (title || description) {
                return title + '<small>' + description + '</small>';
            }
            else {
                return '';
            }

        }
    }
});
```

KUVIO 25. Magnific-popup -lisäosan käyttöönotto projektissa

Kuviossa esitetään kuinka projektissa kutsutaan lisäosaa omasta tiedosto. Tämä lisäosa toimii yksinkertaisuudessaan rivin yksi määrityksen, sekä rivillä kolme olevan määrityksen yhdistelmällä. Muut parametri määritykset tuovat ylimääräistä toiminnallisuutta. Jokainen määritys kutsuu lisäosan lähdekoodin osia, antaen täten kuvalle uusia ominaisuuksia.

Jos lisäosa käsittää yhden tietyn omaisuuden, mikä on sitten sovelluskehys. Nämä kehykset ovat aina huomattavasti laajempia kokonaisuuksia, verrattuna lisäosaan. Tässä projektissa lisäosien lisäksi otettiin käyttöön Bootstrap-niminen sovelluskehys. Toisin kuin lisäosa, joka sisältää vain yhden tyylitiedoston, JavaScript-tiedosto, tai jossain tapauksissa yhdistelmän kumpaakin. Sovelluskehys on aina yhdistelmä kumpaakin ja huomattavasti laajempi. Bootstrap-kehys sisältää lukuisia toimintoa rakennettuna sen sisälle. Pelkästään tämän kehyksen avulla olisi voitu tehdä lisäosa esimerkin mukainen toiminto ja sen lisäksi koko sivuston layout, kuvakaruselli, alasvetovalikot ja paljon muuta.

Käyttöönotto toimii samoin, kuin lisäosan kanssa tyyli -ja JavaScript-tiedostojen käyttöönoton avulla. Kehittäjän on kuitenkin hyvä suunnitella sovelluskehyksen käyttöönotto ennakkoon, sillä kyseiset tiedostot ovat varsin laajoja ja jos sivulle vielä niiden lisäksi otetaan käyttöön muita lisäosia ja omat tyylit, voi selaimen suorituskyky olla kovilla. Alussa kannattaakin istua alas ja miettiä,

mitä toiminto sivulle tulee, sen jälkeen pohtia miten toteutus tapahtuu. Tässä projektissa tehty toiminto, kuvan aukeaminen muun sivun päälle, olisi voitu tehdä Bootstrap-kehyksellakin. Kuitenkin lopputuote sisältää oman lisäosan jolla tehtiin tämä toiminto joten ei ole mitään järkeä sisällyttää tämän kaltaista toimintoa Bootstrap:in. Hieno puoli näissä laajoissa kehyksissä onkin, että ne antavat kehittäjä valita mitä ominaisuuksia kehittäjä haluaa ottaa mukaan.

4 TOTEUTUS

4.1 Paikalliskehitys

Ennen kuin projektin varsinainen työ voitiin aloittaa, täytyi tehdä muutamia valmisteluja. Ensimmäisenä pystytettiin versionhallinta. Tässä projektissa versionhallinnasta vastaa git-niminen järjestelmä. Git:n lisäksi tarvittiin, jokin palvelu johon projektin versiot tallennettiin. Github on tämän kaltaisista palveluista kuuluisin ja käytetyin maailmanlaajuisesti, kuitenkin Github:a ei käytetty tässä projektissa. Bitbucket on palvelu jota käytettiin. Kaksi isointa syytä Bitbucketin valintaan olivat:

- Bitbucket on aikaisemmista projekteista tuttu
- Bitbucket sallii yksityisten repositorioiden tekemisen ilmaiseksi

Palvelusta on olemassa myös maksullinen versio, jossa tehtyjä repositorioita voi jakaa äärettömän määrän henkilöiden kanssa. Ilmaisessa versioissa yksittäisen repositorion voi jakaa maksimissaan viiden henkilön kanssa. Projektia työsti vain yksi henkilö, joten maksullisen version tuoma etu oli tällä saralla turha. Tärkein syy miksi Bitbucket:a suosittiin tässä projektissa oli yksityiset repositoriot. Tämä tarkoittaa sitä, että palveluun tallennetut tiedostot näkyvät vain niille henkilöille joille projektijohtaja on sallinut.

Bitbucket:lle on saatavilla erillinen sovellus, nimeltään SourceTree, jolla voidaan tehdä versionhallinalle luonnolliset toiminnot. Sovelluksen käyttö on kuitenkin vapaaehtoista, koska git:ä voidaan ajaa tietokoneen komentotulkissa.

