Haaga-Helia
ammattikorkeakoulu Oy

Responsiivinen toteutus staattiseen web-projektiin

Teemu Eronen

Opinnäytetyö
Tietojenkäsittelyn ko.
2015

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

15.11.2015

Tekijä
Teemu Eronen

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
Responsiivinen toteutus staattiseen html-projektiin

Sivu- ja liitesivumäärä
63 + 5

Tämän toimeksiannon tarkoitus on luoda responsiivinen toiminnallisuus Haaga-Helian tietojenkäsittelyn tulevalle ohjelmistokehityksen peruskurssille. Sivun tuli asettua näytölle sopivaksi riippumatta päätelaitteesta ja sen asettamista rajoituksista. Sivuston tulee toimia myös käyttäjäystävällisesti ja moitteettomasti.

Haaga-Helian toimeksiantajalta saatiin pohjaksi staattinen verkkosivu, joka käännettiin XSLT-muunnoksella XML-tiedostoista. Tälläinen sivuratkaisu mahdollistaa helpon datan suodattamisen ja tietojen julkaisun sivulle. Samalla se mahdollistaa uuden kurssisivun luomisen noudattamalla samaa XML-dokumentin rakennetta.

Toimeksianto aloitettiin 2015 kesällä verkkosivun kehittämisellä ja samalla kerättiin aineistoa dokumentointia varten. Syksyllä aloitettiin kirjoittaminen ja tehtiin viimeiset muutokset yhteistyönä toimeksiantajan kanssa. Raportti tuli valmiiksi marraskuussa 2015.

Sivun sisällön asettelemiseen käytettiin apuna Bootstrap sovelluskehystä, jolla toteutettiin responsiivinen ruudukkosysteemi, modaalit ja osa tyyleistä käyttämällä Bootstrapin valmiita komponentteja, kuten painikkeita. Muita projektiin sisällytettyjä kirjastoja ovat Prism.js syntaksien korostamiseen ja Reveal.js kosketusnäytöille optimoitujen dia-esitysten tekemiseen.

Käytettyjä tekniikoita toimeksiannossa ovat responsiivinen suunnittelu, joka sisältää mediakyselyt, juoksevan ruudukon ja joustavat kuvat. Viewport meta-tag määritys, joka liittyy verkkosivun lataamiseen ruudulle oikeassa koossaan – ei alku suurentelua. Ohjelmointikieliin kuuluvat XML, XSLT, HTML5, CSS3, sekä Javascript ja Jquery.

Ongelmia tuli sivun latausaikojen kanssa. Sivu asetettiin esirakennus vaiheeseen tarkoittaen, että kaikki tiedostot käännetään palvelimella valmiiksi ja MacigXML-kirjasto tiputettiin pois. Upotetut videot ja Jsbin-editori siirrettiin aukeamaan omiin välilehtiin, koska Embed.js lataaminen resursseille lähteestä kesti 20 sekunttia. Koska Bootstrapin suosio on niin suuri, sillä tehdyt sivut saattaa näyttää samalta kuin yleisimmät sivustot ja hukkua näin massaan. Bootstrap on kuitenkin erinomainen kehys lähteä kehittämään responsiivisia verkkosivuja, tai ottamaan mallia sen ratkaisuista.

Työ paljastaa, että responsiivisuus on yleistyvä tapa tuottaa verkkosivuille lisä-arvoa kun ei tarvitse erikseen tuottaa mobiililaitteille omia verkkosivuja. Helpompi kohdistaa sivujen asetuksia tietyille selaimen leveyksille, jolloin käyttäjän laitteella ei ole muuta väliä kuin, että sillä pääsee verkkoon. Responsiivisen suunnittelun kanssa kannattaa noudattaa mobile first tapaa, jolloin vain tärkeimmät ominaisuudet tulevat mobiililaitteille ja säästytään yhteensopivuusongelmilta, joita saattaa tulla kun skaalataan ominaisuuksia pienemmäksi.

Asiasanat
XML, XSLT, HTML, CSS, Bootstrap, Responsive

Haaga-Helia
University of Applied Sciences

Abstract

15.11.2015

Author
Teemu Eronen

Degree programme
Business Information Technology

Report/thesis title	**Number of pages and appendix pages**
Responsive implementation for html project	**63 + 5**

The purpose of this thesis was to create a responsive frame for an existing static website project. This means that the webpage has to adapt to any screen, regardless of the using device and its boundaries. The page must also work fluidly and be extra user friendly.

Haaga-Helia's representative has made a static website with XSLT-transformation from XML-files. The page translates the transformed XML to HTML using MacigXML-library. This kind of page formatting makes data-filtering easy and publishing possible on a larger scale. It also facilitates creating of a new course-site with the same functionalities and appearance just by following the same XML-data structure.

The project was launched in late May 2015 with developing the website and collecting source data for this document. Writing of this document and final changes to the webpage started in September and the whole project was finished in November 2015.

The used techniques chapter presents the techniques used on a universal level. Course-site refactoring gives a detailed view on how the techniques were used in the assignment.

Bootstrap framework was used to help position the site content with its built in responsive grid system. Modal windows and partly styles were also made using Bootstrap's own components, like buttons. The additional libraries used are Prism.js for syntax highlighting and Reveal.js for touchscreen optimized slideshows.

Among used techniques there are responsive site design (including media queries, fluid grid and flexible images). Also, the viewport meta-tag was used to load the site to a mobile screen at its correct size – no pre-zooming. The programming languages that were used are XML, XSLT, HTML5, CSS3, Javascript and Jquery.

Problems occurred when there were too many HTTP-calls to the server, due to the high amount of exercise files to be converted. The page load took too long. Pre build phase was established, meaning all the files were converted at server-side and MagicXML was dropped out. All pictures and libraries were also compressed. The embedded videos and Jsbin-editor were moved from modals to independent subpages. The reason being that embed.js library took up to 20 seconds to load all embedded recourses. Bootstraps high amount of users makes it overused and therefore almost every page made with it, looks too much alike. Bootstrap's components are sometimes a bit tricky to modify due to its SASS preconverted CSS classes.

The study reveals that responsive web design is increasingly utilized to get more value to any website as it eliminates the need for independent mobile websites. It is easier to target settings to the specific browser width, then the user's device is irrelevant. Responsive design goes hand in hand with mobile first strategy, then only the most critical functionalities come to mobile and developers avoid major compability issues when scaling the site for desktop users.

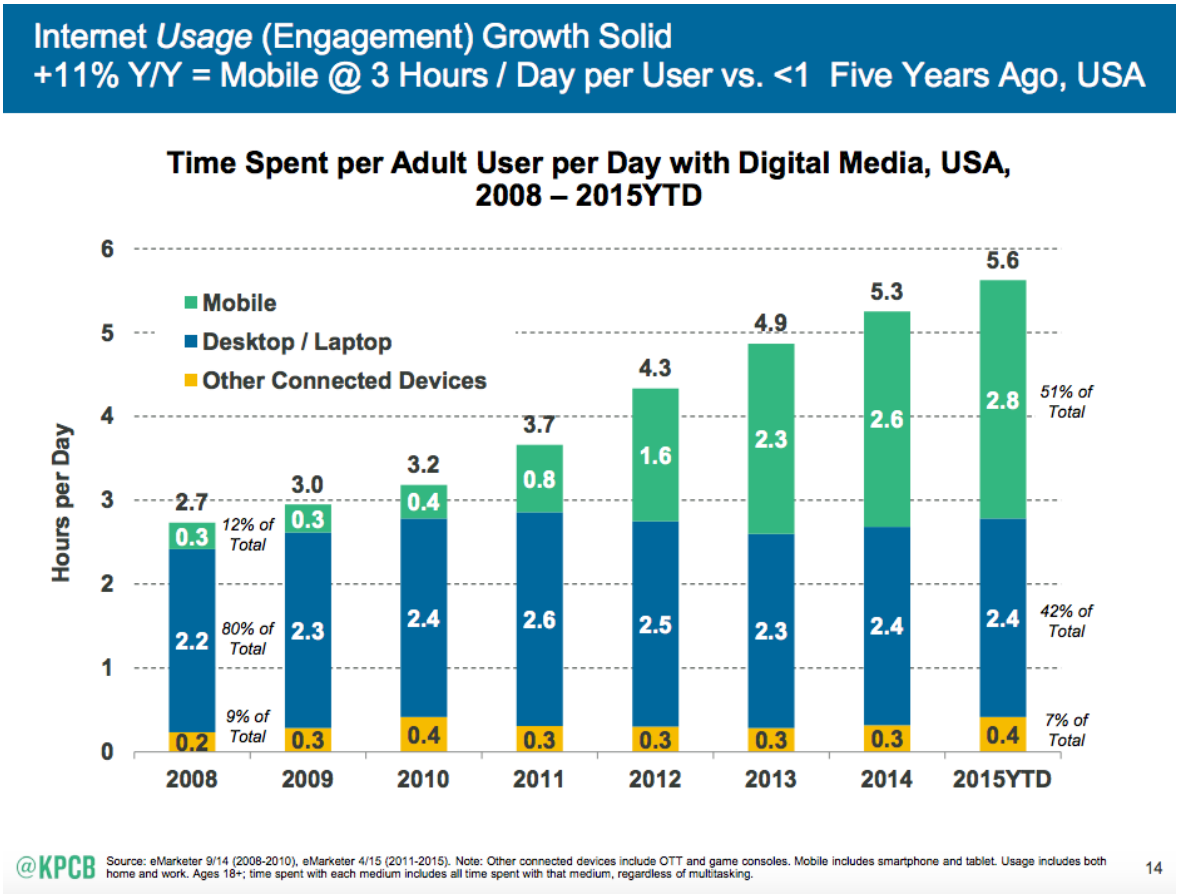
Keywords
XML, XSLT, HTML, CSS, Bootstrap, Responsive

Sisällys

1 Johdanto 1
1.1 Projektin tausta 2
1.2 Projektin tavoite ja rajaus 2
1.3 Käsitteet 3
2 Toteuttamismenetelmä 6
2.1 Responsiivinen suunnittelu 6
2.2 HTML5 10
2.3 Viewport- meta tag 13
2.4 CSS3 14
2.5 Javascript & Jquery 16
2.6 XML & XSLT 20
2.7 Bootstrap 24
3 Sivuston toteutus 27
3.1 Käytetyt työvälineet 27
3.2 Sivuston rakenne 33
3.2.1 XML & XSLT 33
3.2.2 Magic XML 37
3.3 Responsiivisuuden toteuttaminen 38
3.3.1 Media kyselyt 38
3.3.2 Juokseva ruudukko 39
3.3.3 Kuvat ja elementtien piiloittaminen 40
3.3.4 Responsive video 44
3.3.5 Fontit 44
3.4 Graafiset ohjauselementit 45
3.4.1 Modal-ikkunat 45
3.4.2 Painikkeet 49
3.4.3 Navigointi 52
3.4.4 Accordionit 55
4 Johtopäätökset 58
4.1 Kohdatut haasteet 59
4.2 Tulosten hyödyntäminen 62
4.3 Jatkokehitysideat 63
Lähteet 64
Liitteet 66

1 Johdanto

Verkkosivujen mobiiliselailu kasvaa vuosi vuodelta ja mobiiliselailun määrä onkin jo ylittänyt tietokoneella tehtävän selailun digitaalisissa medioissa, mikä käy ilmi Smartinsights-sivun taulukoista verkossa käytettyyn aikaan ja määrään. Keskimääräisesti Amerikassa vuonna 2015 aikuinen viettää noin 5.6 tuntia päivästään digitaalisen median parissa. Tästä luvusta 51 prosenttia (2.8 tuntia) käyttää mobiililaitetta selailuun, kun vuonna 2008 sama luku oli vain 12 prosenttia (0.3 tuntia). Tietokonekäyttäjien tuhlaama aika on pysynyt kutakuinkin samana 2.2- 2.6 tunnin välillä. Mutta koska ihmisten tuhlaama aika digitaalisen median kanssa on myös kasvanut, niin tietokoneiden kokonaiskäyttäjämäärät digitaalisen median parissa ovat pudonneet 80 prosentista 42 prosenttiin. (Bosomworth 2015).



Kuva 1: Smartinsight - Internetin käyttö digitaalisessa mediassa 2015

Verkkosivuja on kehitetty ennen joko puhelimille ja tableteille, tai pöytäkoneille ja kannettaville tietokoneille. Ongelmaksi tällaisessa toteutustavassa muodostuu päivittämisen määrä kun halutaan lisätä ominaisuuksia sivulle. Lisäksi onko järkevää tehdä Iphonelle omaa sivua, sen jälkeen pöytäkoneelle ja sitten Samsung Galaxy S6:lle, tai vastaavalle.

Verkkopalvelun kehittämisessä kannattaa harkita kahden version tekemistä vain jos pöytäkoneversiossa on monimutkaisia ominaisuuksia, joita mobiiliin ei haluta. Tähän ongelmaan on keksitty responsiivinen suunnittelu.

Responsiivisessa suunnittelussa käytetään jonkun laitteen fyysisiä ominaisuuksia määrittämään sivuston leveydet ja näytettävät ominaisuudet. Tämä saavutetaan käyttämällä sekoitusta joustavia ruudukoita, joustavia kuvia ja älykkäitä media kyselyitä . Eli siis kun sivua suurentaa tai pienentää, sivun ulkoasu skaalautuu sopivammaksi ruudulle ja piiloittaa niitä ominaisuuksia mitä halutaan käyttää vain tietynkokoisilla laitteilla. Tällä menetelmällä tehdyt sivut kestää ajan hampaita paremmin, kun ei määritetä mitään tarkkoja raameja sivulle. Tulevaisuudessa tulee olemaan sellaisia resoluutioita, mitä emme osaa ennustaa. Eikö olisikin hienoa rakentaa verkkosivu, joka osaisi varautua valmiiksi melkein kaikkiin resoluutioihin? Sisältö vain asettuisi hyvin ruudulle riippumatta sen koosta. (Knight 2010.)

1.1 Projektin tausta

Opinnäytetyössäni tein Haaga-Helian toimeksiantajan tekemään kotisivuprojektiin lisäyksiä. Vaatimuksena sivuille oli asetettu se, että sivu on käyttöystävällinen eri selaimilla ja sivun tulee toimia yleisimmin käytetyillä päätelaitteilla ja niiden resoluutioilla.

Sivun pohjana luovutettiin XML dataa josta muotoillaan XSLT muunnoksella HTML-dokumentti. Tähän lisätään responsiivisuutta ja toiminnallisuutta käyttämällä Bootstrap-sovelluskehystä, CSS-tyylitiedostoa ja sivun interaktiivisuutta lisätään käyttämällä Javascriptiä ja sen kirjastoa Jquerya.

Sivulle tuotiin myös kaksi kirjastoa helpottamaan presentaatiota. Reveal.js-kirjasto kosketusnäyttö optimoitujen diaesitysten näyttämiseen ja Prism.js, joka tekee koodin näyttämiseen sivuille omat korostetut laatikot.

Projektin tarkoitus on saada HH:n kurssin kotisivusta käyttöystävällisempi opiskelijoita varten, käyttävät he sitten mobiili-, tai pöytäkoneen selainta.

1.2 Projektin tavoite ja rajaus

Projektin tavoitteena oli tehdä olemassa olevasta Haaga-Helian kurssisivusta responsiivinen ja varmistaa selainyhteensopivuus. Sivun tulee skaalautua eri näyttökooilla sopivaksi ruudulle sopivaan tilaan.

Projektia kehitettiin yhteistyössä toimeksiantajan kanssa. Toimeksiantaja oli vastuussa lopullisesta tuotoksesta. Toimeksiantajan kanssa pidettiin yhteyttä sähköpostilla ja järjestettiin tapaamisia tarpeen mukaan. Toimeksiantajan sana painoi paljon lopullisessa työssä hänen laajemman ohjelmointitaustansa takia. Toimeksiantaja tuotti kaiken sisältö materiaalin projektiin ja teki muutoksia taustakoodiin tarpeen mukaan. Oma panokseni keskittyi enemmän käyttöliittymäpuoleen, mutta tein muutoksia myös taustakoodiin, sillä HTML-palikat muodostettiin jo palvelimen päässä.

Sivuston tuli olla käytettävä eri päätelaitteilla: tabletti, pc, puhelin. Käytettävyyttä pyrittiin parantamaan lisäämällä projektiin staattinen navigaatiopalkki isommille ruuduille. Tavoitteena oli myös varmistaa selainyhteen sopivuus suosituimmilla verkkoselaimilla. Selaimiin pyrittiin saada tuki vanhemmille versioille tai ilmoitustyyppinen varoitus liian vanhasta versiosta.

Oppimistavoitteena opinnäytetyön tekijälle on HTML-, CSS-kielen ja Javascript-kirjaston hallinta. Syventää tietämystä XSLT- tyylitiedostojen käytöstä XML- tiedostojen kanssa. Vahvistaa tuntemusta responsiivisesta sivun suunnittelusta, testauksesta ja käytöstä yhdessä Twitter Bootstrap kehyksen kanssa. Parantaa tuntemusta eri selainten eroista. Tarkastaa apukirjastojen käytön selainten väliselle tuelle. Lisäksi oppimistavoitteena on parantaa tiedonkeräämis-, sekä kirjoittamistaitoa.

Projektin tuloksena syntyvän sivun käyttäjät ovat Haaga-Helian opiskelijat ja opettajat.

1.3 Käsitteet

CSS:	(Cascading Style Sheet): Tyylitiedosto, jolla voidaan asettaa tyylejä dokumenteille. Käytetään pääosin HTML-kielen kanssa.
Javascript:	Ohjelmointikieli, jolla ohjelmoidaan interaktiivisuutta web-ohjelmiin ja HTML-dokumentteihin
Jquery:	Javascriptin kirjasto, joka yksinkertaistaa javascriptin käytön. Tarvitaan vähemmän koodirivejä saman asian kirjoittamiseen kun javascriptillä.

HTML:	(Hypertext Markup Language)HTML:lä kirjoitetaan web-sivujen rakenteita ja sisältöä. HTML koostuu elementeistä joihin voidaan lisätä attribuutteja ja tekstiä. HTML:llä voidaan myös välittää metatietoja.
XML:	(Extensible Markup Language) on w3:n kehittämä datan merkkauskieli. Se kehitettiin helpottamaan staattisen sisällön verkkojulkaisua suurella skaalalla. Siinä ei ole ennaltamääritettyjä elementtejä, vaan kaikki määritykset tulee tehdä itse.
XSLT:	(Extensible Stylesheet Language Transformations) on w3:n mukaan suositeltu tyylitiedosto XML:ä varten. Sillä voidaan muuttaa, piiloittaa tai poistaa elementtejä ja attribuutteja suoraan ulostulevasta tiedostosta.
Mobile first:	On nimensä mukaan mobiili ensin ajattelutapa. Siinä lähdetään kehittämään sivuja ensin mobiililaitteille, jonka jälkeen lisätään ominaisuuksia pala kerrallaan isommille ruuduille. Näin saadaan tärkeimmät ja välttämättömimmät ominaisuudet varmasti toimimaan mobiileilla ja voidaan varmistua, ettei ongelmia tule yhteensopivuuden kanssa.
Bootstrap:	Bootstrap on erittäin suosittu sovelluskehys nopeaan sivujen luontiin. Se tarjoaa pohjan kehitykselle lukuisine komponentteineen(painikkeet, navigoinnit) ja toiminnallisuuksineen(responsiivinen ruudukko, modaalit).
Media kysely:	Mediakysely on tapa kohdistaa tyyliasetuksia eri resoluutioille. Media kyselyt otetaan käyttöön kun ruudun tai selaimen koko ylittää tietyn pisteen resoluutiossa.
Breakpoint:	Tässä dokumentissa breakpointilla viitataan mediakyselyissä olevaan ruudun leveyspisteeseen, minkä yli tai ali mennessä vaihdetaan asetuksia CSS:ssä.
Resoluutio:	Tarkoittaa näytön kokoa pikseleissä mitattuna. Resoluutio on muotoa leveys * korkeus. Esimerkiksi 1280x720.

Elementti: Yleinen datan merkkaustyyli. Elementeillä merkataan yleensä jonkin rakennetta. Jos elementin sisällä on toinen elementti, niin silloin siitä tulee lapsielementti, joka on isäntäelementin sisällä. Elementtejä käytetään mm. HTML:ssä, XML:ssä ja XSLT:ssä.

```
<element name=”parent”>
        <element name=”child”/>
</element>
```

2 Toteuttamismenetelmä

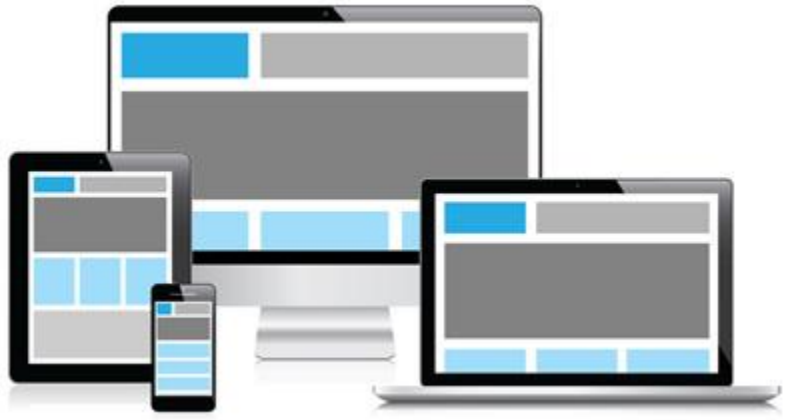
Tässä kappaleessa käydään läpi toimeksiantoon valittuja tekniikoita ja menetelmiä teoreettisella tasolla. Tekniikat käsittelevät enemmän käyttöliittymäpuolta kuin palvelinpuolta.

Projekti toteutettiin käyttämällä HTML5 ja CSS3 tekniikoita. Sivuston data on XML-tiedostossa, joka muotoillaan XSTL:n avulla ja lähetetään selaimelle. Projektissa käytetään apuna Bootstrap sovelluskehystä.

Sivuston toteutus osassa esitellään kuinka projektissa käytetään valittuja tekniikoita.

2.1 Responsiivinen suunnittelu

Ethan Marcotte esitteli vuonna 2010 responsiivisen verkkosivuston periaatteita netti artikkelissa A List Apart. Marcotte vertasi responsiivista sivusuunnittelua arkkitehtuuriin, jossa jonkin tilan muoto muovautuisi sen läpi kulkevien ihmisten mukaan. Käytännössä sivusta voidaan suunnitella sellainen, että sisältö olisi joustavaa ja skaalautuisi laitteen mukaan sopivaksi näytölle. Tämä mahdollistaa sen, että yksi sivu riittäisi kaikille laitteille.