Toinen askel, mikä otettiin ennen varsinaista projektin aloitusta, oli SASS:in alustus. Alustus on yksinkertainen kolmen askeleen teko. Ensimmäisenä SASS tarvitsee toimiakseen Ruby:n joka siis täytyy asentaa koneelle, toisena asennetaan itse SASS mikä tuo mukanaan kaikki tulkin ominaisuudet ja viimeisenä tarvitaan oma sovellus joka osaa tulkita SASS syntaksia ja kääntää sen CSS:ksi. Sovelluksia on useita tarjolla, osa ilmaisia, osa maksullisia ja osasta on tarjolla maksullinen ja maksuton versio. Sovellusta jota käytettiin on nimeltään Compass. Sovellus toimii suoraan komentotulkin päällä. Muun muassa sovellus Koala toimii omassa käyttöliittymässään.

Näiden kahden valmistelun askeleen jälkeen varsinainen työ alkoi. Sivuston ensimmäinen versio tehtiin paikallisena ohjelmointin. Versionhallinta ja SASS olivat siis paikalliskehitystä varten, julkaisujärjestelmällä on oma versionhallinta tätä varten. Koska järjestelmä rakentaa sivun omalla

tyylillään palastelemalla sen sivupohjasta ja moduuleista joihin sisältö tulee, otettiin tämä huomioon paikalliskehityksessä ja rakennetaan vain etusivu. Tällöin toimivasta etusivusta voitiin siirtää koodin osat oikeisiin paikkoihin.

Paikalliskehityksen tueksi otettiin käyttöön Bootstrap-sovelluskehys. Tästä ladattiin koko projektin ajaksi kehitysversio, joka sisältää kaikki ominaisuudet mitä siihen kuuluu. Vasta kun projekti on valmis, voitiin karsia kaikki turhat osat pois. Turhien sovelluskehyksen osat on syytä aina poistaa lopputuotteesta. Poistamalle ne parannetaan sivuston optimaalisuutta. Piirre mikä on tärkeä ajatellen mobiililaitteita

4.2 Järjestelmäkehitys

Paikalliskehityksen aikana projektin määrityksiin oli pariin otteeseen tullut muutoksia. Paikalliskehityksen aikana saatua lopputuotetta ei voitu käyttää järjestelmäkehityksen tueksi. Ajan säästämiseksi sivustosta ei aloitettu uutta paikallisesti kehitettävää projektia, vaan sen tekeminen tehtiin suoraan järjestelmään komponentti kerrallaan. Esimerkiksi sivuston navigaatio rakennettiin paikallisesti, testattiin ja kun se lopulta toimi niin kuin haluttiin se siirrettiin järjestelmään. Tämän jälkeen samalla periaatteella rakennettiin sivuston sisältöosa ja eri elementit mitä sivustolle tehtiin.

Sivuviidakkoon rakentaminen alkaa niin kutsutun sivupohjan tekemisestä, sivupohjaa voidaan verrata WordPress teemaan. Sivupohjia voidaan tehdä useita, tässä projektissa sivupohjia tehtiin neljä. Etusivulle tehtiin oma, oletussivupohja jota käytetään kaikilla muilla sivuston sivuilla. Kaksi viimeistä pohjaa luotiin kirjautumissivulle ja viimeinen pohja on kehitys sivunsivupohja, jota käytetään hallinnoimaan sivun moduuli pohjia ja JSON-tiedostoja. Sivupohjia voidaan ajatella sivuston luurankona, sinne laitetaan vain sen verran HTML-koodia, että sivuston pohja tulee kasaan. Loppu tarvittava koodi kirjoitetaan moduulien sisään, joka kannattaa kirjoittaa siten että ne eivät ole riippuvaisia minne on laitettu, antaen käyttäjälle mahdollisuuden siirrellä mielensä mukaan elementtien paikkoja.