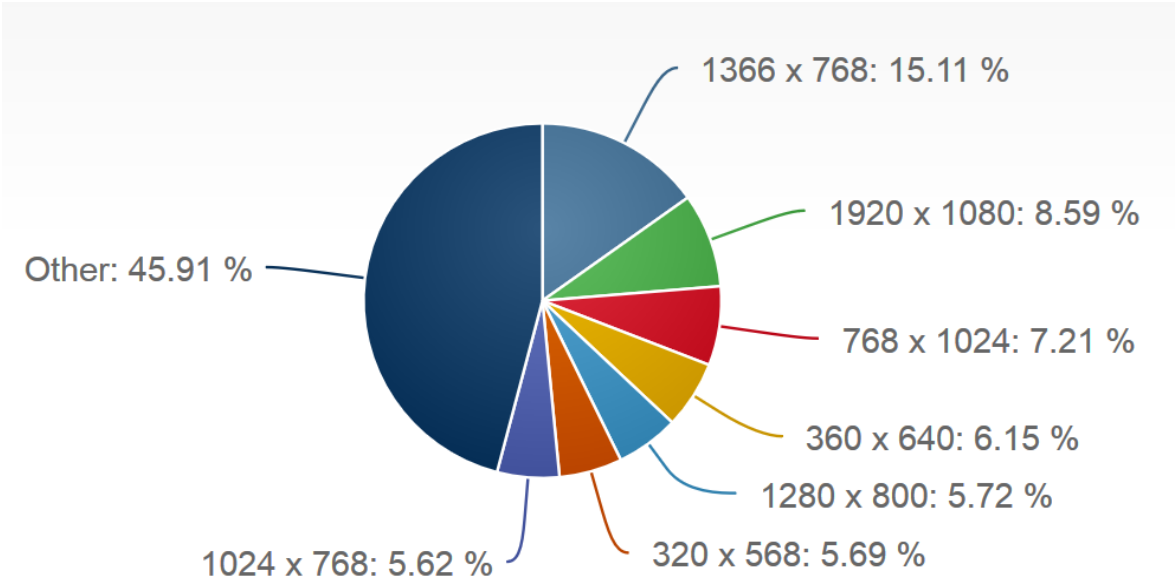
Kuva 2: Juokseva ruudukko esimerkki

Marcotte jakaa responsiivisen suunnittelun kolmeen osaan. Juoksevaan ruudukkoon (fluid-grid), mediakyselyihin (media queries) ja joustaviin kuviin (flexible images).
Sisällön leveydelle ei tulisi määrittää tarkkaa pikseliarvoa vaan leveys ilmaistaisiin suhteellisilla arvoilla, jolloin sisältö suhteutetaan sitä ympäröivään elementtiin.
Usein verkko suunnittelijat lähtevät liikenteeseen täysikokoisesta työpöytäversiosta suunnitellessaan käyttöliittymää, jättäen mobiiliversion toissijaiseksi päämääräksi. Tällöin kehit-

täjät taipuvat käyttämään kaikkea mitä heillä on kehittäjä arsenaalissaan. Sivulle tulee paljon hyödyllisiä ominaisuuksia ja visuaalista karkkia, mutta sitten huomataan, ettei mikään skaalaudu hyvin pienille ruuduille. (Code my views 2015.)

Mobile first on kasvava trendi käyttöliittymäsuunnittelussa, jossa sivu suunnitellaan ensin minimaaliset toiminnot sisältäväksi mobiilisivuksi. Sen jälkeen lisätään iteratiivisesti työpöytäversioon erikseen tarvittavia ominaisuuksia. Näin voidaan välttyä käyttöliittymän skaalautumis ongelmilta, kun on lähdetty pienimmästä liikenteeseen. Mobile first sopii responsiivisen suunnittelun kanssa käsi kädekkäin, koska responsiivinen suunnittelu perustuu pitkälti media kyselyihin. Niillä voidaan tähdentää tietty ruudunkoko, mihin asetukset liittyvät. Eli lisättäisiin ominaisuuksia osissa kokoajan isommille ruuduille.(code my views).

Media kyselyt loi W3C osana CSS3 määrityksiä. Media kyselyillä voidaan tutkia laitteen fyysisiä ominaisuuksia ja tarkastella tiettyjä laiteluokkia. Käytännössä mediakyselyillä annetaan tietty leveyspiste minkä ylittäessään tai alittaessaan vaihdetaan CSS asetuksia. Leveyksiä annettaessa tulisi ottaa huomioon kohderyhmät. Mitkä ovat leveyspisteet milloin asetukset vaihtuvat toisiin?



Kuva 3: Netmarketshare - resolutions based on global market share in 2015

Suositussa Bootstrap sovelluskehyksessä käytetään leveyspisteinä 768 pikseliä ja alle, yli 768 pikseliä, 992pikseliä tai yli ja 1200 pikseliä tai yli. Kuitenkin jos katsoo Netmarketsharen piirakkakaaviota suosituimmista resoluutioista markkinaosuuteen verrattuna, niin suurin yksittäinen käyttäjäryhmä on 1366x768 pikseliä, jossa 1366 pikseliä on suosituin leveys. Pienin suosittu leveys on 320 pikseliä. Bootstrapissä pienin piste on alle 768 pikselin leveyksille. Tämä saattaa aiheuttaa ongelmia pieninäyttöisille puhelimille, koska sivut ovat

optimoitu pienimmillään 768px leveille laitteille. Breakpointit tulisi asettaa mahdollisimman laajalti, jotta sivu tulee optimoitua mahdollisimman monille laitteille näkymään hyvin.

Media kyselyjen hyödyt tulee helpommin esiin tarkastelemalla tyylitiedoston liittämistä sivulle ladattavaksi vain jos tietyt ehdot täyttyvät:

```
<link rel="stylesheet" type="text/css"
  media="screen and (max-device-width: 480px)"
  href="example.css" />
```

Esimerkissä on kaksi osaa: media tyyppi on ruutu (screen) ja sulkujen sisällä oleva max-device-width: 480px määrittää puolestaan, että jos laitteen ruudun leveys on horisontaalisesti 480 pikseliä leveä tai alle, niin ladataan example.css tiedosto. Tällä tavalla voidaan kohdistaa tyyliasetuksia eri laitteille riippuen laitteen fyysisestä koosta. Voidaan vaikka piiloittaa puhelimilta ominaisuuksia, joita halutaan näyttää vain suuremmilla ruuduilla, tai tehdä linkkien ja nappuloiden painoaluetta suuremmaksi pienille näytöille. Kuitenkin media kyselyjä voi käyttää tehokkaimmin rustaamalla kaikki samaan css-tiedostoon (Knight 2011).

Mediakyselyitä voidaan tehdä css:ään seuraavasti:

```
@media only screen and (max-width : 480px){
Asetuksia 480px tai alle leveille näytöille...
}
```

Tai CSS tiedostoon voidaan liittää asetukset toisesta tiedostosta @import direktiivillä:

```
@import url("example.css") screen and (max-device-width: 480px);
```

Joustavaa ruudukkoa on hyvä tarkastella fonttien suhteellisten em-arvojen avulla. Kun asettaa fonteille ”font-size: 100%;”, silloin fonttien koko on suhteellinen selaimen fontin kokoon nähden, joka on yleensä 16px. Muille teksteille voi näin määrittää helposti suhteellisen koon laskukaavalla kohde/sisältö = tulos. Kun halutaan 12 pikselin kokoista tekstiä, niin suhteellinen koko tälle laskettaisiin 12px/16px=0.75, eli silloin teksti olisi 0.75 kertainen, tai paremmin 0.75em. Samaa periaatetta voidaan käyttää minkä tahansa sisällön leveyksien määrittämiseen, jolloin kaikki on suhteessa isäntä-elementteihin. Suhteellisia leveyksiä mitataan esimerkiksi navigointipalkille, joka halutaan 900 pikseliä leveäksi 1200

pikselin kokoiseen tilaan. Laskemalla 900/1200= 0,75 saadaan navigoinnille suhteellinen leveys ”width= 75%;” (Marcotte, 2009).

Elementeille voidaan myös asettaa leveydeksi ”width: 100%;” ja ”max-width:1200px”, jolloin elementti olisi aina 100% leveä suhteessa ympäröivän elementin leveyteen, mutta ei ylittäisi koskaan 1200 pikseliä.

Joustavien kuvien tekemiseen on monia eri keinoja, joista yksi on laskea kuville suhteelliset leveydet samoin tavoin kun Marcotte opasti laskemaan joustavan ruudukon elementtejä. Kuville voidaan antaa leveydet prosentteina jolloin kuvat pysyy ruudulla ja skaalautuu. Kuvat voidaan asettaa noin 1/3 niiden säiliön leveydestä jolloin kuvat asettuu nätisti kolmiriviin. (Marcotte 2010)

```
.image {
  float: left;
  margin: 0 3.31753545023696682% 1.5em 0;   /* 21px / 633px */
  width: 31.121642969984202211%;            /* 197px / 633px */
}
```

Mediakyselyillä voidaan lisätä kuvien järjestäytymistä eri resoluutioille tarpeen mukaan, ja asettaa kuvat alle 400px leveille näytöille esimerkiksi puoliksi, eli noin 50 prosenttia:

```
@media screen and (max-width: 400px) {
 .image{
    margin-right: 3.31753545023696682%;    /* 21px / 633px */
    width: 48.341232227488151658%;         /* 306px / 633px */
  }
```

Yksi tyyli on käyttää CSS:n max-width asetusta. Kun asettaa esimerkiksi: img { max-width: 100%; }. Tällöin kuva ladataan alkuperäisessä koossa, ja kun kuvan näyttöalue pienenee alle kuvan alkuperäisen koon, niin kuvakin pienenee suhteessa näyttöalueeseen. Kuville voi asettaa myös esimerkiksi:

```
img{
max-width: 450px;
width:100%;
}
```

Tällöin kuva ei ylittäisi 450px mutta olisi 100 prosenttia kuvan näyttöalueesta. Huono puoli max-width:n käyttämisessä on kuvien lataus alkuperäisessä muodossa, jolloin suuret kuvat voivat hidastaa sivun latautumista.(Knight 2011.)

Kuvia voidaan myös ladata ja piiloittaa mediakyselyjen avulla riippuen leveydestä. Esimerkissä ladataan iso kuva yli 500px leveydelle ja pieni kuva alle 500px.

```
@media (min-width: 500px) {
  .navbar-brand h3 {
    background: url(iconLarge.png); }
}
@media (max-width: 500px) {
  .navbar-brand h3 {
    background: url(iconSmall.png); }
}
```

Näin saataisiin optimoitua kuvien latausajat kun ladataan puhelimille pienet kuvat ja tietokoneille isommat kuvat.(LePaige 2014.)

2.2 HTML5

HTML (lyhenne sanoista hypertext markup language) on hypertekstin merkkauskieli jolla voidaan kuvata hyperlinkkien sisältämää tekstiä, eli hypertekstiä. Toisin sanoen HTML kielellä voidaan merkata tekstiä, kuvia, videoita ja muuta sisältöä käyttäen HTML sytntaksia, joka on erilaisia merkkejä ja sanoja, joita tietokoneet osaavat tulkita. HTML:ää käytetään myös sivun rakenteen kuvaamiseen. Sillä kuvataan esimerkiksi sivun header (sivun yläosa missä on yleensä navigointi), body (sivun keskiosa missä on varsinainen sisältö) ja footer (sivun alaosa missä on tyypillisesti yhteystiedot ja hyödyllisiä linkkejä). HTML on vastuussa siitä miten sisältö asetellaan sivulle, mutta ei määritä sitä miltä sivu näyttää. CSS (cascading style sheet) hoitaa tyylimäärittelyt HTML-dokumentille. Sivun interaktiivisuus, eli toiminnallisuus rakennetaan Javascriptilla. (West 2012.)

HTML on elävä standardi, jonka viimeisin versio HTML5 viittaakin uusimpiin moderneihin web-tekniikoihin. Näihin tekniikoihin kuuluvat rajapinta WebGL (3d selaingrafiikat), sovelluslisäosat GeoLocation Api (käyttäjän sijainnin selvittäminen) ja File Api (tiedoston lataaminen sovellukseen). HTML5:n omia uusia toimintoja ovat LocalStorage (antaa sovelluksen ladata tiedostoja käyttäjän selaimeen), HTML5 Video ja HTML5 Audio, joilla voidaan

liittää video- ja äänitiedostoja sivulle ilman erillisiä liitännäisiä kuten FLASH, joka on muutenkin tietoturvariskien takia otettu oletuksena pois päältä Firefoxista, Facebookista, Youtubesta ja muista suurista palveluista. (West 2012.)

Kehittäminen HTML:llä voi tapahtua millä tahansa tekstinkirjoitus ohjelmalla. Kehitys tapahtuu merkkaamalla tiedostoon elementtejä, jotka ovat verkkosivun rakennuskomponentteja. Elementit merkataan sarkaimien sisään <elementti></elementti>. Elementeillä on alku ja loppuelementit. Elementin loppumista merkataan kenoviivalla /. Elementtien sisään voidaan asettaa toinen elementti jolloin siitä tulee lapsielementti ja perii isäntäelementin ominaisuudet. Elementit voidaan myös merkitä loppumaan välittömästi jos niiden sisään ei tule lapsielementtejä. (West 2012.) Elementeille voidaan liittää myös attribuutteja:

```
<elementti atribuutti=”www.esimUrl.fi” />
```

Aluksi tehdään index.html, joka toimii etusivuna. Tekstitiedosto tallennetaan ”.html” loppupäätteellä, joka kertoo selaimelle tiedoston sisältävän HTML:ää. HTML sivu vaatii hyvin vähän merkkejä kehityksen aloittamiseksi. Perus pohja etusivulle voisi olla vaikka:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<meta name="author" content="Teemu Eronen"/>
<meta name="keywords" content="HTML,CSS,XML,JavaScript"/>
<title>HTML-pohja esimerkki</title>
</head>
<body>
            <p>Tervetuloa</p>

            <footer>
            Teemu Eronen <br/>
            +358 123 4567 <br/>
            Teemu.eronen@hotmail.fi
            </footer>
</body>
</html>
```

Joka näyttäisi selaimessa tältä:

Kuva 4: Peruspohja selaimessa

Dokumentti aloitetaan määrittelemällä DOCTYPE elementti. Selain katsoo dokumentin DOCTYPE määritystä kun dokumentti avataan. Selain tulkitsee määrityksestä käyttääkö dokumentti moderneja web-tekniikoita vai onko sivu suunniteltu käyttämään vanhoja selaimia. <!DOCTYPE html> viittaa että sivulla käytetään moderneja HTML5 tekniikoita.

Esimerkin rakenne koostuu <html>-, <head>- ja <body>-elementeistä. Elementit sisältävät metadataa, otsikon ja yhden p-elementin(paragraph). Html elementti on tiedoston juuri, jolloin kaikki HTML-koodi tulee laittaa elementin sisälle lukuunottamatta DOCTYPE-määritystä. Head-elementin sisään tulee kaikki sivun tieto mitä ei näytetä sivulla ollenkaan. Esimerkissä headin sisään on asetettu metadataa ja title-elementti. Title-elementti asettaa sivulle selaimen yläreunassa näytettävän nimen. Body-elementin sisään tulee kaikki sivulla näytettävä tieto. Footer-elementin sisälle tulee sivun alareunassa näytettävät tiedot. Header-elementillä asetettaisiin puolestaan yläreunassa näkyvät tiedot, kuten yleensä navigointi.

Metadata on tietoa datasta, jota ei näytetä sivulla ollenkaan mutta on koneellisesti luettavaa. Metadata antaa esimerkiksi tietoa selaimelle ja web-palveluille kuinka sivua käsitellään tai antaa hakukoneille avainsanoja, joilla sivu voidaan löytää paremmin. Esimerkissä asetettiin charset=”UTF-8”, joka kertoo sivun käyttävän utf-8 merkistö standardia. Name=”author” alustaa attribuutin julkaisija, jolla asetetaan sisällöksi tekijän nimi content=”Teemu Eronen”. Keywords on hakukoneiden käyttämä metadata, jolle kannattaa antaa parhaiten sivua kuvaavia sanoja. Tällöin sivu löytyy myös paremmin hakukoneilla. (w3schools.)

HTML-dokumentti kannattaa tarkastaa W3C:n validaattorilla osoitteesta https://validator.w3.org/. Sivulla voidaan syöttää koko koodi kopioimalla se validaattoriin tai lataamalla palveluun. Sivu sitten kertoo onko tekemäsi koodi W3C:n (World Wide Web Consortium) tai WHATWG:n (Web Hypertext Application Technology Working Group) standardien mukaista ja virheetöntä.

2.3 Viewport- meta tag

Mobiili selaimet käsittelevät verkkosivuja hieman erilailla kun työpöytäselaimet. Tämä johtuu siitä, koska mobiililaitteden ruutujen resoluutio vaihtelee niin paljon. Sellaisia sivuja, joita ei ole erikseen suunniteltu mobiiliselaimille, selataan puhelimella käytännössä niin, että sivu piirretään taustalle kokonaan ja pieni pala sivua näkyy puhelimen näytöllä suurennettuna tai sitten sivu näkyy vain puoliksi puhelimen ruudulla. (Ala-Äijälä, 2012.)

Koska puhelimien fyysiset koot vaihtelevat niin paljon ja mobiiliselaimetkin käyttävät hieman eri leveyksiä. Esimerkiksi sisällön asettaminen kiinteästi 480px ei auta, koska safarin oletusleveys on 980px Iphonella. Sivun reunalle jäisi vain tyhjää tilaa 500px ja käyttäjä joutuisi suurennella osia sivusta, joka ei taas ole kovin käyttäjäystävällistä. Tähän on olemassa <meta name=”viewport”>-tagi avuksi. (Ala-Äijälä, 2012.)

Viewport tarkoittaa selaimen näkymää. Viewport tagille voi antaa muutaman oletusarvon, jolla saadaan sivu latautumaan mobiiliselaimilla sen kokoisena kun selaimen leveys on. Viewportille kannattaa antaa arvoksi ”width=device-width” joka palauttaa laitteen fyysisen ruudunleveyden selaimelle ja asettaa sen viewportin leveydeksi. Näin sivu on aina ruudun kokoinen ladattaessa. Viewportille voi antaa myös kiinteän leveyden pikseleinä. (Ala-Äijälä, 2012.)

```
<meta name="viewport" content="width=device-width”/>
```

Jotkut selaimet pitävät sivun leveyden vakiona kun vaihdetaan puhelimella pystynäkymästä (portrait) vaakanäkymään (landscape). Tällöin selain mielummin suurentaa sisältöä ruudulle kuin asettelisi sisällön ruudulle uudestaan. Viewportin content arvolle voi antaa attribuutiksi ”initial-scale=1”, joka luo laitekohtaisten ja css kohtaisten leveyksien välille 1:1 suhteen sallien sisällön uudelleenasettamisen. Attribuutit tulee erottaa pilkulla, joka varmistaa vanhempien selainten lukevan tagin arvot oikein. Viewport tagille voidaan näiden lisäksi asettaa muita atribuutteja, jotka ovat:

- minimum-scale
- maximum-scale

- user-scalable

Näistä tulee ottaa huomioon, että ne voivat estää sivun suurentamisen käyttäjältä ja aiheuttaa ongelmia sivuston luettavuudesta. Sivujen tulisi olla mahdollisimman käytettäviä ja käyttäjän zoomauksen estäminen ei tue tätä periaatetta. (LePage, 2014.)

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

2.4 CSS3

CSS3 (Cascading Style Sheet) on W3C:n kehittämä elävä standardi, joka käytännössä tarkoittaa, että siihen kehitetään kokoajan uusia ominaisuuksia ja lisätään sitä mukaa kun ne ovat valmiita. CSS3 sanana voisikin tarkoittaa kaikkea mitä on kehitetty CSS2.1 version kaikkeen. CSS4 versiota ei tule koskaan, vaan jossain vaiheessa numero tippuu pois ja on vain CSS. CSS3 on menossa modulaariseen suuntaan ja kehitettävät moduulit merkitään tasoilla perustuen siihen, mones kehityskierros on menossa. Uusimpia moduuleja tasolla 4 olisi esimerkiksi selectorit. Koska moduuleja kehitetään jatkuvasti, niin niiden tuki selaimille voi vaihdella suuresti (Gasston 2011). Selaimien tarjoama tuki ominaisuuksille kannattaa tarkistaa esimerkiksi sivulta www.caniuse.com.

CSS3:n uudet ominaisuudet ovat mediakyselyt, joilla voidaan tarkentaa tyylejä eri laitteille riippuen selaimen resoluutiosta. Erittäin hyödyllisiä kun tehdään tyylejä erikokoisille päätteille. Esimerkki asetus voisi olla vaikka:

```
@media only screen and(max-width:768px){
Asetuksia alle 768px leveille selaimen näkymille…
}
```

Selectorit joilla voidaan kohdistaa määrityksiä elementtien attribuuttien avulla. CSS3 selectoreilla voidaan esimerkiksi valita kaikki linkit jotka alkaa ”http” ja lisätä niihin ikoni.

```
a[href^='http'] {
                background: url(“icon.png”) no-repeat left center;
                display: inline-block;
                padding-left: 20px;
}
```

Pseudo-luokat ja pseudo-elementit ovat selectoreja, jotka käsittelevät tietoa dokumenttipuun ulkopuolella samaan pahaan kun ”:hover” tai ”:active”. Pseudo-luokat käsittelevät

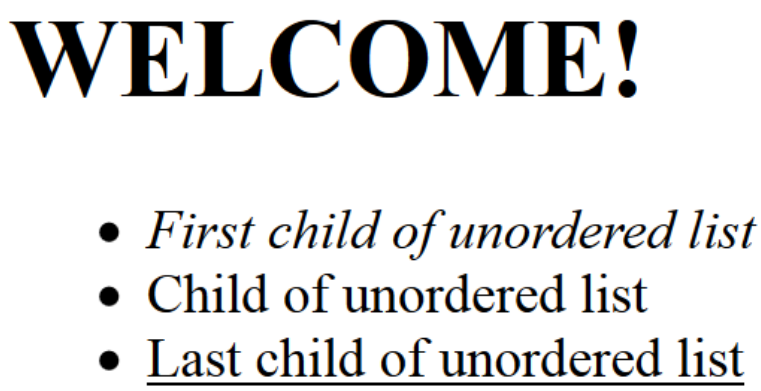
vain elementin tekstiosuutta samoin kuin pseudo-elementit. Pseudo luokilla voidaan esimerkiksi tehdä zebra-stripe tyylinen luettelo, jossa joka toinen rivi olisi harmaalla luettavuuden helpottamiseksi:

tbody tr:nth-of-type(even) { background-color: #DDD; }

Toinen hyvä esimerkki olisi asettaa jonkin listan ensimmäinen elementti isolla ja viimeinen italic tyylillä:

```
<style>
            ul :first-child{ font-style:italic;}
            ul :last-child{ text-decoration:underline;}
</style>
</head>
<body>
<div>
      <h1>WELCOME!</h1>
      <ul>
            <li>First child of unordered list</li>
            <li>Child of unordered list</li>
            <li>Last child of unordered list</li>
      </ul>
</div>
```

Näyttäytyy selaimelle näin:



Kuva 5: Esimerkki pseudo-elementtien käytöstä avattuna selaimessa

Web-fontit käsittää lähinnä @font-face directiivin millä voidaan määritellä fontti selaimelle mitä ei löydy käyttäjän koneelta. @Font-face direktiiville annetaan käytettävä fontti ja tiedoston sijainti. (Gasston 2011.)

@font-face {font-family: FontName; src: w local(' fontname '), x url('/path/ filename.otf ') y format('opentype');

Värejä annettiin ennen RGB (red, green, blue) muodossa. Väreihin lisättiin läpinäkyvyys määritys, jolloin annetaan värit muodossa RGBA(red, green, blue, alpha), eli voitaisiin antaa ”background-color: 255,255,255, 0,5;”, jolloin viimeinen numero määrittäisi 50 prosentin läpinäkyvyyden. (Gasston 2011.)

Tekstille annettiin muutamia mielenkiintoisia 3d ominaisuuksia ja koko määrtyksiä. 3d määritteitä ovat esimerkiksi text-shadow, jolla voidaan antaa tekstille varjostusta tehden siitä 3 ulotteisen näköisen. Muita tekstiominaisuuksia ovat määritykset ylimenevälle tekstille. Näitä voidaan määrittää CSS:ään text-overflow asetuksella. Text-overflow:lle voidaan antaa arvoksi ellipsis ja clip. Mielenkiintoisempi ellipsis lisää tutut kolme pistettä tekstin loppuun jos tila loppuu ja clip poikkasee tekstin siitä kohtaa kun se menee ylitse. Tekstin fontteihin esiteltiin rem-yksikkö (root em), joka on sama kuin em, paitsi suhteessa HTML-dokumentin fontin kokoon, kun em on suhteessa elementin fontin kokoon. (Gasston 2011.)

Responsiivisia kokoja voidaan asettaa uusilla selaimen näkymän mukaan vh (viewport-height), vw (viewport-width), vmax(viewport maximum-width) ja vmin(viewport minimun-width) arvoilla. Suhteelliset yksiköt mitataan prosenteissa, eli vw: 1; tarkoittaa prosenttia selain-näkymän leveydestä. Vmin ja vmax ovat selain-näkymän maksimiarvoja. Selaimissa on Firefoxia lukuunottamatta joitain rajoituksia kyseisten arvojen käyttämiseen(caniuse;). Ennen käyttämistä tulisi tarkistaa miten selain tukee kyseisiä arvoja. (Gasston 2011.)

Muita CSS3 mukana tulleita ominaisuuksia ovat grid-layout, jolla voi luoda oman responsiivisen ruudukon. Flexbox joka venyttää elementit laatikon sisään nätisti. Siirtymiä ja animaatioita varten ovat transaktiot. (Gasston 2011.)

2.5 Javascript & Jquery

JavaScript kehitettiin HTML ja CSS kielten rinnalle tuomaan toiminnallisuutta web- sivujen luomiseen. Sitä tukee kaikki modernit verkkoselaimet. Javascriptiä käytetään myös palvelinpuolen verkko-ohjelmoinnissa muun muuassa node.js-kirjaston kanssa. Lisäksi sitä käytetään pelien ohjelmoinnissa, sekä mobiili ja työpöytä sovelluksissa (wikipedia).

Javascriptilla on puhtaasti olio-ohjelmointi kieli. Sillä ei ole luokkia, niin kuin javassa. Javascriptissa lähes kaikki data on objekteja, tai niihin päästään käsiksi objektin kautta. Javascriptissa voidaan luoda, muuttaa tai poistaa HTML-objekteja milloin vain. Javascipt

korvaa luokkien puuttumisen kahdella eri tietotyypillä, primitiivinen tyyppi ja viittaustyyppi (reference type). (Zakas, Nicholas C.)

Primitiivisiä tyyppejä on viisi erilaista ja on aika yksiselitteisiä. Boolean on muuttuja, joka voi olla true tai false. Number, johon voi tallentaa minkä tahansa numeron tai liukunumeron. String tyyppi on kirjain tai kirjainyhdistelmä: String=”kirjaimia”. Null tietotyypillä on vain yksi arvo null, joka tarkoittaa tyhjää tai puuttuvaa arvoa. Undefined tyyppi on melkein sama kuin null mutta kertoo, että arvoa ei ole vielä määritetty. Primitiiviset tyypitkin käyttäytyvät vähän kuin objektit, koska niillä on olemassa metodeita mitä voidaan hyödyntää.(Zakas, Nicholas C.)

```
Var name =”teemu”;                    //otetaan String muuttujaan talteen nimi
Var ekakirjain = name.charAt(0);      //otetaan name muuttujan ensimmäinen kirjain
Ekakirjain.toUpperCase();             //muutetaan nimen ekakirjain isolla ”Teemu”
```

Referenssi tyypeillä tarkoitetaan objekteja jotka käyttäytyy samaan tapaan kuin javan luokat. Objekti voi olla mitä tahansa ja niille voidaan lisätä arvoja tai muuttaa niitä. Otetaan esimerkki objektin luomisesta. (Zakas, Nicholas C.)

```
Var objekti = new Object();           //objektin luominen
```

Objekteille voidaan asettaa muuttujia seuraavasti:

```
Var kirja = new Object();             //luodaan kirja-objekti
Kirja.nimi = ”Javascriptin perusteita”;
Kirja.isbn =”123456789”;
```

Javascriptissa on myös sisäänrakennettuja objekteja, jotka on tarkoitettu tietyn asian käsittelyyn. Muita objekteja ovat: Array (järjestykseton taulukko), Date(päivämäärä), Error(virheviesti), function, object ja RegExp(Regular Expression). (Zakas, Nicholas C.)

Javascriptissä voidaan muodostaa myös funktioita, joilla voidaan ottaa vastaan muuttuja tai arvo ja veivata siitä mitä ikinä haluammekaan palauttaa selaimelle. Funktiota voidaan sitten kutsua suoraan onClick-metodilla koodista. Seuraava funktio ottaa vastaan minkä tahansa arvon, laskee itsensä yhteen ja palauttaa tuloksen selaimelle.

```
Function palauta(arvo){
Var tulos= Arvo+arvo;
Return tulos;
}
```

Javascriptillä voidaan tulostaa virheviestejä konsoliin. Tämä auttaa vikojen paikantamisessa ja testauksessa. Konsoliin voidaan tulostaa käytännössä mitä tahansa console.log

komennolla. Javascriptilla on yleistä tehdä myös ponnahdusikkunoita. Niitä on kolme erilaista: confirm, alert ja prompt. Yleensä alert-ikkunaa käytetään juuri virheviestien tulostamiseen. Virheviestin voi tulostaa myös suoraan elementin päälle innerHTML toiminnolla. (Zakas, Nicholas C.)

Kehittäjät haluavat sivuilleen interaktiivisia toimintoja kauniin ulkokuoren lisäksi. Tähän on keksitty eri Javascript kirjastoja, jotka tarjoaa helpompia tapoja kehittäjille lisätä toimintoja vähemmällä kokemuksella. Jquery tarjoaa abstrakteja kerroksia verkkosivun scriptaukseen. Se on laajennettavissa ja siihen onkin monia liitännäistä, jotka tuovat lisää toiminnallisuutta. Jqueryn voi jakaa perusominaisuuksiin seuraavasti:

- Dokumenttien elementteihin pääsy
- Web-sivun ulkonäön muuttaminen
- Dokumentin sisällön muokkaaminen
- Käyttäjän vuorovaikutukseen vastaaminen
- Animoida muutoksia dokumentissa
- Hae palvelimelta tietoa päivittämättä sivua
- Yksinkertaistaa monimutkaisia Javascript toimintoja

Koska Javascript luetaan heti kun se kohdataan koodissa ja Jquery kirjasto tulee HTML tiedoston alkuun. Ei Jquery pääse tällöin käsiksi vielä sivun rakenteeseen. Jqueryn $(document).ready() menetelmä odottaa kunnes koko dokumentti on ladattu ja vasta sen jälkeen suorittaa sen sisällä olevat toiminnot. (Chaffer 2014.)

Dokumentin elementteihin pääsy: Ilman helpottavia kirjastoja ohjelmoijat joutuvat kirjoittamaan useita rivejä koodia päästäkseen käsiksi tiettyyn osaan DOM-(dokument object model) puuta, eli sivun rakennetta. Jqueryssä on monia tehokkaita valitsimia(selectors), joilla voidaan hakea tietty osa dokumentista jota halutaan muokata. (Chaffer 2014.)

```
$(’div.content’).find(’p’); //Hae div class=”content” ja etsi kaikki
sen sisällä olevat p elementit
```

Jquery:llä voidaan muokata CSS tyylejä tai vaihtaa elementin luokka painikkeen painalluksella, jopa sen jälkeen kun sivu on ladattu. (Chaffer 2014.)

```
$('ul > li:first').addClass('active');  //Lisää ul listan sisällä
olevaan ensimmäiseen li elementtiin luokka ‘active’
```

Jquery antaa mahdollisuuden muuttaa mitä tahansa dokumentin sisältöä. Tekstiä tai kuvia voidaan vaihtaa tai lisätä tai kirjoittaa vaikka koko html tiedoston jquerynä. (Chaffer 2014.)

```
$('#container').append('<a href="more.html">more</a>'); //Kiinnitä
elementtiin id=”container”, linkki more.html sivulle.
```

Event-handler API:lla voidaan napata tapahtumakutsuja selaimelta kuten silloin kun käyttäjä painaa linkkiä tai painiketta. Jqueryllä voidaan napata painikkeen painallus kiinni ja liittää jokin toiminnallisuus tapahtumaan. Jquery poistaa selainten epäjohdonmukaisuuksia tulkita HTML:n event-handler tekniikoita ja takaa että sivu toimii eri selaimilla samankaltaisesti. Muita tapahtumia jotka Jquery tunnistaa ovat: click, dblclick, mouseenter, mouseleave, keypress, keydown, keyup, submit, change, focus, blur, load, resize, scroll ja unload. (Chaffer 2014.)

```
// näytä div.details kun painiketta painetaan
$('#button').click(function() {
   $('.div.details').show();

//lisää tai poista class=”active” painikkeelta
$('#button').toggleClass('active');
   // muita asetuksia painikkeen painamiseen liittyen...
});

//piiloita p elementti kun sitä tuplaklikkaa
$("p").dblclick(function(){
     $(this).hide();
});
```

Animaatioita tarvitaan visuaalisen palautteen antamiseen sivun käyttäjälle. Visuaalisella palautteella voidaan ohjata käyttäjää oikeaan suuntaan tai antaa välitöntä palautetta sivun toiminnasta. Jqueryn kirjasto tarjoaakin kasan erilaisia liukuma ja siirtymä efektejä tätä varten - kuten myös työkalut omien efektien tekemiseen. (Chaffer 2014.) Esimerkkinä voidaan vierittää näkymää tietyn elementin kohdalle:

```
//Asetetaan muuttujat: container ja targetElement mihin sivu vieri-
tetään
var container = $('#container');
var scrollTo = $('#targetElement');
//Vieritä haluttuun elementtiin -> targetElement sijainti ylhäältä
minus container sijainti ylhäältä.
container.scrollTop(
            scrollTo.offset().top - container.offset().top + con-
            tainer.scrollTop()
);

//Voit myös tehdä saman animaationa:
container.animate({
            scrollTop: scrollTo.offset().top –
            container.offset().top + container.scrollTop()
});

//Voit myös vierittää haluttuun elementtiin käyttämällä id:tä
var container = $('#container');
```

```
container.scrollTop(
            $('#targetElement').offset().top - container.off-
set().top
);

//Elementtejä voidaan animoida antamalla niille uusia arvoja. Seu-
raava esimerkki muuttaa hitaasti div-laatikon kokoa, läpinäkyvyyttä
ja vaihtaa lopuksi sen taustaväriksi punaisen.
$("button").click(function(){
    var div = $("div");
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate ({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({background-color: 'red', opacity: '0.8'}, "slow");
});
```

Jqueryllä voidaan ladata sisältöä suoraan palvelimelta käyttäen AJAX:ia. Jqueryn AJAX toiminnoilla voidaan pyytää palvelimelta XML-, HTML-, JSON- tai tekstitiedostoja. Jquery tarjoaa GET PUT DELETE metodit AJAX:n käyttöön. (Chaffer 2014.)

```
//Esimerkki tekstitiedoston lataamisesta div1 laatikkoon kun painaa
button1 painiketta
$(document).ready(function(){
    $("#button1").click(function(){
        $("#div1").load("tekstiä.txt");
    });
});
```

Dokumenttikohtaisien toimintojen lisäksi Jquery tarjoaa mahdollisuuden jatkaa tai yksinkertaistaa Javascript kirjaston ominaisuuksia kuten tietojen iterointi (yksittäin läpikäynti) ja taulukon manipulointi.(Chaffer 2014.)

2.6 XML & XSLT

XML (Extensible Markuo Language) on yksinkertainen datan merkkauskieli. Sillä voidaan kuvata dokumentin data ja rakenne. Se on metadatan merkkauskieli SGML:n (Standard Generalized Markup Language) alijoukko. XML suunniteltiin kuvaamaan tietoa ja helpottamaan elektronista datan julkaisua isolla skaalalla. Se soveltuu erinomaisesti geneerisen HTML sisällön tuotantoon. (W3C, 2008).

XML-dokumentti alkaa prologilla, joka kertoo XML:n version ja dokumentin koodauksen. (W3, 2008).

```
<?xml version="1.0" encoding="UTF-8"?>
```

Dokumentille voi antaa doctype määrityksen, jos vaikka haluaa dokumentista (X)HTML-dokumentin. (X)HTML on HTML kielen versio, joka on siivottu XML-kieleksi. (W3C 2008).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XML:lle voi antaa käsittelyohjeita. Esimerkiksi XSLT-tyylitiedoston voi määrittää lisäämällä tyylitiedoston sijaintipolun dokumentille seuraavasti:

```
<?xml-stylesheet type="text/xsl" href="../xsl/course.xsl"?>
```

XML:n data kuvataan hyvin samaan tapaan kuin HTML-kielessäkin. Data merkataan elementeillä, jotka toimivat ilmentyminä. Kaikki data tulee olla yhden juuri elementin sisällä. Muuten elementtejä voi olla rajaton määrä sisäkkäin tai allekkain. Kaikki elementit ovat ennaltamäärittämättömiä, jolloin elementtien käyttäytyminen pitää määrittää itse. Elementeille voidaan antaa myös attribuutteja. (W3C 2008.)

```
<root>
<!-- kommentit jätetään samoin kun HTML:ssä -->
        <isantaElementti attribuutti=”arvo”>
                <lapsiElementti1 arvo1=”jokuArvo” arvo2=”toinen_arvo”>
                        Lorem ipsum Lorem ipsum…
                </lapsielementti>
        </isantaElementti>
        <isantaElementti attribuutti=”arvo”>
                <lapsiElementti2>
                        Lorem ipsum Lorem ipsum…
                        <tyhjaElementti url=”www.google.com”/>
                </lapsielementti>
        </isantaElementti>
</root>
```

Dokumentillle voi halutessaan määrittää nimiavaruuden, joka erottaa elementtien nimet omaan avaruuteensa, jotta ne eivät mene sekaisin muiden käytettävien nimien kanssa. (W3C 2008.)

XSLT (Extensible Stylesheet Language Transformations) on w3c:n mukaan suositeltu tyylitiedosto XML:ää varten. XSLT toimii samaan tapaan kuin CSS toimii HTML:lle, eli alkuperäistä XML-dokumenttia ei muuteta, vaan XSLT tekee uuden XML-tiedosto instanssin. XSLT:llä voidaan saada XML-dokumentista muunnettua niin HTML-kieltä, kuin koneluettavaa CSV:täkin. Sillä voidaan muuttaa, piiloittaa tai poistaa elementtejä ja attribuutteja suoraan ulostulevasta tiedostosta. XSLT käyttää datan hakemiseen XML-tiedostoista XPath merkin-täkieltä. (W3C, 1999.)

XSLT otetaan käyttöön lisäämällä sille ensin XML dokumentin määritykset ja sen jälkeen käytettävän nimiavaruuden osoite:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
```

Nimiavaruuden lisäämisen jälkeen voidaan määrittää miksi kieleksi dokumentti halutaan muuttaa. Output method kannattaa määrittää, sillä Internet Explorer ei ainakaan osaa kääntää muutoksella tehtyä HTML-sisältöä oikein. Nimiavaruus estää mahdolliset yhteentörmäykset muiden tekniikoiden käyttämien nimien kanssa. Käytetään W3:n omaa XSL muutoksille tarkoitettua nimiavaruutta.

XML-dokumentin voisi muuttaa helposti HTML-dokumentiksi valitsemalla XML dokumentin juuren ja lisäämällä html-elementit sen sisään. Tämän jälkeen käytettäisiin XSLT:n osoittimia ja suodatettaisiin haluamamme tiedot XML-dokumentista haluamaamme tapaan.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      …
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Tyylitiedostossa voidaan käyttää vain seuraavia elementtejä: xsl:import, xsl:include, xsl:strip-space, xsl:preserve-space, xsl:output, xsl:key, xsl:decimal-format, xsl:namespace-alias, xsl:attribute-setxsl:variable, xsl:param ja xsl:template. (W3C 1999.)

XPath on (XML Path) kieli, jolla voidaan osoittaa XML-dokumentin osia. Sillä voidaan valita elementtejä käsittelyä varten, myös elementtien attribuutteja voidaan valita. XPath sisältää muun muuassa seuraavat valitsimet XML-dokumenteille: (W3C 1999).

<xsl:template>	Käytetään osoittamaan tiettyä elementtiä. Template match=”/” osoittaisi dokumentin juureen.
<xsl:value-of>	Käytetään tietojen hakemiseen XML-tagista.
<xsl:attribute>	Käytetään uuden attribuutin lisäämiseen isäntäelementille. Esimerkki havainnollistaa kolmen ensimmäisen valitsimen käytön

muuntamalla XML-dokumentin sisällä olevat tehtävät avautumaan uuteen välilehteen. Elementit muutetaan a-elementeiksi ja niille annetaan attribuutti src, jonka sisään kopioidaan tiedostojen osoite:

```
//XML dokumentin elementti.
<tehtavä file=”exercises/01.01.xml”/>
<tehtavä file=”exercises/01.02.xml”/>

//XSLT ottaa tehtävä elementin vastaan ja muuntaa avaamaan ne uuteen selainikkunaan.
<template match=”tehtava”>        //valitaan ”tehtava” elementit
<a target=”_blank”>               //linkki joka avautuu uuteen sivuun
  <xsl:attribute name=”src”>                //lisätään <a src=””>
     <xsl:value-of select=”@file”/>         //lisätään <a src=”@file”>
  </xsl:attribute>
Tehtävä<xsl:number/>              //lisää numeron jokaiselle tehtävälle.
                                  //ie. Tehtävä1…tehtävä2…tehtävä3
</a>
</template>
```

<xsl:for-each> Käytetään tietojen läpikäymiseen jos elementtejä on monia ja halutaan käydä kaikki läpi. Tällöin voidaan suodattaa tietoa helposti ja listamaisesti.

<xsl:sort> Käytetään XML-tietojen lajittelemiseen.

<xsl:if> Käytetään toteuttamaan ehtolause.

<xsl:choose> ja
<xsl:otherwise> Käytetään toteuttamaan moniulotteinen JA-TAI ehtolauseke.

Valitsimien lisäksi XSLT sisältää joitain omia funktioita. Niillä voidaan esimerkiksi generoida yksilöllinen tunnus jollekkin elementille lisäämällä generate-id() funktio sille.

XSLT:n hyöty tulee siitä, että se antaa paljon valtaa siihen kuinka XML-lähdedokumentti näytetään. Sillä voidaan ottaa käsittelyyn tietty tekstinpätkä, osio tai elementin arvo ja valita mitä siitä näytetään. XSLT pohjaa voidaan käyttää muuntamaan useita XML dokumentteja, jotka noudattaa samaa elementtirakennetta, helpottaen näin uusien sivujen lisäämisen projektiin. XSLT tukee muunnoksen tekemistä niin palvelimella kuin selaimen päässäkin. Jos muunnettavaa on paljon, niin kannattaa miettiä palvelimen päässä muunnosta suorituskyvyn vuoksi. (W3C 1999.)

2.7 Bootstrap

Bootstrap (tunnetaan myös Twitter Bootstrap) on yksi, ellei suosituin front-end sovelluskehys responsiivisten HTML- CSS ja Javascript projektien tekoon. Bootstrap on kehitetty mobile first periaatteet mielessä, jolloin omakin kehitys tulisi tukea mobile firstin periaatteita.(Bootstrap 2015.)

Bootstrapin kehitti Mark Otto ja Jacob Thornton Twitterillä työskennellessään vuonna 2010. Bootstrap kehitettiin open-source sovelluskehykseksi, joka tarkoittaa, että sitä saa vapaasti ladata ja muokata omiin projekteihin sopivaksi. Virallinen Bootstrapin versio tuli kaikkien ladattavaksi elokuussa vuonna 2011. Sen jälkeen bootstrapista on julkaistu noin 20 versiota. Viimeisimpänä versio 3.3.5, jossa uutena asiana sisällytettiin kirjastot olemaan vakiona responsiivisia ja mobile first lähestymistavalla.(Bootstrap 2015.)

Bootstrapin käyttöönotto uuteen projektiin tapahtuu sivulle ladattavilla bootstrap.js ja bootstrap.css tiedostoilla. Tiedostoja on kaksi versiota, joista edellä mainitut ovat kehitysversioita. Lisäksi tiedostoista saa niin sanotut minifoidut versiot, joista on poistettu kaikki välilyönnit ja rivinvaihdot. Näin ollen saadaan pienempi tiedostokoko. Tiedostoja tarjotaan myös CDN-linkkeinä(Content Delivery Network) mxcdn.com:n palvelimella ladattavaksi urlin avulla. CDN linkit lisätään index.html tiedostoon <body> elementin loppuun seuraavasti.