Sivuston rakentamisen kannalta oltiin vasta puolessa välissä, kun sivuston HTML- ja CSS -sekä JavaScript-koodi oli kirjoitettu ja siirretty järjestelmään. Tämän pisteen jälkeen oli aika ruveta siirtyä rakentamaan sivustolle uutiset ja tuotteet. Sivuviidakolla on rakennettu sisään mahdollisuus tehdä erinäköisiä listoja, kuten uutis -, tapahtuma -, kontakti -tai tuotelistoja. Erot näiden listojen välillä on pienet, tässä projektissa otettiin käyttöön uutislistat sekä tuotelistat. Listat toimi-

vat samankaltaisesti kuin WordPress:ssä tehtävät uutiset. Niille täytyy antaa kategoria, jonka mukaan määritellään mille sivulle sisältö tulee. Sivuviidakko antaa kuitenkin mahdollisuuden antaa kategorioille ominaisuuksia. Uutisille annettiin tekstin lisäksi kuva. Tuotelista on kuitenkin paljon monimuotoisempi tässä projektissa. Tuotelistan tarkoituksena on listata tehtyjä projekteja, jonka tiedot syötetään järjestelmään ja listataan sivustolle. Listauksen taustalle rakennettiin hieman monimutkaisempi kategoria rakenne, jotta sille voitiin antaa kaikki tarvittavat ominaisuudet. Nämä tallennettavat tuotteet piti pystyä liittämään yrityksen omiin tuotteisiin, eli jos oli tehty Sivuviidakkoon liittyvä projekti, piti tuotelistauksen elementti pystyä liittämään siihen tuotteeseen. Listaussivulle tuli sitten hakulomake, jonka avulla voidaan hakea esimerkiksi Sivuviidakkoon liittyvät projektit. Mikäli jo tallennettua tuotetta pitää muokata myöhemmin, järjestelmän piti osata täyttää tuotteen tallennuslomake niillä tiedoilla automaattisesti, mitä sille oli jo annettu.

Tämän kaltaiset toiminnot vaativat alakategorioiden luontia. Tuote kategoria johon tallennettavat tuotteet tallennettiin, on nimeltään reference tai viite. Tähän viite kategoriaan oli luotava kolme ala kategoriaa, jokainen nimettiin tuotteiden mukaan.

Lomakkeessa käyttäjän on valittava kolmesta painikkeesta yksi, jolla lisättävä tuote liitetään oikeaan alakategoriaan. Valitun tuotteen mukaan, johon tallennus liitetään käyttäjää saa lomakkeeseen uuden kentän täytettäväksi. Moniarvoinen kenttä johon voi antaa useita arvoja. Käyttäjä pystyy valitsemaan alasvetovalikosta tuotteen ominaisuuden mikä viitteeseen halutaan lisätä, sekä antamaan sille omin sanoin kuvauksen. Nämä ominaisuudet on määritetty omaan tuotekategoriaansa.

Poikkeuksellisesti viite tuotteen yhteydessä pyydettiin tallentamaan projektin asiakas, tämäkin tieto olisi voitu tallentaa myös omaan alakategoriaan. Mutta tällaista toteutusta ei tehty vaan luotiin kehityssivulle tyhjä lista, josta luotiin JSON lista. Lomakkeessa idea on, että kenttä on normaali tekstikenttä joka osaa ennakoida asiakkaan nimen jos se löytää sen JSON-tiedostosta. Jos nimeä ei löydy asiakasta ei voida tallentaa, ennen kuin kentän viereisestä painikkeesta painaa. Bootstrap-modaali lomake aukeaa, johon käyttäjä voi lisätä uuden asiakkaan. Tarkoitus tässä toiminnossa oli varmistaa että sivuston käyttäjät eivät voi vaikuttaa tämän listan sisältöön, toisin sanoen he eivät voi poistaa asiakkaita siitä. Taustalla toimii yksinkertainen ajax haku.

Viimeinen askel kehitystyössä oli testaus, mikä oli kaksiosainen tehtävä. Ensin testattiin sivuston toiminnallisuus ja korjattiin virheet. Kaksi isomman luokan virhettä löytyi, joiden korjaaminen ei ollut niinkään työlästä, kuin vaikeaa löytää syy missä ohjelma menee solmuun. Ensimmäinen virhe liittyi tallennetun tuotteen muokkaukseen. Kun käyttäjä siirtyi tuote sivulta muokkaamaan

olemassa olevaa tuotetta, ei järjestelmä osannut löytää kyseisen tuotteen tietoja. Vika löytyi lomakkeen lähdekoodista, tallentaminen tapahtui oikein mutta tiedon hakemiseen tarvittavaa tietoa ei linkittynyt oikein järjestelmään taustaan. Toinen vaikeampi virhe korjata oli tuotesivulla oleva listaus. Listaus oli täyttynyt järjestelmään aina oikein, mutta tiedon haku ei toiminut oikein. Vika lopulta paljastui olevan hakulauseen ehdossa. Toinen testauksen vaihe oli selaintestaus. Aikaisemmin mainittu käyttäjäkunta on niin pieni ja käyttäjien laite taso on tunnettu, joten vanhemmat selainversiot eivät aiheuttaneet korjaustoimenpiteitä. Katsottiin ainoastaan että sivusto toimii oikein jokaisen selaimen uusimmassa versiossa. Laitetestausta sivustolle ei suoritettu, koska sivustolla on rajattu responsiivisuus.