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/boot-
strap/3.3.5/css/bootstrap.min.css">

<!-- Latest compiled and minified JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/boot-
strap.min.js"></script>
```

Bootstrap on suhteellisen raskas monipuolisuutensa takia, joten sen kehittäjät antavat mahdollisuuden kustomoida ladattavia paketteja omien tarpeiden mukaan. Silloin saa vain ne ominaisuudet mitä itse tarvitsee.

Bootstrap tuo mukanaan monia Javascript liitännäisiä ja komponentteja sivuston tekemiseen. Kaikki liitännäiset käyttävät Jquery-kirjastoa, joten se tulee sisällyttää mukaan sivustoon. Liitännäisiin kuuluvat transitiot, jotka tuovat siirtymä efektit. Modaalit, jotka tuovat ruudulle avautuvat dialogi ikkunat. Dropdownit sallii helpon pudotusvalikoiden lisäämisen, Scrollspy liitännäinen vaihtaa valikon aktiivitilaa samalla kun sivua selaa eteenpäin. Tab-liitännäisellä voi luoda näyttäviä välilehtivalikoita. Ohjetekstejä, joita tyypillisesti lisätään esimerkiksi nappuloille, ja ne ilmestyy kun hiiri viedään niiden päälle, voi tehdä Tooltip lii-

tännäisellä. Popover liitännäisellä voi tuoda pienen sisältölaatikon esiin painettavan elementin ympärille. Alert liitännäisellä voidaan tehdä varoitus laatikoita. Button liitännäisellä tuodaan button komponenteille toiminnallisuutta kuten ”painettu” tilan. Collapse liitännäisellä voi tehdä haitarimaisia sisältölaatikoita jotka avautuu painettaessa. Carousel liitännäisellä saa helposti näyttävän kuva ja sisältö karusellin jota voi selata nuolilla. Affix liitännäinen tuo staattisen navigaatio valikon joka tarrautuu johonkin kohtaan ruutua paikoilleen.(Bootstrap, 2015.)

Bootstrapin komponentit ovat valmiiksi tyyliteltyjä uudelleenkäytettäviä sivun osia jotka nopeuttavat sivuston tekemistä. Komponentteja voi itse muokata CSS:n avulla ja ne ovat yleensä luokkia, mitä tiputellaan elementtien arvoiksi. Komponentteihin kuuluu erilaisia nappuloita, lomakkeita, latauspalikoita, ikoneita ja navigointi elementtejä.

Bootstrapin tärkeimpiin CSS ominaisuuksin kuuluu responsiivinen mobile first ruudukko systeemi, joka koostuu 12 eri ruudusta. Ruudukkosysteemi toimii siten, että sisältö asetetaan säiliöön. Säiliö luokkia on kahdenlaisia, keskitetty kiinteä leveys(.container) ja juokseva kokoselaimen levynen(.container-fluid). Säiliö luokka annetaan yleensä body elementille. Kaikki sisältö puolestaan pitää asettaa row, eli rivi luokan sisälle. Rivi luokka annetaan div elementille. Rivit jaetaan 12 sarakkeeseen johon asetetaan sisältöä. Sisällön vaatiman tila ilmoitetaan elementille sarakkeiden määränä. Bootstrapissä viitataan rivin yhteen sarakkeeseen columnilla. Viittaukset muodostetaan lisäämällä halutulle elementille luokaksi esimerkiksi ”.col-xs-4”, jolloin kyseinen elementti vie 4/12 osaa, eli 1/3 rivin leveydestä. Samanlailla voidaan muodostaa helposti vaikka blogi, jossa sivuston tekstipalstoja olisi 3 vierekkäin symmetrisesti. Luokan nimessä ”xs” viittaa mediakyselyihin, eli breakpointteihin jossa ruudun leveys ylittää tai alittaa tietyn pisteen. Näillä voi esimerkiksi asettaa 2 saraketta näkymään pinottuna mobiilinäytöillä ja vierekkäin asettamalla sarakkeiden elementeille class=”col-xs-12 col-sm-6”. Vaihtoehtoina breakpointeille ovat bootstrapissa col-xs(alle 768px), col-sm(yli 768px), col-md(998px tai yli) ja col-lg(1200px tai yli). Luokkia käytettäessä tulee huomioida että luokat periytyy ylöspäin, eli jos asettaa ”.col-sm-5”, niin saman verran sarakkeita varataan md ja lg luokilta, mutta xs luokka pysyy ennallaan. (Bootstrap 2015.)

.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1	.Col-xs-1
.Col-xs-4				.Col-xs-4				.Col-xs-4			
.Col-xs-12											

Bootstrapin ytimeen leivotut mediakyselyt koostuvat neljästä breakpointista: alle 768px, yli 768px, 992px tai yli ja 1200px tai yli. Näillä voidaan järjestellä, piilottaa tai muokata sisältöä erikokoisille näytöille sopiviksi. Bootstrappiin on asetettu Sass-esikääntäjä kielellä seuraavat mediakyselyt jotka vastaavat xs-, sm-, md- ja lg-luokkia:

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */
...asetuksia mobiilille...
/* Small devices (tablets, 768px and up) */
@media (min-width: @screen-sm-min) { ...asetuksia tableteille... }

/* Medium devices (desktops, 992px and up) */
@media (min-width: @screen-md-min) { ...asetuksia kannettaville... }

/* Large devices (large desktops, 1200px and up) */
@media (min-width: @screen-lg-min) { ...aetuksia pöytäkoneille ja isommille... }
```

Saman voi tehdä puhtaasti css:llä, jos ja kun haluaa asettaa omia asetuksia eri ruudun leveyksille. Nämä breakpointit tulee pitää samoina kun bootstrapissä, jotta tyyli ei vaihdu ennen ruudukon vaihtumista pienempään.

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */
...asetuksia mobiilille...
/* Small Devices, .visible-sm-* */

@media (min-width: 768px) {...asetuksia tableteille...}

/* Medium Devices, .visible-md-* */

@media (min-width: 992px) {...asetuksia kannettaville... }

/* Large Devices, .visible-lg-* */

@media (min-width: 1200px) {...aetuksia pöytäkoneille ja isommille... }
```

Bootstrapin huonoiksi puoliksi nousee varmaan sen laaja käyttö, jolloin monet sivut näyttävät samanlaisilta. Bootstrapin ominaisuuksia voi toki muuttaa mutta usein se on aika työlästä. Hyviä puolia ovat sen nopea käyttöönotto ja helppo opittavuus. Bootstrapin suosion myötä siihen on kehitetty lukuisia liitännäisiä ja lisä-osia, joilla saa lisää toiminnallisuutta sivulle nopeasti. Bootstrapistä ei ole pakko ottaa kaikkia ominaisuuksia käyttöön, joka onkin mielestäni paras tapa suorituskyvyn kannalta. Itse en varmaan käytä jatkossa kun ruudukoita ja nappuloita kehityksessäni. Suosittelen kaikkien kokeilemaan, sillä Bootstrapissa on älykkäitä ominaisuuksia joiden kautta oppii tekemään niitä itse.

3 Sivuston toteutus

Tässä kappaleessa käydään läpi miten esiteltyjä tekniikoita käytetään kurssisivun luomisessa. Ensin esitetään työvälineet, joita tarvitaan kehittämiseen ja testaamiseen. Tämän jälkeen käydään läpi miten sivun rakenne muodostetaan ja miten voisimme tehdä uuden kurssin käyttämällä samaa xslt-tiedostoa.

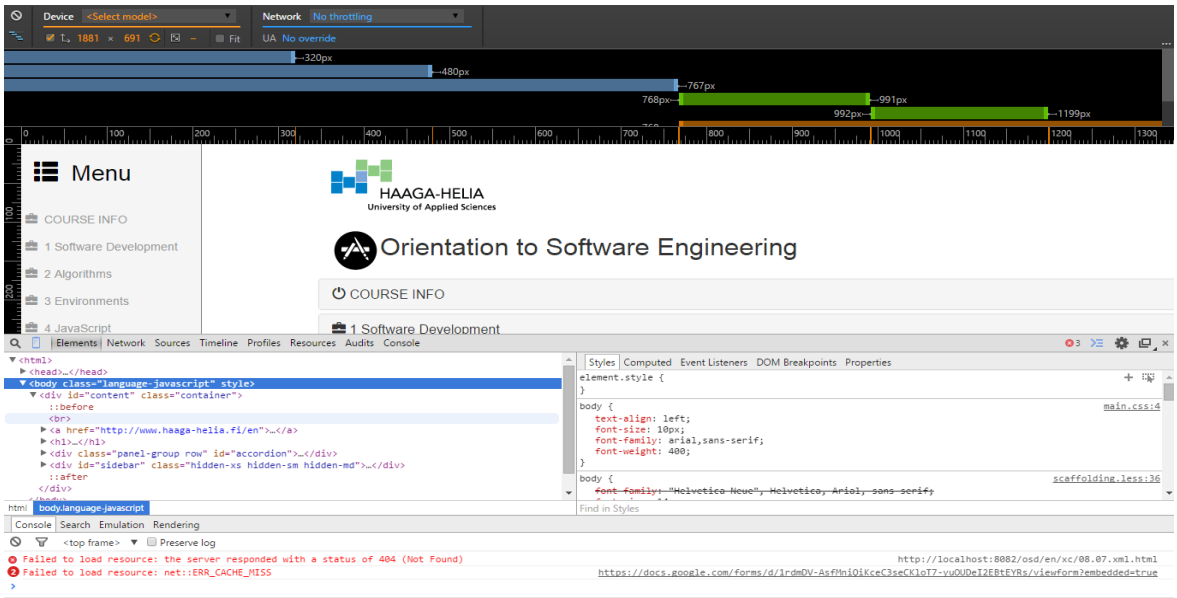
Responsiivisuus kohdassa käydään läpi miten responsiivisuus toteutettiin projektiin. Eli miten sivu saadaan skaalautumaan halutusti resoluutiosta riippumatta. Marcotte määritteli responsiivisuuden joustavaan- tai juoksevaan ruudukkoon, joustaviin kuviin ja media kyselyihin. Nämä otetaan tarkempaan tarkasteluun dokumentin responsiivisuus osiossa.

Käytettävyys osiossa käydään läpi tekniikoita, joilla luodaan sivulle vaadittavat toiminnallisuudet. Lisäksi luodaan pöytäkone käyttäjien selailua helpottava staattinen sivuvalikko.

3.1 Käytetyt työvälineet

Selaineditorit

Selaineditoreista on iso apu verkkosivujen käyttöliittymien teossa. Niillä löytää helposti missä sivun osassa on vikaa. Selaineditorissa näemme sivun HTML-rakenteen ja voimme avata näkymään suoraan jonkun elementin painamalla hiiren oikeaa painiketta elementin päällä ja valitsemalla inspect element. Selaineditoreilla pystytään muuttamaan reaaliaikaisesti sivun tyylejä ja nähdä miten muutokset vaikuttaa. Editoreilla nähdään kaikki kutsut palvelimelle ja kauan niissä kestää, joka on hyödyllistä kun aletaan optimoimaan sivun latausaikoja.



Kuva 6 - Chromen selaineditori

Chromen ja Internet Explorerin selaimissa on lisäksi emulaattorit joilla voidaan testata sivun käyttäytymistä eri mobiililaitteilla ja selaimilla. IE:n tarjoama emulaattori vanhemmille IE:n versioille ja Windows phone emulaattori ovat hyödyllisiä jos sivun haluaa toimimaan vanhemmillakin versioilla. Lisäksi editorien consolessa voi ajaa vaikka javascriptia reaaliaikaisesti. Esimerkiksi kirjoittamalla consoleen Document.getElementById(”123”) hakee kaikki elementit id:llä 123. Toinen helpottava esimerkki voisi olla jos halutaan testata, että kaikki tooltipit(ohjeteksti, joka ilmestyy painikkeen viereen laatikossa, yleensä myös ylipitkät otsikot) toimivat niin kuin pitää. Kirjotettaisiin konsoliin: ”$('.btn').trigger('mouseover'); “, jolloin kaikki “.btn”-luokan omaavat käyttäytyisivät kuin niiden päälle olisi viety hiiri ja kaikki tooltipit tulevat näkyviin niille kuuluville paikoille. Olettaen tietysti, että tooltipit ovat asetettu mouseover eventtiin.

Eclipse

Eclipse IDE(Integrated Development Environment) on kehitystyökaly kaikenlaiseen kehittämiseen. Sen pääkäyttötarkoitus on kuitenkin Java-kehitys. Eclipse itsekin on nimittäin kirjoitettu Javalla. Eclipse on erittäin laajalti laajennettavissa. Siihen on tehty lisä-osat lähes kaikille eri ohjelmointi-kielille. Tuen eri kielille voi helposti ladata editorin sisältä help-välilehdellä löytyvästä Eclipse Marketplace:sta. Eclipsen uusin versio Mars(4.5), on seuraava natiivisti Javan versio 8:aa tukeva kehitysympäristö.

Github

Github on web-pohjainen Git-varasto palvelu. Se siis varastoi tekemiäsi projekteja. Varastot voivat olla julkisia tai yksityisiä, jolloin se sopii hyvin myös kehitystiimin lähdekoodin hallintatyökaluksi. Github tarjoaa hyvän versiohallinnan ja lähdekoodin hallinnan (SCM).

Github otetaan käyttöön luomalla tunnukset Githubin sivuille. Tämän jälkeen luodaan reposition, eli säiliö projektia varten. Projekti sitten kopioidaan joko suoraan eclipseen tai ensin omalle koneelle komentorivityökalulla git clone komennolla. Projektin voi tuoda suoraan Eclipseen lokaalisti tai linkin avulla välilehdestä file->import->projects from Git. Lisätään sen jälkeen linkki projektiin. Kun projekti on tuotu tällä tavalla eclipseen, niin tehdyt muutokset näkyvät Git komentorivi työkalussa kun kirjoittaa git status.

Githubissa pystyy tekemään brancheja, eli haaroja omaan projektiin. Vähän kuin puu olisi projekti, josta lähtee useita oksia, jotka ovat haaroja. Kun tehdään uusi haara ja siirrytään siihen tekemään muutoksia, niin tällöin tehdyt muutokset koskettavat vain sitä haaraa missä ollaan. Master-branch on projektissa se päähaara tai runko, joka tulisi olla aina toimivassa kunnossa ja ominaisuuksien kehittäminen tulisi tapahtua muissa haaroissa, jotka

ovat nimetty kehitettävän toiminnallisuuden mukaan. Valmiit ominaisuudet yhdistetään master-haaraan merge:llä tai rebase:lla ja sen jälkeen pusketaan Githubin palvelimelle.

Githubia voidaan käyttää myös Git:n konsolityökalua koneelta suoraan. Tässä projektissa käytin Git Bash komentorivityökalua muutoksien julkaisemiseen seuraavalla tavalla:

```
Git status                    //näyttää julkaisemattomat muutokset ja tilan
Git add .                     //lisää kaikki muutoksia sisältävät tiedostot staging alueelle
Git comment –m ”jotain muutoksia..” //lisää kommentti mitä muutoksia tehty
Git push                      //pusketaan muutokset ja kommentit githubiin
TAI
Git push origin master        // sama kuin git push mutta tarkentaa, että lokaali branch menee vain
                              master branchiin.
```

Githubiin saa integroitua helposti GH-Pages lisäosan, joka julkaisee githubin projektin verkkoon. GH-Pages toimii Githubissa omassa branchissaan. Kaikki muutokset projektiin saadaan puskettua samalla gh-pages sivulle, joka julkaisee suoraan muutokset sivulle. Projektissa voidaan asettaa gh-pages ottamaan muutokset automaattisesti samalla kun alkuperäiseen kansioon julkaistaan muutoksia. Manuaalisesti julkaisu gh-pagesiin menee komentorivi-ikkunassa seuraavasti:

```
Git checkout gh-pages         //Mene gh-pages branchiin
Git rebase master             //tuo master branchin uudet muutokset gh-pagesin päälle
Git push origin gh-pages      //julkaise lokaali gh-pages branch githubiin
Git checkout master           //takaisin master branchiin
```

Prism

Prism-kirjasto on ilmainen avoimen lähdekoodin kirjasto tekstin korostamiseen. Tässä projektissa sitä käytettiin koodi-esimerkkien korostamiseen. Prism-kirjaston saa käyttöön ottamalla kirjastopaketin sivulta http://prismjs.com/ ja lisäämällä se projektiin. Tämän jälkeen voidaan käyttää valmiita elementtejä, jotka luovat korostetun laatikon sivulle. Esimerkki miten kirjasto korostaa koodia käyttämällä valmiita elementtejä:

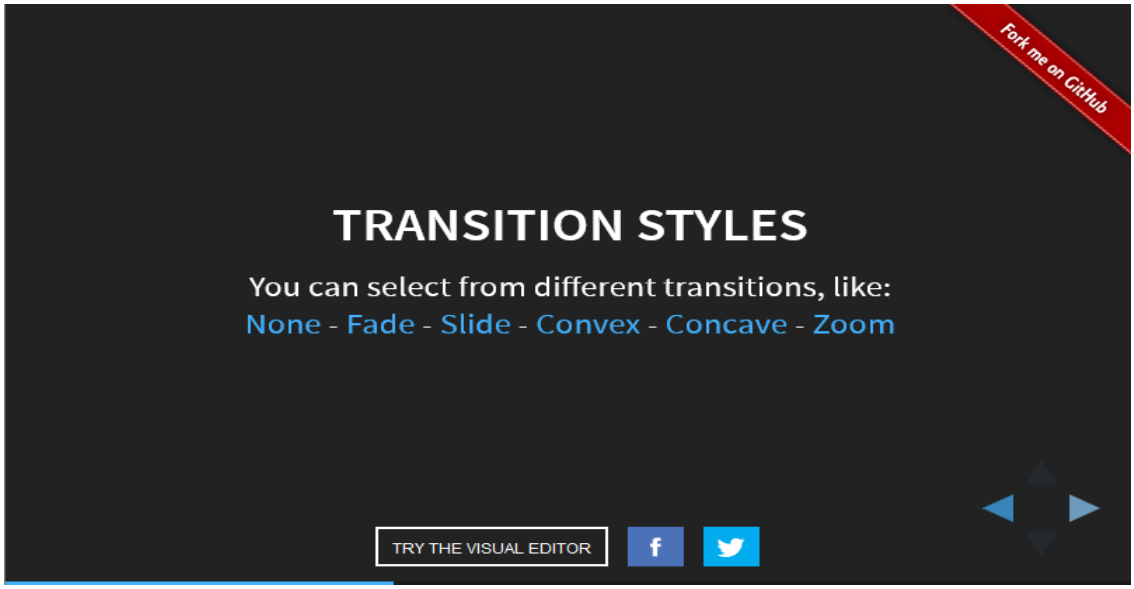
```
<pre><code>
//Example: a and b are variables
var a = 5;
var b = 10;
</code></pre>
```

```
//Example: a and b are variables
var a = 5;
var b = 10;
```

Kuva 7. Prism-kirjaston syntax highlight

Reveal

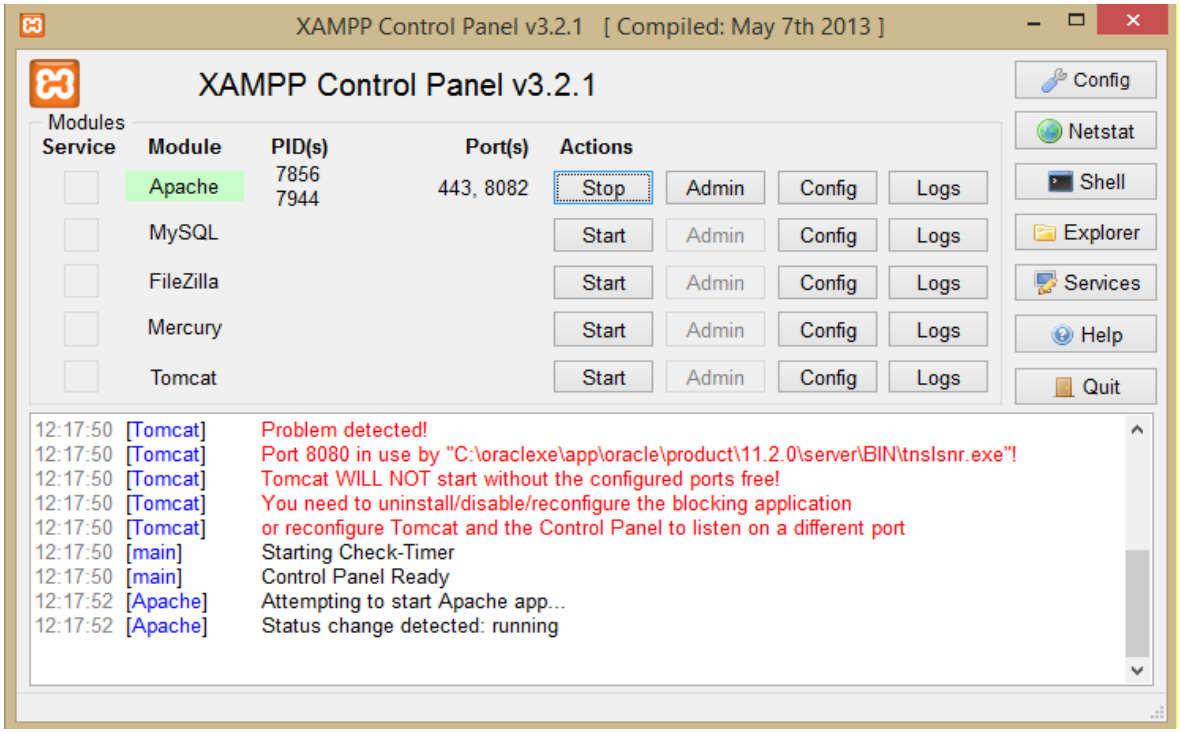
Reveal-kirjasto on ilmainen avoimen lähdekoodin kirjasto näyttävien mobiilioptimoitujen dia-esitysten tekoon. Sillä luodaan helposti dia-esityksiä, jossa voi siirtyä nuolinäppäimillä tai pyyhkäisytoiminnoilla mobiilissa. Siirtymiä voi olla ylä-, tai alatasoille ja sivulle päin. Kynnys aloittaa slaidien tekemisen tällä kirjastolla on pieni. Ei tarvitse osata edes koodata, että saa tehtyä omat diat. Sivulla www.slides.com pystyy tehdä slidet ”drag and drop” tyylisesti hiirellä vetelemällä haluttuja ominaisuuksia dokumenttiin. Valmis dia-esitys ladataan omalle sivulle. Slides-sivulle voidaan luoda ilmainen tunnus, jolla saa pienen 250mb tallennustilan sivun palvelimilta ja oikeudet tehdä dioja. (Hakim.)



Kuva 8 - Reveal diaesitys. Perustyyli esimerkki

XAMPP

XAMPP (Apache+MariaDB+PHP+Perl) on alusta, joka on suunniteltu ensisijaisesti PHP-kehitykseen. Projektiin otettiin lokaalin kehittämisen avuksi kyseinen alusta, sillä projektin data haetaan AJAX kutsuilla. Tämä tarkoittaa, ettei sivua voi suoraan avata selaimessa file-kutsulla, vaan väliin tarvitaan palvelin. XAMPP:ssa on mukana Apachen lokaali palvelin, jota voimme käyttää hyväksi.

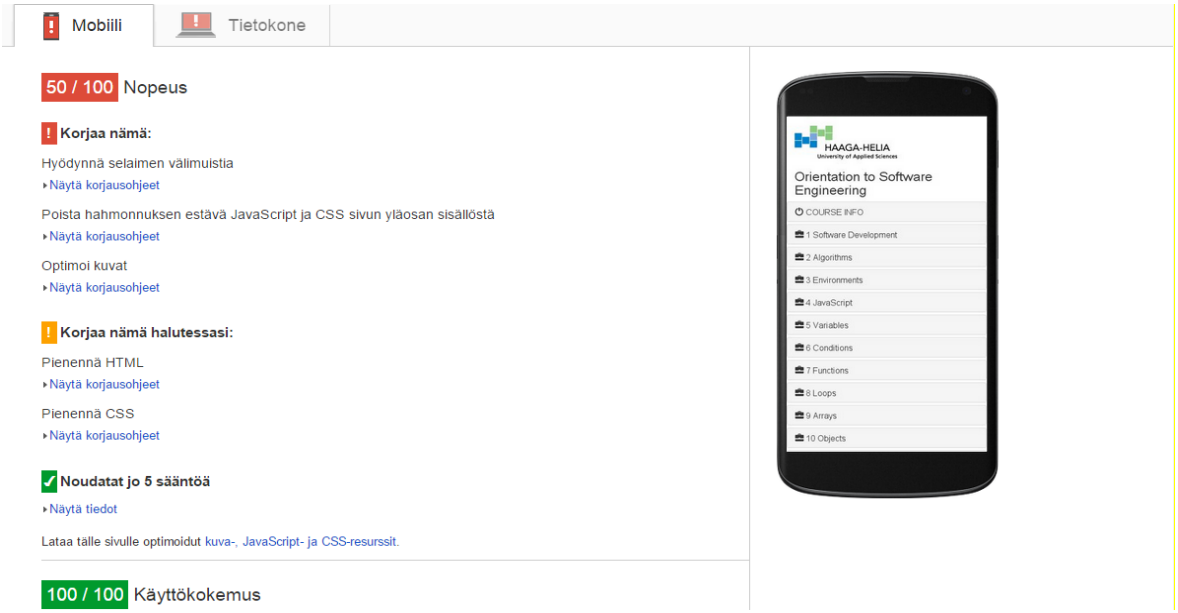


Kuva 9 – XAMPP Control-panel

XAMPP on hyvin yksinkertainen käyttää. Siinä laitetaan lokaalisti julkaistavat tiedostot htdocs-kansion sisään, jonka jälkeen käynnistetään xampp-control.exe-ohjelma ja käynnistetään Apache palvelin. Tämän jälkeen voidaan suunnistaa selaimessa osoitteeseen localhost:8080/omaprojektikansio/index.html. Control-panel-ikkunassa voi nähdä kaikki kuunneltavat portit NETSTAT painikkeen alta, jos 80 on käytössä. Apachen kuunteleman portin voi vaihtaa tiedostosta xamp/apache/conf, jossa on kohta http.conf: port 80, tai vastaavaa.

Google Page Speed Insight

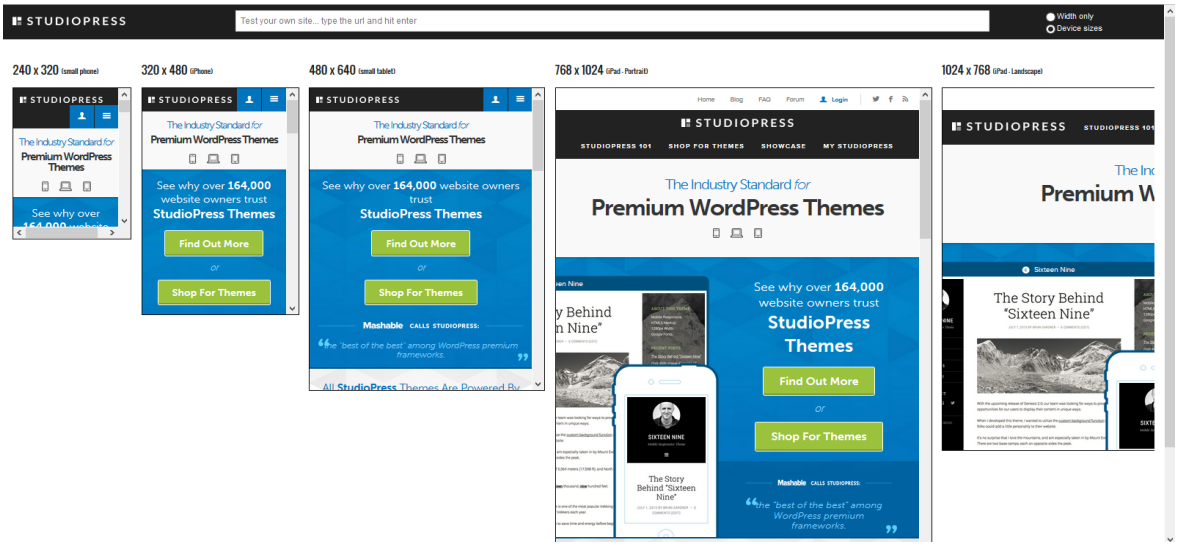
PageSpeed Insight on Googlen selaimessa toimiva sivun latausnopeus analysaattori. Se antaa neuvoja mitä voidaan tehdä, jotta sivu latautuisi nopeammin. Lisäksi voit ladata Page Speed-moduulin, jonka pitäisi automatisoida sivun optimointi, jos käytössä on Apachen tai Nginx:n palvelin.



Kuva 10 - Page Speed Insights analyysi

Studiopress testaus

Studiopress tarjoaa sivuillaan WordPress-teemoja, jotka on toteutettu heidän omalla Genesis arkkitehtuurilla. Studiopressin sivuilla on kuitenkin erittäin hyvä testaustyökalu, jolla voidaan ladata oma sivu vierekkäin erikokoisiin säiliöihin. Voidaan verrata helposti eri leveyksillä miten oma sivu skaalautuu. Testaus sivu löytyy osoitteesta www.studiopress.com/responsive.



Kuva 11 – Studiopress, responsiivista testausta

3.2 Sivuston rakenne

Koska projektissa on oleellisena osana datan muuntaminen XML dokumentista XSLT muunnoksen avulla HTML:ksi käyttäen Magic XML kirjastoa, koitan selkeyttää sitä tässä kappaleessa.

Hyöty tässä lähestymistavassa on, että käyttämällä samaa XSL-pohjaa voidaan luoda useita eri kurssisivuja, jotka näyttävät samalta ja omaavat samat toiminnot. Tulee vain muistaa käyttää XML-tiedostoissa samoja elementin nimiä, jotta XSLT osaa muuntaa oikeat elementit.

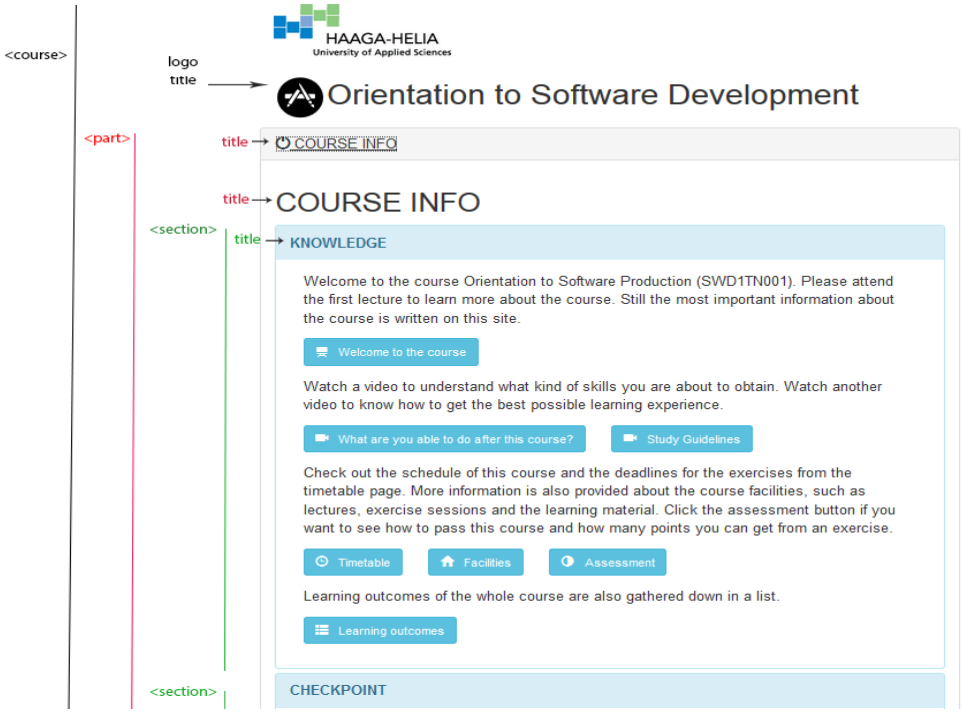
3.2.1 XML & XSLT

Sivun rakenne on yhden XML-dokumentin sisällä. Tehtävät ovat erillisillä XML-dokumenteilla, jotka lisätään rakenne dokumenttiin. XML sisältää vain sivulla näytettävää dataa ja kuvaa sivun rakenteen, mutta ei osaa lisätä loogisuutta sivulle. XML-elementit pitää vielä kääntää HTML-kieleksi, joka onnistuu muun muuassa XSLT muunnoksen avulla. XSLT:n avulla lisätään myös toiminnallisuus sivulle. XML-dokumentin alkuun, ennen muuta sisältöä, tulee lisätä viittaukset versioon ja käytettävään merkistöön. Lisäksi tulee ilmoittaa käytettävän XSLT-skeematiedoston sijainti.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="../xsl/course.xsl"?>
```

Xml-tiedoston rakenne koostuu Course-elementistä, jonka sisällä on <part>-elementtejä, nämä toimivat sivun accordioneina. Accordionit sisältävät kaiken sisällön otsikkoa ja reunassa olevaa menua lukuunottamatta. Accordionien sisältö puolestaan on jaettu <section>-elementtien sisälle. Section tuli HTML5: n mukana ja on hyvä otsikollisen sisällön jaotteluun.

```
<course logo="img/app.svg" title="Orientation to Software Develop-
ment">
<part title="COURSE INFO" icon="off" type="default">
            <section type="info" title="KNOWLEDGE">
                        <p>Welcome to the course Orientation...</p>
            </section>
            <section type="info" title="CHECKPOINT">
            ...sisältöä
            </section>
</part>
```



Kuva 12: Esimerkki xml dokumentin rakenteesta

Hyöty tälläisen staattisen XML-dokumentin käyttämiseen datan lisäämiseen tulee siitä, että käyttämällä samaa XML-rakennetta pohjana voidaan luoda uudelle kurssille sivu, joka näyttää täysin samalta ja omaa samat ominaisuudet. Edellyttäen tietysti, että viittaus samaan XSLT-tiedostoon on lisätty XML-dokumentin metatietoihin. Uusi kurssi voitaisiin helposti tehdä vaikka näin:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="../xsl/course.xsl"?>
<course logo=”kurssin logo” title=”kurssin nimi”>
        <part title=”kappaleen nimi” type=”tyyppi” >
          <section type=”tyyppi” title=”Ensimmäinen osio”>
                <p>pohjustusta aiheeseen…</p>
          </section>
          <section type=”info” title=”Toinen osio”>
                ….
          </section>
        </part>
        <part title=”kappale2” type=”default”>
                …
        </part>
</course>
```

Projektin data muunnetaan XSLT (EXtensible Stylesheet Language Transformations) tyylitiedoston avulla toiseksi XML-dokumentiksi. XSLT:n avulla asetetaan samalla ominaisuudet ja tyylit sivulle. Tämän jälkeen Magic XML puskee muunnetun XML-dokumentin selaimelle ymmärrettävään muotoon automaattisesti.

Koska XSLT on XML:ää, niin se tarvitsee alkuun XML julistuksen. XSLT:n määritykset tulevat kaikki <xsl:stylesheet>-elementin sisään. Stylesheet-elementti vaatii version ja viittaukseen käytettävästä namespacesta, eli tietojen järjestelykirjastosta. Internet Explorer vaatii vielä erikseen määrittelyn siitä, että kumpaa kieltä halutaan selaimelle lähettää (Microsoft Developers).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:output method="html"/> <!-- tarvitaan IE ->
...
</xsl:stylesheet>
```

Tarkastellaan XML osiossa kuvaillun koodin vastaavaa XSLT osaa:

```
<xsl:template match="/course">
<h1>
    <img alt="App" class="hidden-xs">
      <xsl:attribute name="src">
        <xsl:value-of select="@logo"/>
      </xsl:attribute>
    </img>
  <xsl:value-of select="@title"/>
</h1>
<div class="panel-group row" id="accordion">
  <xsl:apply-templates/>
</div>
</xsl:template>

<!—Course elementti XML-tiedostossa-->
<course logo="img/app.svg" title="Orientation to Software Develop-
ment">
```

Esimerkissä määritellään ensin <template match=”/course”>, jolla osoitetaan kohta XML-dokumenttiin mistä halutaan alkaa muokkaamaan puhekieltä. Tässä tapauksessa osoitetaan <course>-elementtiin. Pelkkä kenoviiva <xsl:template match="/ "> osoittaisi XML-dokumentin juureen. Sivulla on h1 elementti, jonka sisällä otsikon edessä on logo ja itse ot-

sikko. Otsikkoon otetaan data course-elementin arvoista. Course-elementissä on omat attribuutit logo ja title. Niiden arvot sijoitetaan XSLT muunnoksen avulla h1-, ja img-elementeille oikeisiin kohtiin. Muunnoksen jälkeen saisimme tälläisen otsikon:

```
<h1 title="Orientation to Software Development>
<img alt="App" class="hidden-xs” src=“img/app.svg"></img>
</h1>
```

Osoite saadaan course-elementiltä src attribuutin arvoksi laittamalla <xsl:value-of-select=”@logo”/> sen sisälle. Value-of-select:llä pystymme valitsemaan elementin yksittäisen arvon. @-merkki nimen edessä valitsee nimenomaan attribuutin. Ilman sitä voidaan valita ankkuritekstiä elementiltä - ”<element>Tämä on ankkuritekstiä</element>”. Huomioi myös kuinka title siirtyy sen isäntäelementille h1, samaan tapaan.

Samaan tapaan voidaan kohdistaa jokaiselle elementille omia asetuksia. Tässä vielä section-elementtien muodostus. Asetukset koskettavat kaikkia section-elementtejä. Esimerkissä lisätään sectionien sisään div-elementti, mille tulee luokka class=”panel panel-@type”, missä type on section elementin samannimisen attribuutin arvo. Panel-heading luokassa sama juttu, mutta haetaan sectionin otsikko sen sisään. Tämän jälkeen kerrotaan, että kaikki section-elementin sisällä olevat lapsielementit laitetaan panel-bodyn sisälle.

```
<xsl:template match="section">
 <div>
     <xsl:attribute name="class">
        panel panel-<xsl:value-of select="@type"/>
     </xsl:attribute>
     <div class="panel-heading">
        <b><xsl:value-of select="@title"/></b>
     </div>
     <div class="panel-body">
        <xsl:apply-templates/>
     </div>
  </div>
</xsl:template>
```

<!—Section elementti XML-tiedostossa -->

```
<section type="info" title="KNOWLEDGE">
    <p>Welcome to the course ...</p>
    ...lisää sisältöä
</section>
```

Tämä muuntaa koodissa kaikki section elementit käyttämään samaa rakennetta:

```
<section class="panel panel-info">
     <div class="panel-heading">
       <b>KNOWLEDGE</b>
     </div>
     <div class="panel-body">
       <p>Welcome to the course ...</p>
       ...lisää sisältöä
     </div>
</section>
```

3.2.2 Magic XML

Magic XML on XSLT-liitännäinen joka käyttää Javascriptiä puskemaan käännetty koodi sivulle ilman vaivaa. Se osaa vetää oikean XML-dokumentin ja käyttää oikeaa XSLT-tiedostoa muunnokseen. Tämän jälkeen MXML puskee sen sivulle käyttäen vain yhtä riviä koodia.

Magic XML otetaan käyttöön hyvin yksinkertaisesti lataamalla tiedostot kotisivulta ja asettamalla projektin sisälle. Tämän jälkeen ladataan scripti sijoittamalla sen HTML koodiin muiden ladattavien Javascript pakettien kanssa.

```
<script type=”text/javascript” src="js/m-xml.js"></script>
```

Kun liitännäinen on tuotu sivulle, niin voidaan määrittää, mikä XML-dokumentti muunnetaan ja mitä XSL-tiedostoa se käyttää. Esimerkissä sijoitetaan tiedot div-elementin sisälle. Data-xml osoittaa, mistä löytyy muunnettava XML-tiedosto, ja data-xslt puolestaan missä on muunnokseen käytettävä XSL-tiedosto.

```
<div data-xml="xml/course.xml" data-xslt="xsl/course.xsl"></div>
```

Joudumme vielä kirjoittamaan Javascript tiedostoon, että haluamme MagicXML:n aloittavan kääntämään sivua.

```
//course xslt
magicXML.parse();
```

Voidaan myös tarkentaa erikseen Magic XML:lle mitkä elementit haluamme parsittavan. Projektissa tehtävät käyttävät omaa XSL pohjaansa.

```
//all class=”exercise” xslt
magicXML.parse(".exercise");
```

3.3 Responsiivisuuden toteuttaminen

3.3.1 Media kyselyt

Projektiin luotiin 6 mediakyselyä, kaikki eri leveyksille. Projektista haluttiin tehdä mahdollisimman muokattavissa oleva. Tarkoitus oli siis pienentää muokattavia paloja, jotta voidaan kohdentaa asetukset laajasti eri ruudun kokoisille näytöille. Mediakyselyjä voisi lisätä sille leveys-skaalalle mitä tarvitaan. Voitaisiin tehdä vaikka omat asetukset alykellon kokoiselle näytölle, joskin tuskin kukaan sivua sellaisella selailee.

Koska Bootstrapissa on vakiona mediakysely vain alle 768px näytöille, ja monilla ihmisistä on myös esimerkiksi 480px näyttöisiä puhelimia, niin on hyvä tehdä breakpointit pienemmille laitteille, jotta voidaan kohdistaa asetuksia suoraan niihin CSS-tiedostossa. Mediakyselyt tehtiin 480px näytöille ja 320px näytöille. Niiden pitäisi kattaa pienemmätkin puhelimet.

```
  /* Custom fixes for XXXS Devices, custom, Phones */

@media only screen and (max-width : 320px){

}

/* Custom fixes for XXS Devices, custom, Phones */

@media only screen and (max-width : 480px){

}
```

CSS-tiedostoon otettiin ne media kyselyt vakioiksi, mitkä vastaavat Bootstrapin eri koko luokkia xs, sm, md ja lg. Tämä sen takia, että tyylit vaihtuvat samassa pisteessä kun asetetut Bootstrapin sarakkeet vaihtuvat. Bootstrap käyttää siis jo näitä mediakyselyjä valmiiksi. Varmistutaan vain, että tehdään oikeaan leveysväliin muutoksia.

```
/* Extra small devices (phones, less than 768px) */
...asetuksia mobiilille...
/* Small Devices, .visible-sm-* */

@media (min-width: 768px) {...asetuksia tableteille...}

/* Medium Devices, .visible-md-* */

@media (min-width: 992px) {...asetuksia kannettaville... }

/* Large Devices, .visible-lg-* */

@media (min-width: 1200px) {...aetuksia pöytäkoneille ja isommille... }
```

3.3.2 Juokseva ruudukko

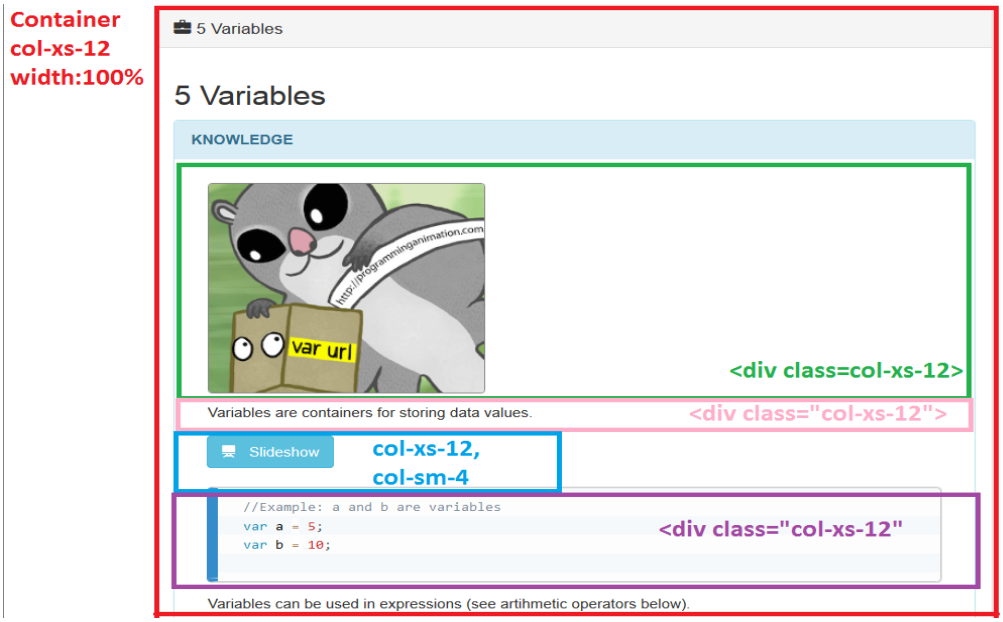
Sivun sisältö jaetaan ruudukon sisälle, joka joustaa sivun pienentyessä tai suurentuessa. Joustavan ruudukon voisi tehdä myös itse CSS:llä, mutta koska bootstrap on mukana projektissa, niin käytämme sen mukana tullutta responsiivista ruudukkosysteemiä. On olemassa myös muita liitännäisiä, jotka tuovat pelkän ruudukon sivustoon. Yksi hyvä on Cute Grids.

Bootstrapin ruudukot toimivat niin, että data jaetaan riveihin jotka ovat 12 saraketta leveitä. Tämän jälkeen määritellään sarakkeita elementeille sillä perusteella, kuinka monta saraketta halutaan sen riviltä vievän. Esimerkkinä luokka ”col-xs-12 col-sm-6” elementissä merkkaisi sen olevan xs-koossa koko rivin mittainen, ja sm-koossa puolikkaan rivin mittainen. Koska rivien pituudet periytyvät myös ylöspäin, niin myös md- ja lg-koot saavat 6 saraketta leveydeksi. Jos käy niin, että rivin sarakkeiden yhteismäärä ylittää 12:sta, niin Bootstrap pinoaa automaattisesti ylimenevän elementin.

Ruudukkoa testattaessa kannattaa tehdä sivun yläreunaan teksti, joka kertoo mikä koko on kyseessä sillä resoluutiolla. Sen saa yksinkertaisimmin Bootstrapin visible luokalla. Saman voi tehdä myös määrittämällä CSS:ään display:none ja display:block määritykset jokaiselle mediakyselylle erikseen, jos Bootstrap ei nappaa.

```
<span class="visible-xs">SIZE XS</span>
<span class="visible-sm">SIZE SM</span>
<span class="visible-md">SIZE MD</span>
<span class="visible-lg">SIZE LG</span>
```

Toteutuksessa käytettiin ruudukkoa siten hyväksi, että kaikki sisältö accordion-paneelien sisällä varaa itselleen koko rivin verran tilaa, eli col-xs-12. XSLT-tiedostossa otetaan kaikki div-elementit paneelin sisältä ja lisätään kyseinen luokka niille. Tällöin sisältö pysyy nätisti allekkain paneelien sisällä. Poikkeus ruudukoissa on nappuloiden kohdalla. painikkeet varaavat koko rivin tilan vain xs-koosa. Sm-koosta ylöspäin ne varaavat itselleen 4 saraketta, jolloin se tarkoittaisi, että nappuloita mahtuu riville 3 vierekkäin 4+4+4=12.



Kuva 13 - Bootstrapin ruudukko sivulla

3.3.3 Kuvat ja elementtien piiloittaminen

Vaikka bootstrapissä olisikin valmiina luokka ”.img-responsive”, niin tehdään oma luokka CSS-tiedostoon, joka tekee juuri saman kuin Bootstrapin luokka.

```
p img{
            max-width:100%;
            height:auto;
            display:block;
            /*  margin: 0 auto;*/ /* Centers images */
}
```

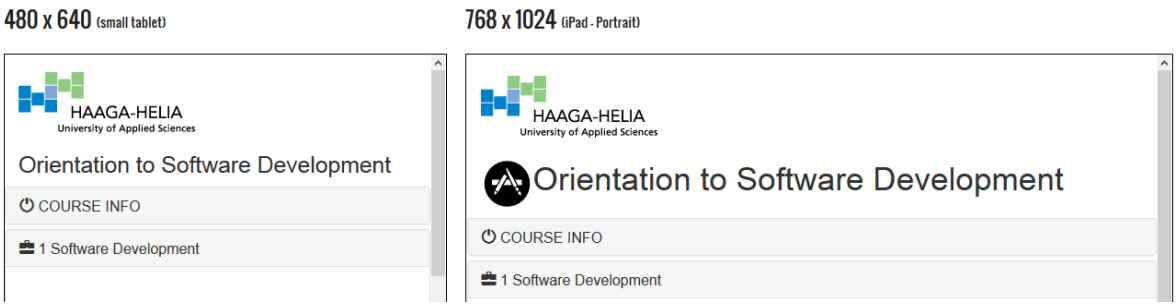
Näin kaikki p-elementin sisällä olevat kuvat ovat aina maksimissaan 100 prosenttia leveitä suhteessa isäntäelementtiinsä. Korkeus on automaattinen, jotta se skaalautuu leveyden mukana, eikä vääristä kuvasuhdetta. Display block määrittää, että kuva näytetään yhtenä lohkona. Uloskommentoitu ”margin: 0 auto;” taas määrittää marginaalin kuvalle siten, että kuva on aina keskellä.