4.3 Ongelmat

Ensimmäiseksi haasteeksi nousi projektin paikalliskehityksen aikana versionhallinnan opettelu. Git oli entuudestaan tuttu, kuten myös BitBucket. Haasteeksi nousi kuitenkin itse git:n opettelu, sillä ennen oli turvauduttu erillisen sovelluksen käyttöön. Tämä sovellus on kuitenkin raskaskäyttöinen, joten versionhallintaa varten opeteltiin ajamaan git-komentoja tietokoneen komentotulkissa. Aluksi tämäkin tuntui raskaalta vaihtoehdolta, sillä komennot täytyy osata ulkoa ja kirjoittaa oikein. Komennot, joita tarvittiin projektin aikana:

- git init
- git add remote origin <bitbucket repositorion osoite tähän>
- git add <tiedosto, jota muokattu ja halutaan siirtää versionhallintaa tähän>
- git commit –m ”Viesti versionhallintaa varten”
- git push origin master
- git pull
- git clone <bitbucket repositorion osoite tähän>

Versionhallintajärjestelmässä on lukuisia muitakin komentoja, joita opeteltiin projektin ulkopuolella mutta niitä ei tarvittu tässä projektissa. Pääosin syystä, koska projektin läpivienti tapahtui yksin.
Sivuston suunnittelu toi mukanaan omia ongelmia. Eikä pelkästään viitata sivun ulkoasun suunnitteluun, vaan enemmänkin miten sivustolla oleva tieto tallennetaan, mihin tallennetaan ja mistä se haetaan. Ulkoasun suunnittelua ei tehty erikseen, tämän kaltaisissa projekteissa on tyypillistä että sivustosta toimitaan leiskat kehittäjälle, jonka pohjalta sivustoa rakennetaan. Leiska on yleensä PhotoShop ohjelmalla piirretty tarkka kuva kaikista sivuista, sekä niiden toiminnoista. Tässä projektissa ei sille kuitenkaan varattu aikaa. Tarkempaa suunnittelua kuitenkin vaati sivus-

ton tuoterakenteen luominen, sekä asiakastietojen tallentaminen. Viitteiden yhteydessä käytetään näitä kahta tietoa, mutta ne tallennetaan eri paikkoihin. Viite itsessään tallennetaan suoraan julkaisujärjestelmän taakse. Asiakastiedot, tässä tapauksessa, vain nimi tallennetaan kuitenkin JSON-muotoon omaan tiedostoon. Alussa asiakkaatkin tallennettiin ihan samaan muotoon kuin koko viite, mutta sen jälkeen nousi kysymys onko siinä mitään järkeä? Kun asiakkaan nimi meni järjestelmän taakse, tarkoittaa se kuka vain jolla on ylläpito-oikeudet sivustoon voi selata sitä listaa ja mahdollisesti vahingossa poistaa yksittäisen nimen. Tämä johtaisi datan hukkumiseen ja mahdollisiin sekaannuksiin, joten jälkikäteen kehitettiin tälle oma viritelmä. Tieto tallennetaan edelleen järjestelmän taakse, mutta ylläpitäjät eivät löydä listaa elleivät he osaa sitä etsiä oikeasta paikasta, sekä tiedon poistaminen on erehdyksessä nyt mahdotonta.

5 POHDINTA

Sivuston rakentaminen oli mukavaa vaihtelua normaaleihin asiakasprojekteihin verrattuna. Kun projektin tilaa entuudestaan tuntematon asiakas, on sille annettava tietynlainen takuu. Normaalisti sivuston on toimittava oikein uusimmissa laitteissa ja selaimissa, unohtamatta vanhempia laitteita sekä niiden käyttöjärjestelmiä. Suurimmaksi ongelmaksi näissä projekteissa muodostaa Internet Explorer sen heikon HTML5 ja CSS3 tuen takia. Toisena ei voida tietää minkälaisia käyttäjiä sivulla on, mikä vahvistaa vain syytä tehdä sivusto toimivaksi mahdollisimman monessa selaimessa. Talon sisäinen projekti, kuten tämä, oli erittäin mukavaa vaihtelua tehdä sillä käyttäjäkunta on huomattavasti pienempi ja mikä parempaa käyttäjien laitteet ovat tunnettuja. Toisin sanoen sivusto voitiin rakentaa käyttämällä CSS3-tekniikoita, joita ei muuten ole voitu käyttää. Erityisesti sivuston ulkoasu oli mukava tehdä käyttämällä CSS3 flexbox tyyliä.