Kuva 14 - Kuvien skaalautuminen

Sivulla käytetään paljon ikoneita miellyttävämmän visuaalisen ilmeen luomiseksi. Kuitenkin pienillä mobiilinäytöillä ikonit vievät paljon tilaa esimerkiksi nappuloiden pitkiltä otsikoilta. Bootstrapissa on visible-, ja hidden-luokat, joilla voidaan piiloittaa helposti sisältöä mediakyselyihin viitaten. Visible-luokka lisää elementille ”display:block; “, kun taas hidden-luokka lisää elementille “visible:none;”. Luokkia kohdistetaan samoin kun columneja: ”visible-xs”, ”visible-sm”, ”visible-md” ja ”visible-lg”. Luokkia käytettäessä tulee huomioida, että esimerkiksi käyttämällä hidden-sm luokkaa, elementti on piiloitettu vain small leveydellä ja on näkyvissä sekä xs-, md- ja lg-luokilla.

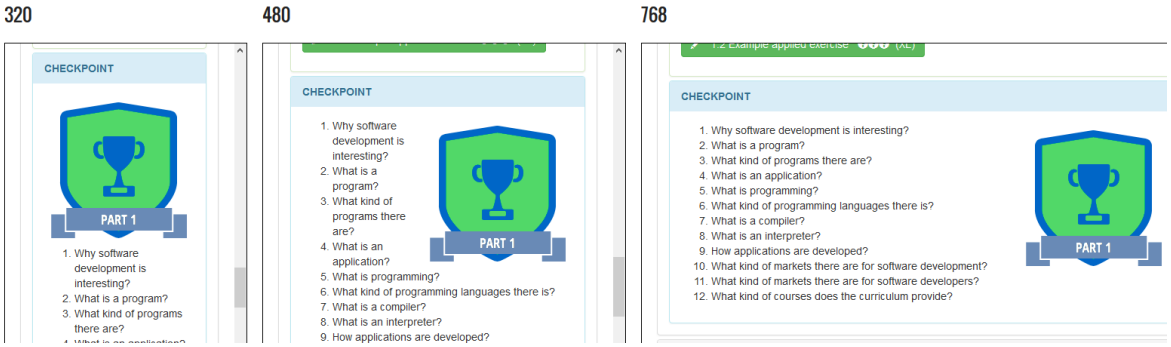
Piilotetaan logo hidden-xs luokan avulla, eli alle 768 pikseliä leveiltä ruuduilta.

```
<xsl:template match="/course">

<h1><img alt="App" class="hidden-xs"><xsl:attribute name="src"><xsl:value-of
select="@logo"/></xsl:attribute></img><xsl:value-of select="@title"/></h1>

</xsl:template>
```



Kuva 15 - Logo hidden ja visible

Esimerkin voi tehdä siis puhtaasti CSS:llä suhteellisen pienellä vaivalla. Projektissa on badgeicon, jossa jouduttiin tekemään omat hidden luokat. Hidden luokat tehtiin koska haluttiin vaihtaa badgen paikkaa vasta kun ruudunleveys alittaa 480 pikseliä. Kuvassa näkyy checkpoint osio, jonka tarkoitus on antaa lyhyt kertaus kappaleesta, ja badge sen osion suorituksesta. Badge kuitenkin sitoutuu tekstin ympärille ja rikkoo listaa. Badge on myös liian suuri, joten sillekkin tulee lisätä responsiivinen skaalaus.



Kuva 16 - Badgeicon alkutilanne

Parhaiten saamme kaikki badget tehtyä samaan aikaan XSLT muunnoksella XML-dokumenttiin. Tehdään XML-dokumenttiin oma elementti, jotta img-elementtien asetukset eivät ylikirjoita näitä, tai toisinpäin.

```
<badgeImg><img src="img/badge00.png" alt="Badge 0"
class="badgeicon"/></badgeImg>
```

Tämän jälkeen voimme tarttua badgeImg-elementtiin XSL-dokumentissa:

```
<!-- Responsive badgeicon elements and ol- list -->
<xsl:template match="badgeImg">
<div class="img-responsive col-xs-3 pull-right hide-xs">
          <xsl:copy-of select="."/>
</div>

<div class="img-responsive col-xs-12 pull-right show-xs">
          <xsl:copy-of select="."/>
</div>
```

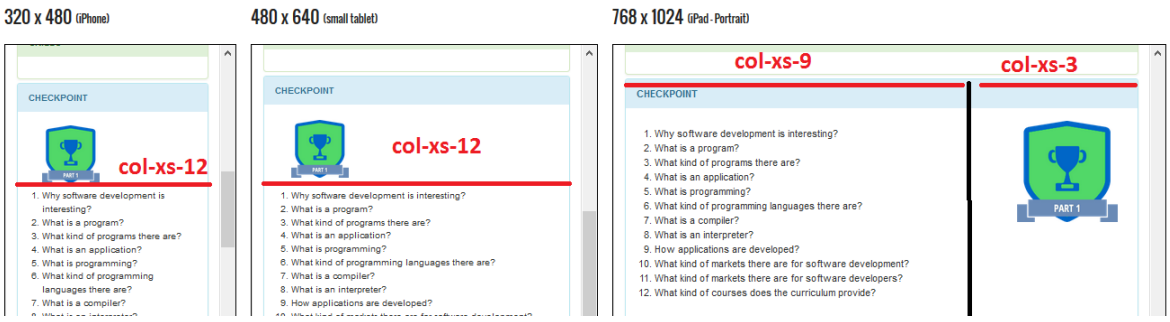
Ensin otetaan vastaava elementti template match:llä. Lisätään badge käyttämään 3 saraketta 12:sta. Lisätään myös Bootstrapin oma pull-right luokka, joka lisää ”float:right” elementille. Käytetään tässä tilanteessa Bootstrapin omaa img-responsive luokkaa. <xsl:copy-of-select=”.”> kopioi kaiken badgeImg-elementin sisällä olevan tiedon div:n sisään.

Lisätään vastaavat hide-xs ja show-xs luokat css:ään. Niillä piiloitetaan pieni kuva ja lista alle 480 pikseliä leveiltä ruuduilta ja näytetään pieni kuva ja lista vierekkäin isoilla ruuduilla. Displaytä määrittäessä tulee muistaa, että ”display:hidden;” lataa elementit silti valmiiksi taustalle, joka saattaa hidastaa sivua turhaan. ”Display:none;” ei lataa ollenkaan elementtejä valmiiksi.

```
/* Universal styles Extra small devices (phones, less than
768px)*/
/* No media query since this is the default in Bootstrap */

.hide-xs{
display:inline-block; /* inline block antaa lohkon olla sa-
malla rivillä */
}
.show-xs{
display:none;
}

/* Custom fixes for XXS Devices, custom, Phones (portrait and
landscape)*/
@media only screen and (max-width : 480px){
.hide-xs{
display:none;
}
.show-xs{
display:inline-block; /* inline block antaa lohkon olla sa-
malla rivillä */
}

}
```

Joudutaan myös muuttamaan listan vieressä vastaamaan vapaana olevaa tilaa:

```
<xsl:template match="ol">
            <div class="col-xs-9  hide-xs">
            <xsl:copy-of select="."/>
            </div>

            <div class="col-xs-12 show-xs">
            <xsl:copy-of select="."/>
            </div>
</xsl:template>
```



Kuva 17 - Badgeicon lopputilanne

3.3.4 Responsive video

Responsiivisen videon saaminen ei ole hirveän vaikeaa omalle sivulleen. On olemassa muutamia yleisesti käytettyjä kikkoja videoiden skaalautumiseen oikein. Suurin ongelma videoiden kanssa on videoiden kuvasuhde. Yksi tapa on kuitenkin laskea videoille itse yksi kuva suhde tai sitten käyttää eri määrityksiä eri kuvasuhteille.

Yksi tapa on asettaa videoille säiliö joka on suhteellisessa paikassa. Sen ei anneta vuotaa yli ja se asetetaan alkamaan korkeudelta 0. Tälle säiliölle lasketaan padding-bottomin avulla haluttu kuvasuhde. 4:3 kuvasuhde olisi 75% ja 16:9 kuvasuhde olisi 56.25%. Suhde lasketaan kaavalla korkeus/leveys*100 (basicuse).

```
.video-responsive {
  position: relative;
  padding-bottom: 75%; /*Aspect ratio: 4:3 -> 3 / 4 * 100 = 75%, for 16:9
aspect ratio -> 9 / 16 * 100 = 56.25% */
  height: 0;
  overflow: hidden;
}
```

Säiliön sisälle asetetaan video-objekti joka tulee olla absoluuttisella sijainnilla ja alkaa paikasta ”top:0” ja ”left:0”, eli ylävasemmalta. Asetetaan vielä video olemaan 100 prosenttia leveä ja korkea, jotta se on koko säiliön kokoinen vaikka säiliö pienenisikin.

```
.video-responsive iframe,
.video-responsive object,
.video-responsive embed,
.video-responsive video {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```

Tämä tapa on helppo ja yksinkertainen mutta tukee vain yhtä kuvasuhdetta. Toki muutkin videon toimivat ihan hyvin näillä asetuksilla, mutta reunoille voi jäädä ylimääräistä mustaa tilaa eri kuvasuhteen videoilla. Toki voisi tehdä kaikille kuvasuhteille omat säiliöasetukset tai rakentaa skripti, joka laskee kuvasuhteen jokaiselle videolle erikseen.

3.3.5 Fontit

Fontit tehdään em arvoilla, jotka ovat suhteellisia kokoja verrattuna oletusfonttiin. Asetetaan ensin body-elementille fontin kooksi 10px, joka toimii pohjana. Oletuksena selainten perusfontti on yleensä aina 16px. Halutaan selkeyttä fonttien asettamista asettamalla fontiksi 10px, koska silloin esimerkiksi 1.2em = 12px. Tämän jälkeen voimme asettaa otsikoille, p-elementeille ja listalle omat fontti em-arvot. Muuttamalla pohja fonttia saadaan

muutettua sivun kaikkien fonttien kokoa. Ei ole pakko muuttaa pohja fonttia, vaan em arvot voidaan laskea myös suoraan selaimen oletusfontista, joka on 16px. (W3C.)

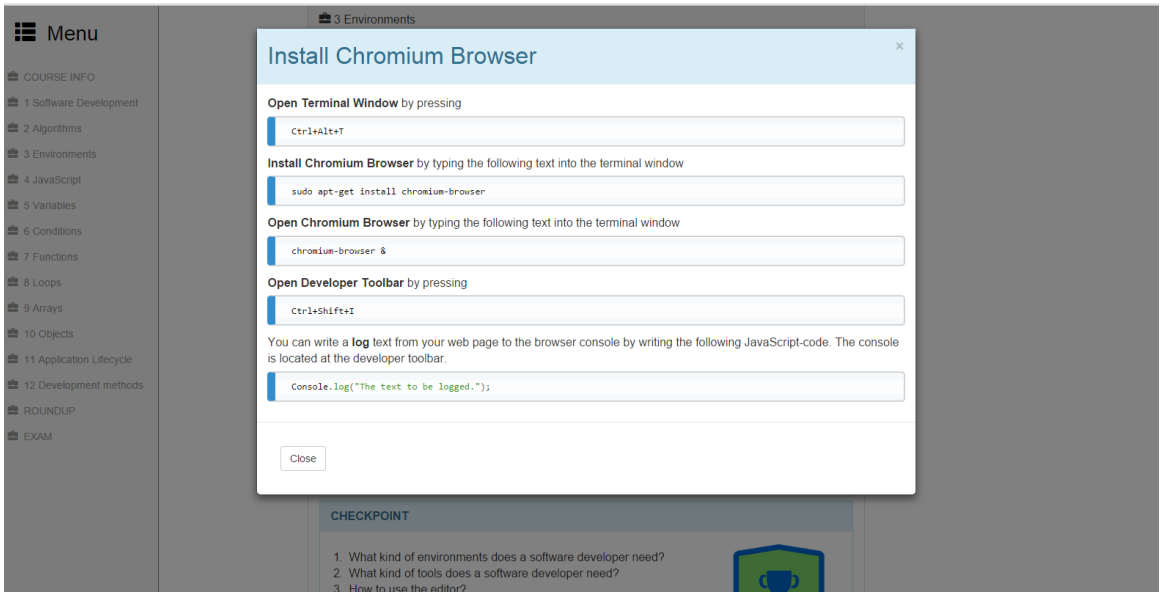
Tätä samaa periaatetta voidaan käyttää myös muun sisällön muuttamiseksi suhteellisiksi leveyksiksi. Esimerkiksi 120px leveän elementin suhteellinen leveys olisi, jos sen isäntäelementti on 768px, 120px/768px = 15.265 %.

	Alle 320px	Alle 480px	Alle 768px	Alle 992px	Alle 1200px	Yli 1200px
H1	1.8em	1.9em	2em	2.2em	2.2em	2.2em
H2,h3	1.6em	1.7em	1.8em	2em	2em	2em
Sisältö-teksti	1.2em	1.3em	1.5em	1.6em	1.6em	1.6em
Basefont	10px (10 * x = actual font)					

3.4 Graafiset ohjauselementit

3.4.1 Modal-ikkunat

Projektissa käytetään modaaleja lisäsisällön näyttämiseen. Modaaleihin voidaan asettaa mitä tahansa sisältöä. Bootstrapin modaalit toimivat kirjoittamalla painike, joka kutsuu modaalia ja itse modaalin sisältöä. Modaali aukeaa kutsumalla sen tunnusta HTML-koodista. Modaalin voi puolestaan sulkea lisäämällä modaalin sisällä painettavalle elementille data-dismiss=”modal”.



Kuva 18 - Avattu modaali ikkuna

<!—Laukaise modaali painikkeella -->

```
<button type="button" class="btn btn-info col-xs-12 col-sm-4"
data-toggle="modal" data-target="#{generate-id()}">
            <xsl:value-of select="@title"/>
</button>
```

<!— modal -->

```
<div class="modal fade" tabindex="-1" role="dialog"
aria-hidden="true" id="{generate-id()}"
aria-labelledby="{generate-id()}">

 <div class="modal-dialog">
<!— modal content -->
   <div class="modal-content">
<!—modal header -->
     <div class="modal-header alert-info">
          <button type="button" class="close" data-dis-
          miss="modal" aria-hidden="true">&#215;
          </button>
       <h4 class="modal-title">
          <xsl:value-of select="@title"/>
<!-- Otsikko generoidaan annetusta titlestä-->
       </h4>
     </div>
<!—modal body -->
     <div class="modal-body">
       <p>
          Modaalin sisältöä…
       </p>
     </div>
<!—modal footer -->
      <div class="modal-footer">
     <button type="button" class="btn btn-default" data-dis-
miss="modal">Close</button>
     </div>
   </div>
 </div>
</div>
```

Modaalit toimivat siis ensin kirjoittamalla painike, joka avaa modaalin. Buttonille annetaan arvot data-toggle=”modal” ja data-target=”id”. Data-toggle=”modal” avaa modaalin ja data-target tarkentaa mikä modaali avataan antamalla arvoksi modaalin id. Esimerkissä generoidaan id käyttämällä XSLT:n omaa generate.id() funktiota.

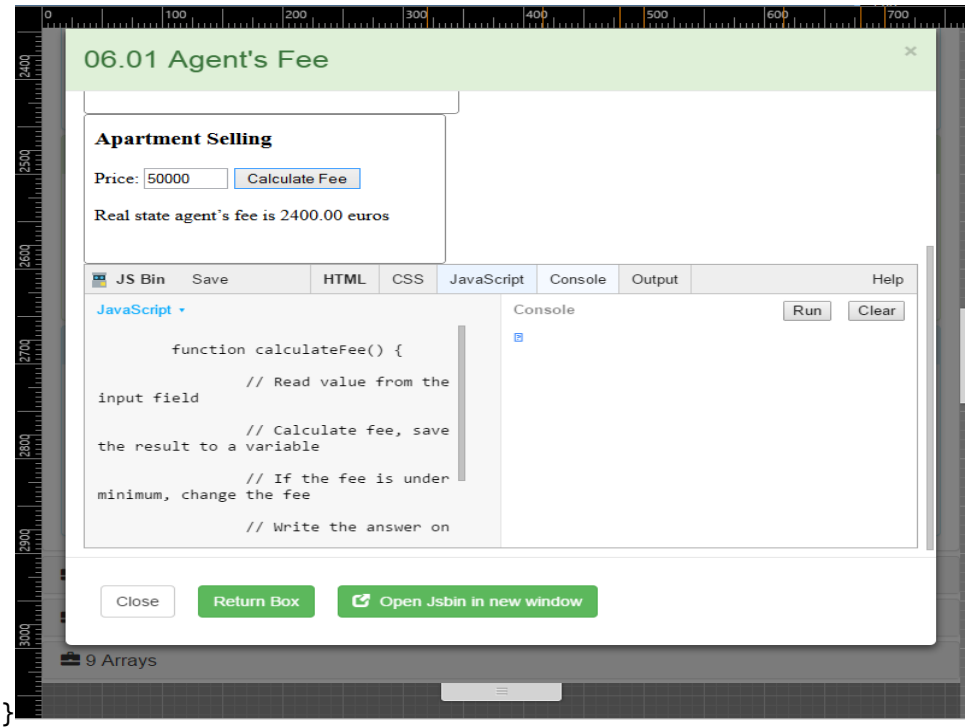
Tämän jälkeen kirjoitetaan itse modaali. Modaali määritetään antamalla diville luokka ”modal fade”, jossa modal määrittää divin olevan modaali ja fade lisää siirtymä efektin modaaliin, eli modaali ilmestyy ja piiloutuu sulavasti. Role=”dialog” auttaa selainta ymmärtämään, että tiedot dialogin sisällä ovat ryhmitettyjä ja erotettuja muusta sivun sisällöstä. Modaalit tulee myös olla kunnolla otsikoitu (label). Modal-dialog luokka asettaa modaalin leveyden ja korkeuden oikein.

Modal header osion sisään tulee siis modaalin otsikko. Otsikko tehdään modal-header luokan sisään. Button class=”close” muotoilee ruksin painikkeeksi ja data-dismiss=”modal” sulkee modaalin painettaessa. H4 elementin sisään otetaan otsikko (title) XML dokumentista.

Modal body osassa määritetään modaalin sisälle tuleva sisältö. Modal footer osiossa taas modaalin ala-osa, johon lisätään myös toinen ”Close”-painike modaalin sulkemista varten.

Tehtävä modaalit haluttiin leveämmiksi niiden sisältämän editorin käytettävyyden helpottamiseksi. Tähän tehtiin oma modal-wide luokka joka vie 95 prosenttia käytettävissä olevasta tilasta, kun ruudun koko on pienempi kuin 769 pikseliä.

```
/* Universal styles Extra small devices (phones, less than 768px)*/
/* No media query since this is the default in Bootstrap */
.modal.modal-wide .modal-dialog {
  width: 95%;                    /* modaali vie 95% tilaa leveydeltään*/
  margin: 0 auto; /*modaalin keskitys*/
}
.modal-wide .modal-body {
  overflow-y: auto; /* pysty-vierityspalkki jos sisältö ei mahdu */
```



Kuva 19 - Modal wide alle 769 pikseliä

Modaalille asetetaan myös maksimileveys max-width, jotta se ei suurene loputtomiin kun näytönkoko suurenee. Otetaan myös yli 768 pikseliä leveiltä näytöiltä leveyttä pois, koska modaalista kasvaa muuten liian leveä nopeaan selailuun.

```
/* Small Devices, .visible-sm-* */

@media (min-width: 768px) {

.modal.modal-wide .modal-dialog {
            width: 75%; /* modal takes 75% of page-width until max-width; */
            margin: 0 auto;
            max-width:1200px; }}
```

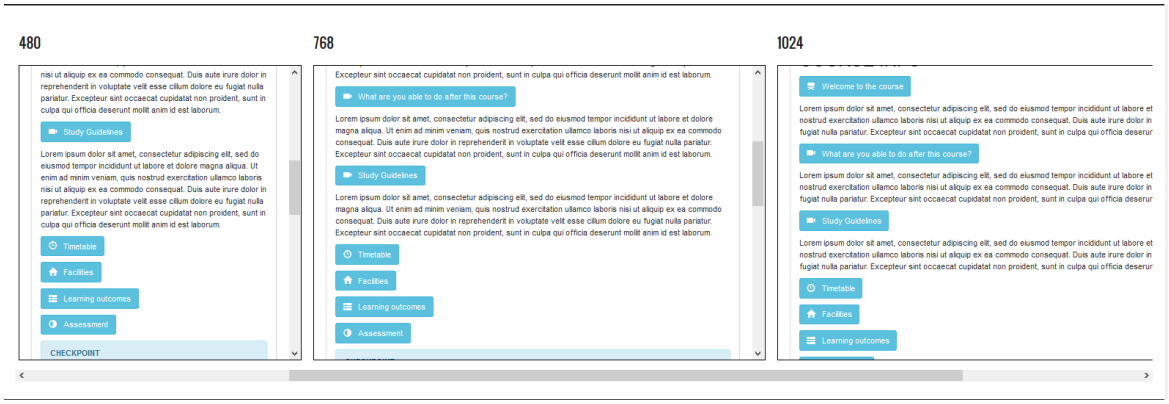
Leveitä modaaleja voidaan käyttää lisäämällä modaali elementille luokka ”modal-wide”.

```
<div class="modal modal-wide"
```

Leveistä modaaleista löytyy lisäkuvia liite-osasta dokumenttia.

3.4.2 Painikkeet

Painikkeille laitettiin määrityksiä, sekä bootstrapin omilla luokilla, että puhtaasti CSS:llä. Bootstrapin ruudukkosysteemillä osoitettiin painikkeille käytettävissä oleva tila ja CSS määrittää niiden leveyden. Tehtävä painikkeet laitettiin puoliksi ”skills” osion sisälle täyttämään kokonaan niille annettu tila. Muut painikkeet laitettiin määrittämään leveytensä automaattisesti ja maksimissaan mahtumaan kolme vierekkäin.



Kuva 20 - Yleiset painikkeet alkutilanne

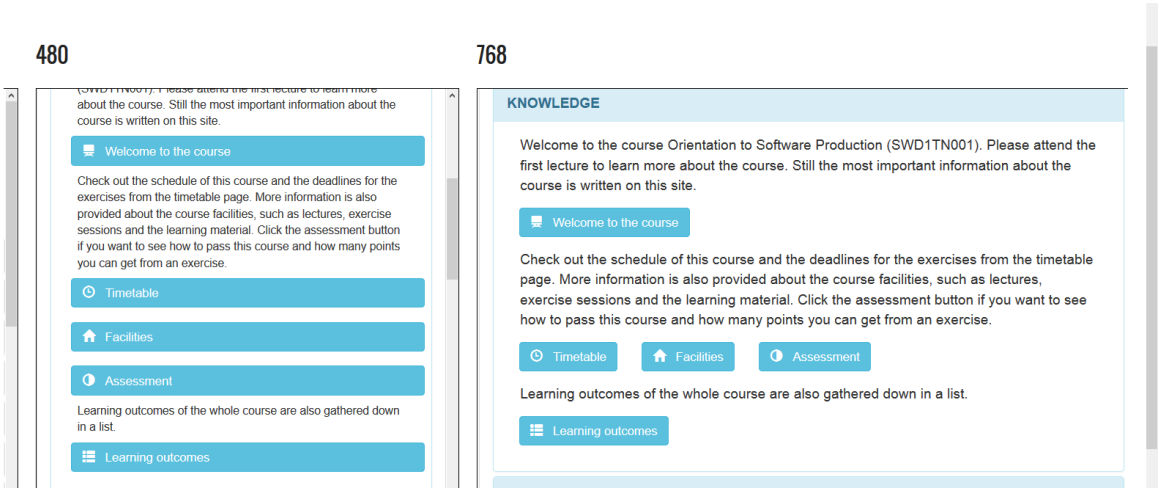
Kuvassa näkyy yleisten painikkeiden alkutilanne. Painikkeet vievät turhaa tilaa sivulta kun ne ovat pinottuina. Optimi tilanne olisi se, että pienimmillä ruuduilla painikkeet ovat koko ruudun leveyksisiä, jotta niiden painaminen on mahdollisimman vaivatonta ja suurilla ruuduilla ne olisivat vierekkäin. Yleisille painikkeille annettiin luokat col-xs-12 ja col-sm-4, jolloin pienillä näytöillä painikkeet olisivat päällekkäin ja varaisivat koko rivin tilan. Small ruuduilla ne olisi 4+4+4=12, eli kolme vierekkäin (`class="btn btn-info col-xs-12 col-sm-4)`.

CSS:ään pitää tehdä määritykset yleisille painikkeet asettamalla maksimileveys 100 prosenttiin ja teksti alkamaan oletuksena vasemmalta oikealle. Maksimileveyden asettaminen 100 prosenttiin tarkoittaa käytännössä, että elementti voi olla maksimissaan 100 prosenttia leveä suhteessa siihen elementtiin, minkä sisällä se on. Tällöin se pysyy sille asetetun tilan sisällä, vaikka ruutua suurentelisi tai pienentelisikin. Leveyden asettaminen automaattiseksi asettaa painikkeen leveyden sen sisältämän tekstin määrän mukaan. Exercise modaalin sisällä olevat painikkeet laitetaan käyttämään samoja asetuksia, jotta seuraavaksi tehtävät tehtävä painikkeiden asetukset eivät tee niistä liian suuria.

```
/* Universal styles Extra small devices (phones, less than 768px)*/

.btn,
.exercise .modal .btn{
            max-width:100%;
            width:auto;
            text-align:left;
```

```
    margin:2%;
}
```

Kun CSS määritykset ovat lisätty, tulisi ”.btn”- ja ”.exercise.modal.btn”-luokkien painikkeiden käyttäytyä seuraavanlaisesti. Kuvassa xs- ja sm-koot.



Kuva 21 - Muut painikkeet lopputilanne xs-, ja sm-koot

Tehtävä painikkeet eivät skaalaudu hyvin ja teksti jää piiloon kun ruutua pienentää. Voitaisiin käyttää koko tila hyödyksi asettamalla painikkeet puoliksi isoilla ruuduilla laatikon sisään.

Kuva 22 - Tehtävä buttonit alkutilanne

Tehtävä painikkeet asetellaan puoliksi laatikon sisään käyttämällä Bootstrapin sarakkeita hyväksi. Asetetaan painikkeille sarakkeet `class="btn col-xs-12 col-sm-6",` jolloin painikkeet ovat päällekkäin ja koko rivin mittaisia, alle 768 pikseliä leveiltä selaimilta. Yli 768 pikseliä leveillä selaimilla painikkeita mahtuu kaksi vierekkäin 6+6=12. Jos teksti on kuitenkin niin pitkä, ettei se mahtuisi hyvin ruutuun. Silloin bootstrap pinoaa painikkeet automaattisesti. Kumotaan pinoaminen kuitenkin asettamalla painikkeiden teksi niin, että se saa mennä yli ja lisätään kolme pistettä ylimenevän osan tilalle CSS:llä. Tämän jälkeen

voitaisiin lisätä Jqueryllä Bootstrapin tooltip-ominaisuus kaikille painikkeille. Se on tietolaatikko mikä ilmestyy kun hiiren vie painikkeen päälle. Tähän laatikkoon voisimme laittaa painikkeiden kokotekstin.

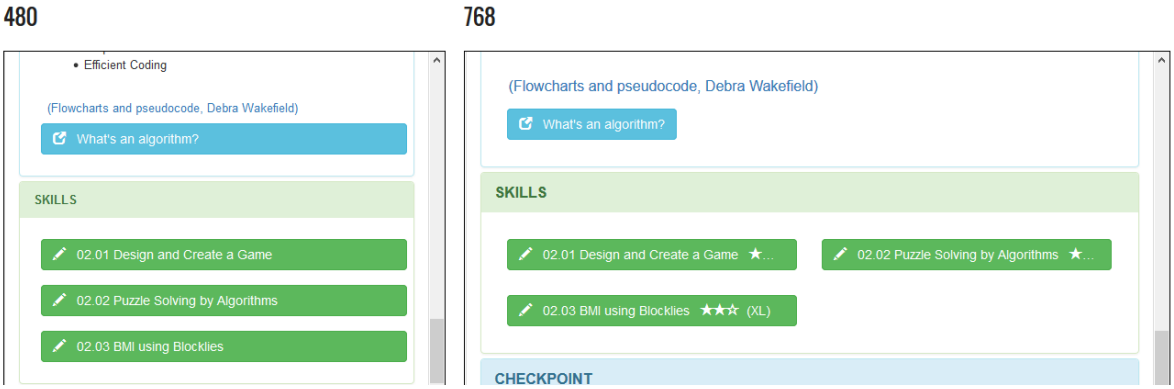
Tooltipit saadaan päälle hyödyntämällä mouseover-eventiä. Eli kun hiiri menee ”.btn” luokan päälle lasketaan onko elementin leveys pienempi kuin vieritysleveys ja otsikko. Tooltip pitää aktivoida antamalla tooltip halutulle elementille. Voidaan määrittää myös tooltipille mitä tekstiä näytetään ja mihin tietolaatikko ilmestyy. Määritetään vielä, että näytetään tooltip kun funktio käynnistyy, eli hiiri menee painikkeen päälle.

```
//Enable BS tooltips on overflowing buttons
    $(document).on('mouseover', ".btn", function() {
        var $this = $(this);
        if(this.offsetWidth < this.scrollWidth && !$this.attr('title')) {
            //add title to tooltip
             $this.tooltip({
                  title: $this.text(),
                  placement: "top"
             });
             $this.tooltip('show');
        }
   });
```

Lisätään tehtävä painikkeille seuraavaksi yleiset tyylit. Tehtävä painikkeille laitetaan 46 prosenttia leveydeksi, joka on noin puolet säilion leveydestä, jos ottaa mukaan painikkeiden 2 prosentin marginaalit. Tällöin painikkeet ovat aina sopivan kokoisia olemaan puoliksi säiliössä. ”White-space:nowrap;” kertoo, ettei tekstiä saa paketoida kahdelle riville tai mitenkään muutenkaan. Overflow hidden varmistaa, ettei vierityspalkkia tule painikkeille ja ellipsis määrittää tekstin perään kolme pistettä, jos teksti ei mahdu painikkeeseen. Toinen vaihtoehto olisi antaa ”text-overflow:clip;”, joka katkaisee tekstin ylimenevästä kohdasta jättämättä pisteitä perään.

```
.exercise .btn{
          white-space:nowrap;
          overflow:hidden;
          text-overflow: ellipsis !important;
          width:46%;
          }
```

Tehtävä nappuloiden lopussa olevat tähti-ikonit ja vaikeusaste piiloitetaan käyttämällä samaa luokkaa mikä tehtiin kuvat ja elementtien piiloittaminen kohdassa badgeiconille, eli käytännössä annetaan display:none ja display:inline-block riippuen leveydestä. Inline-block sen takia, että ikonit tulevat samalle riville.

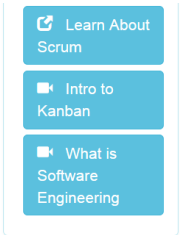


Kuva 23 - Tehtäväbuttonit lopputilanne

Mobiilinäytöillä olisi hyvä jos kaikki painikkeet olisi aina koko isäntäelementin levyisiä. Painikkeet mahtuvat aivan hyvin olemaan normaalikoossaan välillä 480px-768px. Laitetaan alle 480px näytöille minimi leveys 100 prosenttiin, jolloin ”.btn” luokan on pakko olla aina 100 prosenttia leveä suhteessa elementtiin, minkä sisällä painikkeet ovat.

```
@media only screen and (max-width : 480px){
            .btn{
                        white-space: normal; !important;
                        min-width: 100%;
            }
}
```

Kaikille painikkeille, poislukien tehtäväpainikkeet, jotka ovat alle 480 pikseliä näytöillä, asetetaan ”whitespace: normal important!”. Tämä antaa nappuloiden tekstin jakautua kahdelle riville, jos se ei mahdu niiden sisään. Tätä ei tarvita suurempiin näyttöihin koska halutaan mielummin pinota painikkeet, kuin mahdollisesti pinota tekstiä.



Kuva 24 - Whitespace:normal; jakaa tekstin useampaan riviin

3.4.3 Navigointi

Sivulle haluttiin jonkin näköinen navigointi isoille näyttöpäätteille. Sillä helpotetaan käyttäjän selailua sivulla siten, ettei käyttäjän tarvitse aina vierittää sivua ylös tai alaspäin löytääkseen seuraavan avattavan osion. Käyttäjä voi avata suoraan keskeltä osiota minkä tahansa accordionin.

Menu toimii etsimällä muun sisällön ulkopuolelta kaikki accordionit, eli ne mitä halutaan avata menun linkeistä. Tehdäänkin XSLT muunnoksella dokumentin juureen sidebar, joka käyttää visible-lg luokkaa, jolloin menu ei näy kuin isoimmilla ruuduilla. Menu on rakenteeltaan järjestyksetön lista (ul) ja käyttää lista-elementtejä (li). Menu hakee XML-dokumentista kaikki part elementit ja käy ne läpi yksitellen <xsl:for-each>-silmukalla. Koska lista-elementit ovat for-each silmukan sisällä, tulee lista-objekteja tasan niin monta kun part-elementtejä on tehty.

Silmukan sisällä tulostamme part-elementin otsikon. Annamme myös data-toggle=”collapse” ja href=”#collapse”<xsl:number>, sekä kerromme, että tämän collapsen isäntä on sama kuin avattavan.

```
<xsl:template match="/course">

...
<!-- Sidebar -->
 <div id="sidebar" class="visible-lg">
   <ul class="sidebar-nav">
      <li class="sidebar-brand">
          <h3>
           MENU TEKSTI + IKONI
          </h3>
      </li>

<!—Käydään kaikki part elementit läpi xsl:for-each loopilla -->
      <xsl:for-each select="part">
      <li>
<!—Avataan luokka collapse, id:llä “collapse<xsl:number>” joka toimii
myös isäntäelementtinä -->
      <a data-toggle="collapse">
           <xsl:attribute name="href">#collapse
           <xsl:number/></xsl:attribute>
           <xsl:attribute name="data-parent">#collapse
           <xsl:number/></xsl:attribute>
<!—piiloitetaan otsikko pieniltä näytöiltä -->
           <span class="hide-xs hidden-sm">
                 <!—haetaan otsikko-->
```

```
                <xsl:value-of select="@title"/>
            </span>
        </a>
        </li>
        </xsl:for-each> <!—end of for-each -->
    </ul>  <!—end of sidebar-nav -->
  </div>   <!—end of #sidebar -->
</xsl:template>
```

Nyt meillä on navigoinnin rakenne valmis. Joudumme kuitenkin antamaan tyylit navigoinnille. Joudumme tekemään menun pohjan, joka on aina koko ruudun kokoinen. Menu tulee myö pysyä paikallaan kun sivua selailee.

Menun pohja luodaan seuraavasti:

```
/* wrapper for menu elements */
#sidebar{
    height:100%;
    position:fixed; /* stays still when scrolling */
    top:0;
    left:0;
    background-color: #F5F5F5;
    border-right: 1px solid #808080;
    overflow-y:auto;
    z-index:1;
}
```

Menun pohjalle laitetaan korkeus 100 prosenttia, jolloin se on aina ruudun korkuinen. Position: fixed; asettaa pohjan olemaan paikoillaan vaikka muu sivu liikkuisikin. Top:0; ja left:0; kertovat mistä kohdasta pohja alkaa, eli tässä tapauksessa ylävasemmalta kohdasta 0,0. Overflow-y: auto asettaa vierityspalkin y-akselin suuntaisesti jos sisältö ei mahdu pohjalle.

Seuraavaksi tarvitaan pohja linkeille. Ei tarvitse muuta kun ottaa ”bullet”-pallot listan linkkien edestä pois.

```
/* wrapper for menu buttons */
.sidebar-nav {
    list-style: none;
}
```

Asetetaan linkit olemaan 100 prosenttia leveitä ja erotetaan ne toisistaan line-height:llä. Annetaan marginaalia vierityspalkille, joka ilmestyy jos linkit eivät mahdu menuun. Lisätään vielä linkeille värit ja eri värit silloin kun hiiren vie linkin päälle (a:hover).

```
/* menu buttons */
.sidebar-nav li {
    width:100%;
    font-size:14px;
    line-height: 35px;
    padding-left:5px;
    margin-right:20px; /*padding for scrollbar*/
}
```

```
.sidebar-nav li a {
    display: block;
    text-decoration: none;
    color: #999999;
}

.sidebar-nav li a:focus,
.sidebar-nav li a:hover{
            color: #000000;
}
```

Menu avaa nyt hyvin accordioneja, mutta jättää edelliset auki. Ratkaisuksi tein yksinkertaisen Jquery funktion, joka odottelee navigoinnin linkin klikkausta ja katsoo sitten onko aukinaisia accordioneja ja sulkee ne.

```
// Close all open accordions when sidemenu is clicked. -> href opens next
accordion.
            $('#sidebar li a').click(function() {
                        $('.panel-collapse.in').collapse('hide');
            });
```

Menun ja accordionien yhteiskäyttö jätti joitain accordioneja auki. Ongelma ratkesi sillä, että kuunnellaan Bootstrapin tapahtumaa 'show.bs.collapse'. Tapahtuma alkaa silloin kun sivu on avaamassa jonkun uuden accordionin. Lisätään tähän tapahtumaan kaikkien avonaisten accordionien sulkeminen.

```
//when accordion panel is about to be opened, close all open panels.
    $(".collapse").on('show.bs.collapse', function(){
                        $('.panel-collapse.in').collapse('hide');
    });
```

3.4.4 Accordionit

Sivu koostuu accordioneista tehdyistä osista. Accordionit ovat haitarimaisesti avautuvia laatikoita. Accordionit luodaan XSLT muunnoksella seuraavasti. Asetetaan course elementille, joka on juuri elementti, niin panel-group ja id accordion. <xsl:apply-templates/> osoittaa mihin kohtaan lapsielementit sijoitetaan.

```
<xsl:template match="/course">

            <div class="panel-group row" id="accordion">
                        <xsl:apply-templates/>
            </div>

            ...
```

Course-elementin seuraavat lapsielementit ovat part-elementit. Ne toimivat samalla sivun accordioneina. Part-elementit ovat oikeastaan paneeleja, jotka käyttäytyvät accordionien tavoin. Paneelit luodaan asettamalla niille otsikko ja sisältö. Projektissa otsikkoa klikkaamalla avautuu paneelin sisältö. XSLT etsii kaikki part-elementit XML-dokumentissa, ja käy ne läpi yksitellen samoilla muotoilusäännöillä. XSLT asettaa eri id:n jokaisella läpikäynnillään, jolloin accordionit ovat yksilöityjä.

```
<xsl:template match="part">
    <div class="col-xs-12">
          <xsl:attribute name="class">
              panel panel-<xsl:value-of select="@type"/> accordion-caret
          </xsl:attribute>

<!—TÄMÄ ON PANEELIN OTSIKKO JOKA AVAA SEN SISÄLLÄ OLEVAN ACCORDIONIN -->
    <div class="panel-heading">
      <h2 class="panel-title">
<!—data-toggle=”collapse” kertoo mitä avataan ja data-parent osoittaa
course elementin id=”accordion” kohtaan -->
      <a data-toggle="collapse" data-parent="#accordion" class="accor-
      dion-toggle collapsed">
        <xsl:attribute name="href">#collapse<xsl:number/></xsl:attribute>
             <xsl:value-of select="@title"/>
       </a>
      </h2>
    </div>

<!—TÄMÄ ON AVATTAVA ACCORDION. HUOMAA ID=”collapse1”…-->
    <div class="panel-collapse collapse">
       <xsl:attribute name="id">collapse<xsl:number/></xsl:attribute>
        <div class="panel-body">
           <!—xsl:apply templates tuo kaikki lapsielementit panel-bodyn
           sisään-->
          <xsl:apply-templates/>
        </div>
      </div>
    </div>
</xsl:template>
```

Esimerkissä ensin otetaan kaikki part-elementit ja luodaan niistä paneeleja. Paneelin otsikon sisään lisätään a-elementti, jolle annetaan accordionin avaavat määritykset. Data-toggle määrittää, että avataan collapse-luokan omaavia elementtejä. Data-parent kertoo minkä elementin sisällä avattavat accordionit ovat. Annetaan vielä luokka accordion-toggle collapsed, joka määrittää accordionit avaavan luokan. Luodaan href-attribuutti XSLT:n attribute funktiolla ja annetaan sille arvoksi ”#collapse”. Ei haluta kuitenkaan avata kaikkia accordioneja, joten lisätään ”#collapse<xsl:number/>”, joka lisää jokaiseen linkin perään yksilöllisen numeron - #collapse1, #collapse2, #collapse3 ja niin edespäin. Paneelin otsikon sisään otetaan vielä part elementin otsikko XML-dokumentista.

Avattava accordion sisältää varsinaisen datan. Tehdään div jolle annetaan luokaksi ”panel-collapse collapse”, joka kertoo, että tämä on paneeli joka voi romahtaa/aueta. Annetaan tämän jälkeen attribuutti ”id”, jonka arvoksi tulee asettaa sama arvo, kuin avaavan otsikon href osoittaa. Tässä tapauksessa generoidaan numero collapsen perään samanlailla, kuin osoittavaan linkkiin. Sitten sijoitamme part elementtien sisällön accordionin sisään panel-bodyyn apply-templates elementillä.

Accordionit vieritti sivua alaspäin ja saattoi vierittää sivua puoleenväliin, jolloin käyttäjä joutui vierittämään takaisin osion alkuun. Tehdään Jquery:llä funktio, joka nappaa ”eventin”, eli tapahtuman kaikista elementeistä, joiden id on accordion. Tapahtuma on shown.bs.collapse, eli viimeksi avattu accordion. Etsitään edellinen panel-heading luokka tämän avautuneen accordionin sisältä ja tallennetaan se muuttujaan. Tämän jälkeen animoidaan sivun vieritys tallennettuun muuttujaan eli paneelin otsikkoon. 500 lopussa antaa padding arvoa, jolla voidaan hienosäätää ruutu menemään oikeaan kohtaan. (stackoverflow.)

```
//use bootstraps built-in event callback to scroll up to accordion header
            $("#accordion").on("shown.bs.collapse", function () {
                var myEl = $(this).find('.collapse.in').prev('.panel-heading');
                $('html, body').animate({
                    scrollTop: $(myEl).offset().top
                }, 500);
            });
```

4 Johtopäätökset

Tässä luvussa käydään läpi projektia ja sen hyötyjä. Kuinka projekti onnistui ja minkälaisia kokemuksia se tuotti. Projektin aikana tuli haasteita, joita tulen avaamaan enemmän.

Projekti toteutettiin onnistuneesti ja kaikki vaadittavat ominaisuudet saatiin valmiiksi.

Projektin tekniikat sopivat erinomaisesti nopeaan sisällön julkaisemiseen geneerisesti verkossa. XML ja XSLT on oiva lisä omalle verkkosivulle jos julkaistavaa materiaalia on rutkasti. Helposti voidaan suodattaa oleellisin data XML-dokumenteista sivulle. Kynnys lähteä muokkaamaan XML-dokumentteja XSLT-muunnoksella on pieni. XSLT ei sisällä monia eri funktioita tai muutakaan. Se on vain tyylitiedosto samaan tapaan kuin CSS. XSLT:ssä pärjää, kun hahmottaa miten XPathillä osoitetaan jotain tiettyä osaa XML-dokumentista. XSLT ottaa elementin ja kaikki sen lapsielementit ja suodattaa niistä tietoa miten kehittäjä haluaa. XML ei itsessään tee mitään, vaan vaatii jonkun sitä käyttävän tekniikan, jolla voidaan XML:n dataa lukea.

HTML-kieltä syntyy XSLT muunnoksen lopputuloksena. Kun valitaan XML-dokumentista elementtejä käsittelyyn XSLT:llä, niin suodatetaan tietoja suoraan HTML-rakenteeseen, joka syötetään ulostulevaan tiedostoon samaan kohtaan. HTML on vakiintunut peruskieli kaikenlaisen verkkosivujen tekemiseen ja on siksi pakollinen projektin kannalta.

Javascript ja Jquery on erittäin helppo opetella, mutta saattaa olla vaikea sisäistää, mitä kaikkea niillä voidaan tehdä. Onneksi internet on pullollaan esimerkkejä ja ohjeita kaikenlaisiin toiminnallisuuksien kehittämiseen. Lisäksi W3Schools:n sivuilla on kattavat esimerkit aloittelijoille ja kokeneemmillekkin koodareille kaikista perustekniikoista.

Github on erittäin pätevä lähdekoodin hallintaan tarkoitettu työkalu, joka yhdistettynä gh-pages liitännäisen kanssa on erittäin hyvä. Github tarjoaa kattavat dokumentaatiot asentamiseen, oman projektin luomiseen ja kloonaamiseen työasemalle. Githubin käyttäminen komentoriviltä vaatii totuttelemista, mutta palkitsee sen jälkeen käyttäjänsä yksinkertaisuudellaan.

Bootstrap ei turhaan ole suosituin sovelluskehys responsiivisten verkkosivujen luomiseen. Bootstrapin omilla sivuilla on erittäin kattavat dokumentit kaikista komponenteista, tyyleistä ja javascriptia vaativista osista. Bootstrap antaa paljon, mutta voi myös opettaa paljon. Bootstrapin ominaisuuksista suurin osa on yksinkertaisia CSS-luokkia, jotka ovat helposti saavutettavissa myös itse. Kuitenkin siinä säästää pakostikin aikaa, kun ei tarvitse

säätää pienten palikoiden kanssa. Tosin koska BS on niin suosittu, syntyy helposti liikaa samalta näyttäviä sivuja. Jotkut BS:n elementit ovat aika kiven alla ja hankala lähteä muokkaamaan rikkomatta sivua.

4.1 Kohdatut haasteet

Projekti eteni hieman eri järjestyksessä kun normaalisti tutkimustyyppisissä opinnäytetöissä. Projektissa aloitettiin heti kurssisivun lähdekoodin kehittämisessä. Samalla etsittiin tietoa käytetyistä tekniikoista. Merkattiin lähteet ylös asian sisäistämisen jälkeen. Tämän jälkeen toteutettiin tekniikka opitulla tavalla. Vasta projektin ollessa noin 80 prosenttia valmis, alettiin kirjoittamaan tietoperustaa ja empiiristä osuutta. Toteutustapa on hyvä, mutta jälkeenpäin saattaa tulla ideoita miten olisi tehnyt asian paremmin.

Ajankäyttö tuotti vain vähän haasteita, sillä projektin kimppuun päästiin jo ennen kesälomaa. Olin itse koko kesän ajan kolmivuoro työssä, jolloin projektia tehtiin vaihtelevasti aina kun aikaa oli. Oli myös haastavaa saada projektia täysin loppuun kesän aikana, sillä toimeksiantajaan sai yhteyttä vain silloin tällöin. Aineistoa kirjoittamiseen kerääntyi runsaasti projektin teknillisen osuuden aikana, joka edes auttoi kirjoittamista opinnäytetyötä varten.

Kesän jälkeen projekti valmistui pikavauhtia loppuun toimeksiantajan kommenttien ja yhteistyön tuloksena. Yhteistyötä tapahtui sähköpostilla ja järjestämällä tapaamisia, joissa käytiin läpi mitä voisi parantaa ja mitä on havaittu toimivan oudosti. Keskusteltiin toimista mitä voidaan tehdä ongelmien korjaamiseksi. Toimeksiantajan ohjelmointitaustan takia tapaamiset olivat hyvin tuottoisia ja ongelmiin löytyi ratkaisu helposti. Muut projektin parissa olevat sidosryhmät hoitivat lähinnä sisällöntuottamista.

Projektin vaatimuksissa oli mainittu selainyhteensopivuus suosituimmilla selaimilla. Kuitenkin päätettiin rajata Internet Explorerin versio 8 pois projektista. Niin vanha versio IE:stä on erittäin vähän käytetty, ja jos näin vanha vielä joltain löytyy, tulisi neuvoa käyttäjää hankkimaan uusi versio selaimesta. Myös media kyselyt, ja osa kirjastoista ei toimi oikein IE8:lla. Media kyselyt kyllä saisi toimimaan esimerkiksi respond.js kirjastolla, joka tekee juuri sen. IE:lle pystytään tekemään ehdollisia lauseita metatietoihin. Tehdäänkin IE suorittamaan vain versiolla 8 tai pienempi, javascript pätkä, joka pompauttaa alert-ikkunan ruudulle ja ohjaa käyttäjän Microsoftin sivuille lataamaan uusinta versiota Internet Explorerista.