Viimeiseksi mietteeksi nousee, aina projektin valmistuttua. Oliko projektilla merkitystä ja kuinka siitä selviydyttiin? Voisi ajatella että projektin elinkaaren aikana tapahtuneet muutokset määrityksiin tekisivät tästä vaikean ja huonosti toteutetun projektin. Todellisuudessa tämän projektin läpivienti on ollut varsin normaalia, verrattuna muihin projekteihin, missä on oltu mukana. Määritykset muuttuvat, joskus suunnitellaan huonoja ideoita jotka vasta kehityspöydällä huomataan. Tällaiset tilanteet ovat yleisiä ja vaativat vain kykyä mukautua tilanteeseen. Mitä sitten tulee mietteeseen, oliko tällä työllä merkitystä. Tässä vaiheessa vaikea kommentoida asiaa, riippuu täysin kuinka sivustoa aletaan käyttämään, mutta jos tämä projektin avulla valmistuminen koulusta muuttuu realiteetiksi voidaan vastata. Oli merkitystä.

LÄHTEET

CSS – tricks. 2.4.2015. BEM 101 | CSS – tricks. Viitattu 2.8.2015. https://css-tricks.com/bem-101/

CSS – Tricks. 9.7.2008. The Difference Between with ID and Class. Viitattu 15.5.2015. https://css-tricks.com/the-difference-between-id-and-class/

CSS Neuse. 2015. The History of CSS. Viitattu 15.5.2015. http://www.cssneuse.net/the-history-of-css.php

Infoworld. 18.9.2013. Bitbucket vs. GitHub: Which project host has the most?. Viitattu 29.5.2015. http://www.infoworld.com/article/2611771/application-development/application-development-bitbucket-vs-github-which-project-host-has-the-most.html?page=2

Linux wiki. 9.5.2014. Git – Linux. Viitattu 23.5.2015. http://www.linux.fi/wiki/Git

Magnific popup. 2015. Magnific Popup: Responsive jQuery Lightbox Plugin. Viitattu 4.10.2013 http://dimsemenov.com/plugins/magnific-popup/

MDN. 17.4.2015. Introduction. Viitattu 8.8.2015. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

MDN. 7.4.2015. HTML attribute reference - HTML (HyperText Markup Language) | MDN. Viitattu 2.5.2015. https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes

Ohjelmointi. 2015. Mitä on ohjelmointi. Viitattu 1.8.2015. http://ohjelmointi.medianurkka.com/?p=59

Ross Shannon. 21.8.2012. The History of HTML | From the HTML 1.0 spec to XHTML 1.0.... Viitattu 2.5.2015. http://www.yourhtmlsource.com/starthere/historyofhtml.html

Ross Shannon. 21.8.2012. What is HTML? | HyperText Markup Language explained. Viitattu 2.5.2015. http://www.yourhtmlsource.com/starthere/whatishtml.html

SASS. 26.6.2015. SASS_REFERENCE. Viitattu 9.9.2015. http://sass-lang.com/documentation/file.SASS_REFERENCE.html#cache_stores

TIOBE. 2015. TIOBE Software: Tiobe Index. Viitattu 8.8.2015. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Treehouse Blog. 2.6.2014. The 2014 Guide to Responsive Web Design. Viitattu 22.4.2015, http://blog.teamtreehouse.com/modern-field-guide-responsive-web-design

Tutorialspoint. 2015. JavaScript Variables. Viitattu 8.9.2015. http://www.tutorialspoint.com/javascript/javascript_variables.htm

Twitter Bootstrap. 2015. Bootstrap · The world's most popular mobile-first and responsive front-end framework. Viitattu 4.10.2015. http://www.getbootstrap.com/

W3C. 13.3.2014. The HTML head element - Web Education Community. Viitattu 2.5.2015. https://www.w3.org/community/webed/wiki/The_HTML_head_element#Head.3F_What_head_are_we_talking_about.3F

W3C. 23.3.2015. HTML 5.1 Nightly. Viitattu 22.4.2015, http://www.w3.org/html/wg/drafts/html/master/

W3C. 23.5.2001. Introduction to CSS3. Viitattu 22.4.2015, http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

W3C. 27.6.2012. A Short History of JavaScript. Viitattu 22.4.2015, https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

W3C. 9.12.2014. HTML5 Differences from HTML4. Viitattu 3.5.2015. http://www.w3.org/TR/html5-diff/

W3Schools. 2015. HTML DOM Events. Viitattu 8.8.2015. http://www.w3schools.com/jsref/dom_obj_event.asp