```
<!--[if lt IE 9]>
  <script type="text/javascript">
          alert("Seems you're using IE8 or lower. Please upgrade to IE9
          or higher.");
```

```
            window.location.href("http://windows.microsoft.com/en-us/in-
            ternet-explorer/download-ie");
    </script>
<![endif]-->
```

Modaalien sisältämä editori tiputettiin pois sivulta, koska kesti vähintään 20 sekuntia ladata embed.js-kirjasto, jota se käyttää. Embedattu editori tulisi olla omalla välilehdellään, johon ei ladattaisi mitään muuta. Modaaleihin laitettiin painike, jolla voidaan avata editori uuteen välilehteen. Tällöin Embed.js kirjasto ladataan vain kerran sille tehtävälle mikä avataan. Lisäksi käytetään XSLT:n metodia xsl:choose tarkistamaan onko kyseisellä modaalilla tehtävää ollenkaan. Painiketta ei julkaista jos tehtävä puuttuu.

```
<xsl:choose>
    <xsl:when test="a/@href and a/@class ='jsbin-embed'">
            <a target="_blank" class="btn btn-success col-xs-12 col-sm-
            4">
            <xsl:attribute name="href">
                    <xsl:value-of select="a/@href"/>
            </xsl:attribute>
            <span class="glyphicon glyphicon-new-window"/>  Open
            JSBin</a>
</xsl:when>
</xsl:choose>
```

Lisäksi tehtäviä varten on luotu omat ”standalone”-sivut, joissa on vain kyseinen tehtävä uudella välilehdellä. Tämä helpottaa tulostamista ja saattaa olla helpompi selata mobiililla.

Uuteen välilehteen avataan myös slidet, koska Reveal-kirjasto ei suostunut toimimaan Firefoxissa upotettuna modaalissa. On mahdollista tuottaa slideista pdf muotoista sisältöä, mutta silloin slidet ovat allekkain ja koko kirjasto menettää merkityksensä. Tulisi ainoastaan käyttää tulostamiseen. Uudessa välilehdessä slidet toimii odotetusti kaikissa selaimissa.

Videoiden upottaminen suoraan sivulle, eli embeddaus, tiputettiin sivulta kokonaan pois ja laitettiin avautumaan uuteen välilehteen. Videoiden lataaminen modaaleihin hidasti liikaa käyttökokemusta. Videot avautuvat suoraan youtubeen antamalla youtubeid XML-tiedostossa. XSLT muunnos lisää sen a-linkkiin, jossa on youtuben osoite valmiina.

```
<a type="button" target="_blank" class="btn btn-info col-xs-12 col-sm-4">
<xsl:attribute name="href">https://www.youtube.com/watch?v=<xsl:value-of
select="@youtubeid"/></xsl:attribute>
<span title="Open video in new window" class="glyphicon glyphicon-
facetime-video"></span>  <xsl:value-of select="@title"/>
</a>
```

Tehtävä painikkeet vaihdettiin olemaan täysileveitä ”skills”-osion sisään. Samalla vaihdettiin ikonit saman luokan sisään ja asetettiin ne oikeaan reunaan float-määrityksellä. Tehtäväpainikkeet jakautuivat aluksi puoliksi, ja ylimenevä teksti piiloitettiin text-overflow:ellipsis määrityksellä, joka katkaisee tekstin ylimenevältä osin ja asettaa kolme pistettä (…) sanan

loppuun. Tämä toimii upeasti jos nappuloissa ei ole kuin pelkkää tekstiä. Tehtävänappuloissa on tähti-ikoneja merkkaamassa tehtävän vaikeusastetta. Firefox tulkitsee ikonit tekstiksi ja osaa leikata tekstin pisteillä, myös jos yksi tähti menee yli painikkeesta. Chrome taas ei tulkitse ikoneja tekstiksi ja näin ei osaa leikata niitä. Chrome leikkaa pisteillä vasta kun tähdet ovat jo ulkona painikkeesta, ja teksti alkaa mennä ulos.

```
.exercise .btn{
            width:98%;
            margin:2%;
            }

.exercise .ratings {
            float:right;
}
```

Kuva 25 - Uudet tehtäväpainikkeet

XSLT:n muunnos suoritettiin MagicXML:llä selaimessa, joka toi hitausongelmia suuren tehtävämäärän vuoksi. Kaikki tehtävät tulisi siirtää käännettäväksi palvelimen puolella. Päätettiin luopua MagicXML:n käytöstä kokonaan ja laittaa sivusto ”pre-build” vaiheeseen, joka kääntää kaikki XML-tiedostot palvelimella ennen selaimelle lähettämistä.

Kääntäminen tapahtuu lisäämällä xsl-tiedostossa juuritemplaten sisään html-elementti, jonka sisään kopioidaan etusivun HTML-koodi accordioneihin asti. Ne luodaan <xsl:apply-templates>-osoittimella, joka asettaa kaikki course-elementin sisällä olevat lapsielementit paneelien sisälle. Loput elementit muotoillaan samaan tapaan erikseen.

```
<xsl:template match="/course">
<html>
head>
<meta charset="utf-8"/>
<meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<title>Orientation to Software Engineering</title>

<!-- CSS -->
<link rel="stylesheet" type="text/css" href="css/bootstrap.css"/>
...
<!-- SCRIPTS -->
<script src="js/jquery-1.11.3.min.js"></script>
...
</head>
```

```
<body class="language-javascript">

<div id="content" class="container">
...LOGO JA OTSIKKO
<div class="panel-group row" id="accordion">
            <xsl:apply-templates/>
</div>
            <!-- Sidebar -->
</div>
</body>

</html>
</xsl:template>
```

Tehtävät generoidaan myös HTML-koodiksi ennen selaimelle lähetystä. Tämä vaatii ensin vielä pienen scriptin, joka lisää jokaiseen exercise luokkaan funktion. Funktio lisää kaikkien XML-tehtäväsivujen perään ”.html”-tagin.

```
$(function() {

            //get prexslt exercises by xhr
            $(".exercise").each(
                        function() {

            $(this).load($(this).data("xml")+".html");
                                    }
);
```

Projektiin on toimeksiantajan ansiosta nyt olemassa javalla tehty XSLT-muunnin, joka ylikirjoittaa kaikki HTML-tiedostot XML- ja XSL-tiedostojen mukaan uudestaan. Tämä tehdään koska selaimelle lähetetään HTML-tiedostoja ja halutaan helpottaa useiden tehtäväsivujen julkaisemista.

Lisäksi Googlen PageSpeed Insight-sivu, jolla mitataan sivun lataamisnopeutta, kertoi että kaikki kuvat tulee pakata pienemmiksi. Tämä karsi pois muutaman sekunnin latausajoista.(PageSpeedInsight.)

4.2 Tulosten hyödyntäminen

Haaga-Helian organisaatio hyötyy projektista siten, että he saavat sulavasti toimivan verkkosivun uudelle ohjelmoinnin kurssille. Haaga-Helia voi luoda helposti uusia vastaavia kurssisivuja XSLT-pohjaa hyväksikäyttäen. Tarvitsee luoda vain XML-dokumentti, joka sisältää kurssin tiedot. En tiedä tuleeko kurssisivut näkymään ulkopuolisille, mutta tulevat kurssin opiskelijat ainakin saavat HH-organisaatiota edustavan näköiset sivut itselleen, joita he voivat selailla vaivattomasti puhelimella, tabletilla tai tietokoneellaan.

Dokumentti antaa Haaga-Helian opettajille kattavan dokumentaation siitä miten kurssisivu on muodostettu ja auttaa heitä jatkamaan sivuston päivittämistä. Opettajat voivat jatkossa käyttää tätä työtä käsikirjana sivun päivittämistä varten.

Haaga-helia voi myös hyödyntää kurssisivua opettaakseen XSLT muunnoksen tekemistä XML-dokumenteille. Lisäksi voidaan käyttää projektia responsiivisen suunnittelun ymmärtämiseen ja sisäistämiseen.

Opinnäytetyö dokumenttia voi hyödyntää oman tuntemuksen vahvistamiseen verkkopalveluiden toteutuksesta käyttäen responsiivisia menetelmiä. HTML- ja CSS-kielten parissa olevat aloittelijat ja kokeneemmat ohjelmoijat voivat saada ideoita omaan projektiinsa tai vahvistaa tietämystään.

XSLT-projekteista kiinnostuneet ja aloittelevat ohjelmoijat voivat saada käsityksen XML-tiedostojen ja XSL-muunnosten käytön hyödyllisyydestä kun rakennetaan staattinen verkkosi-vusto.

Responsiivisesta suunnittelusta ja asettelusta kiinnostuneet saavat käsityksen miksi se on hyvä tapa rakentaa verkkosivusto esimerkiksi tulevaisuuden haasteita varten, kun päätelaitteita on useita eri kokoja. Samalla saa käsityksen mediakyselyjen ja joustavien elementtien ja kuvien tuomasta hyödystä. Testauksesta voi oppia muutaman tavan helpottaa testausta kun käsitellään responsiivista verkkosivua.

4.3 Jatkokehitysideat

Tässä kappaleessa pohditaan mitä sivulle voisi vielä kehittää, jotta toimeksiannon sivustosta saataisiin vieläkin tarkoituksenmukaisempi.

Standalone tehtäväsivut tehtiin, jotta tulostaminen olisi helpompaa kun välilehdellä olisi vain tehtävänanto, eikä muuta. Ehdottaisinkin, että näille sivuille yritettäisiin sisällyttää embedattu Jsbin-editori. Index-sivulla tämä aiheutti massiivisen hidastumisen kun embed.js ladattiin kaikille tehtäville, mutta jos kirjasto ladataan vain kerran, niin tuskin hitaus tulee olemaan huomattava. Nyt suoraan Jsbin:n sivuille aukeavat tehtävät ovat itsessään riittävät, mutta toki olisi kiva jos tehtävänanto olisi samalla sivulla. En usko, että tehtävien tulostamiseenkaan editori vaikuttaa. Tätä voisi kokeilla ja ottaa käyttöön, jos editori ei vaikuta latausaikaan tai tulostamiseen negatiivisesti.

Tällä hetkellä Prism.js kirjaston highlight-laatikot jättävät yhden ylimääräisen rivin tyhjää tilaa jokaisen koodipätkän jälkeen. Ei ole sinänsä iso ongelma, mutta jos esimerkkejä on paljon, niin kyllä se turhaa tilaa vie. Tulisi selvittää miksi kyseinen kirjasto tämän tempun tekee. Voi olla, että se liittyy johonkin whitespace määritykseen prism.css-tiedostossa.

Lähteet

Administrative user 2014– Resizing YouTube and HTML5 videos proportionally using CSS for responsive web design. Luettavissa: http://basicuse.net/articles/pl/textile/html_css/resizing_youtube_and_html5_videos_proportionally_using_css_for_responsive_web_design. Luettu: 20.08.2015.

Ala-Äijälä J. 2012. Mobiiliselaimet ruotuun viewport-tagilla. Luettavissa: http://www.aucor.fi/blogi/mobiiliselaimet-ruotuun-viewport-tagilla. Luettu: 07.09.2015.

Bosomworth, D. 2015. Mobile marketing statistics. Luettavissa: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. Luettu: 20.08.2015.

Bootstrap 2014. Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Luettavissa: http://getbootstrap.com/. Luettu: 15.5.2015.

Can I Use. Viewport units. Luettavissa: http://caniuse.com/#feat=viewport-units. Luettu: 20.10.2015.

Chaffer, J. & Swedberg, K. 2014. Learning jQuery (fourth edition).

Code My Views. Mobile First Design - Why It's Great and Why It Sucks. Luettavissa: https://codemyviews.com/blog/mobilefirst. Luettu: 21.08.2015.

Gasston, P. 2011. Book of CSS3 - A Developer's Guide to the Future of Web Design (2nd Edition).

Google. PageSpeed Insight – Identify ways to make website faster. Linkki: https://developers.google.com/speed/pagespeed/.

Hakim El Hattab 2015. A framework for easily creating beautiful presentations using HTML. Luettavissa: http://lab.hakim.se/reveal-js/#/. Luettu: 10.10.2015.

Knight, K. 2011. Guidelines for responsive web design. Luettavissa: www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design. Luettu 16.08.2015.

LePaige, P. 2014. Images in CSS. Luettavissa: https://developers.google.com/web/fundamentals/media/images/images-in-css. Luettu 16.08.2015.

LePage, P. 2014. Responsive Web Design Basics - Set the viewport. Luettavissa: https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/set-the-viewport?hl=en. Luettu 07.09.2015.

Marcotte, E. 2010. A List Apart - Responsive Web Design. Luettavissa: http://alistapart.com/article/responsive-web-design. Luettu: 15.08.2015.

Marcotte, E. 2009. A List Apart - Fluid Grids. Luettavissa: http://alistapart.com/article/fluid-grids. Luettu: 15.08.2015.

Microsoft Developers – XSLT Compability Change (IE). Luettavissa: https://msdn.microsoft.com/library/hh180178%28v=vs.85%29.aspx. Luettu: 15.06.2015.

Netmarketshare. Resolutions based on global market share in 2015. Luettavissa: https://www.netmarketshare.com/report.aspx?qprid=17&qptimeframe=Y. Luettu: 15.08.2015.

Stackoverflow – Scroll to content after accordion is opened. Luettavissa: http://stackoverflow.com/questions/23063669/scrolling-to-content-after-a-bootstrap-accordion-is-opened. Luettu: 23.08.2015.

Wikipedia – Esineiden internet. Luettavissa: https://fi.wikipedia.org/wiki/Esineiden_internet Luettu: 15.08.2015.

West, M. 2012. HTML5 Foundations.

W3Schools – Meta tag. Luettavissa: http://www.w3schools.com/tags/tag_meta.asp. Luettu: 24.08.2015.

Wikipedia. JavaScript. Luettavissa: https://en.wikipedia.org/wiki/JavaScript. Luettu: 13.10.2015.

W3C 2008. Recommendations for XML(1.0 (fifth edition)). Luettavissa: http://www.w3.org/TR/xml/#sec-doc-entity. Luettu: 10.06.2015.

W3C 1999. Recommendations for XSLT(1.0). Luettavissa: http://www.w3.org/TR/xslt Luettu: 10.06.2015.

Zakas, Nicholas C. 2014. Principles of Object-Oriented JavaScript.

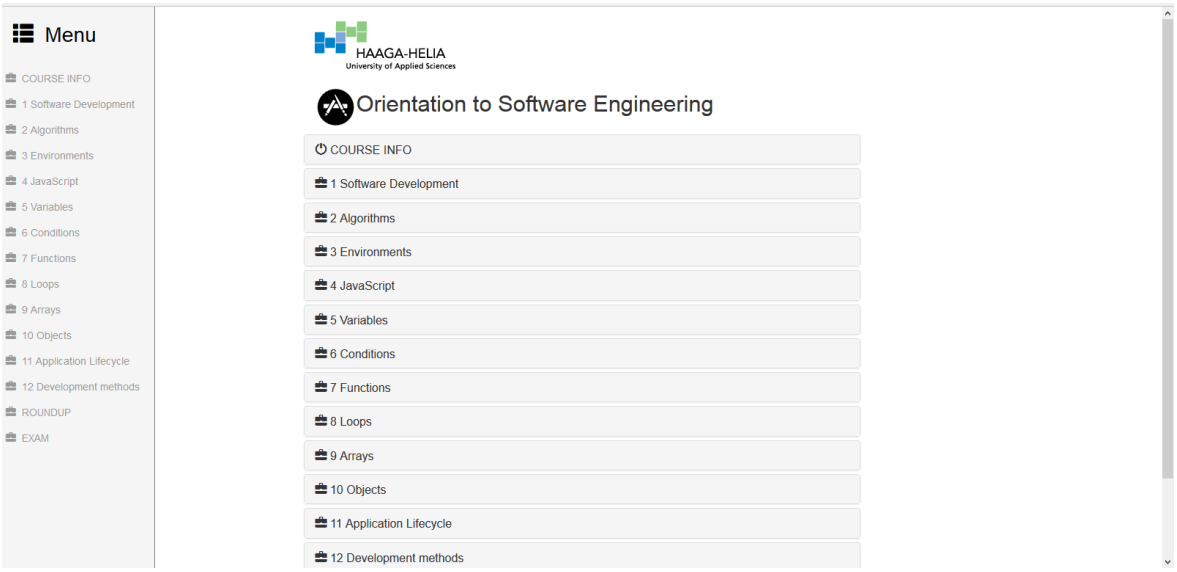
W3C. CSS Fonts. Luettavissa: http://www.w3schools.com/css/css_font.asp. Luettu: 17.07.2015.

KUVA1. Katsottavissa: http://lunarpages.com/uptime/basics-of-responsive-design-using-fluid-grids.

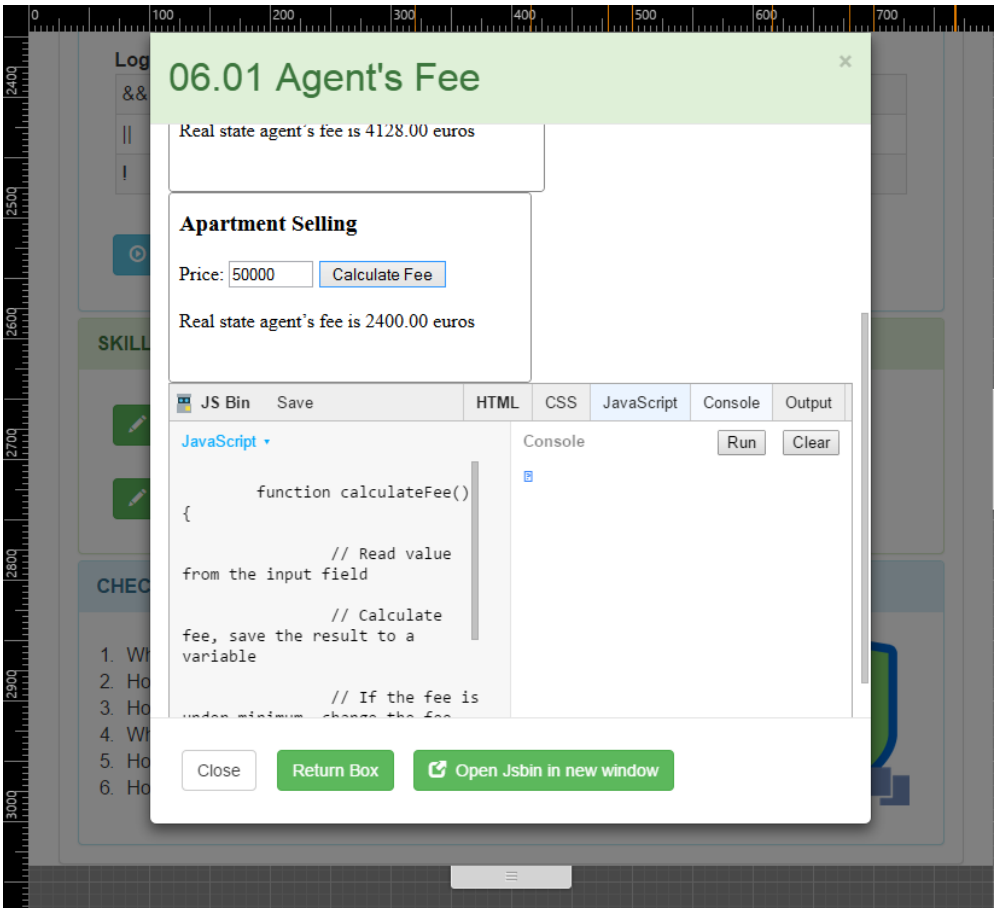
Liitteet

no.	tehtävä	lopputulos	aloituskriteeri	tuntia	viikko	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
1	**Projektin käynnistys**																											
1.1	Aloituskokous	Kokous pidetty				x																						
1.2	Pöytäkirjan laatiminen	Pöytäkirjat jaettu					x																					
2	**Toteutuksen suunnittelu**					22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
2.2	Tekninen suunnittelu	Ratkaisun tekniset vaatimukset määritelty. Kirjastot valittu.							x																			
3	**Ratkaisun toteutus**					22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
3.1	Kirjastojen sisällyttäminen	Kirjastot on tuotu sivulle						x	x																			
3.2	Sivuston skaalautuminen	Sivusto skaalautuu näyttökoon mukaan							x	x	x	x																
3.3	Selainyhteensopivuus	Sivusto latautuu eri selaimilla											x	x	x													
3.4	Testaus									x	x	x	x	x	x	x												
3.5	Valmiin ratkaisun luovuttaminen	Toimeksiantajalla valmis tuote															x											
4	**Teoriaosuuden toteutus**					22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
4.1	Teoriaosan suunnittelu	Teoriaosan rakenne valmis								x																		
4.2	Aineiston keräys	Teoriaosalle riittävä perusta							x	x	x	x	x	x	x													
4.3	Toteutus															x	x	x	x	x								
4.4	Tarkistus	Teoriaosa oikoluettu																			x							
5	**Tuloksien analysoiti**					22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
5.1	Analyysin suunnittelu	Rakenne valmis																		x	x							
5.2	Analyysin kirjoittaminen	Opinnäytetyön teoriaosaa ja ratkaisua hyväksikäyttäen kirjoitettu analyysi																				x	x	x	x	x	x	
5.3	Analyysin tarkistus	Tuloksien oikoluku																										x
6	**Projektin hallinta**																											
6.1	Edistymisen raportointi																											
6.2	Ohjausryhmän kokous																											
6.3	Pöytäkirjan laatiminen	Pöytäkirjat laadittu																										

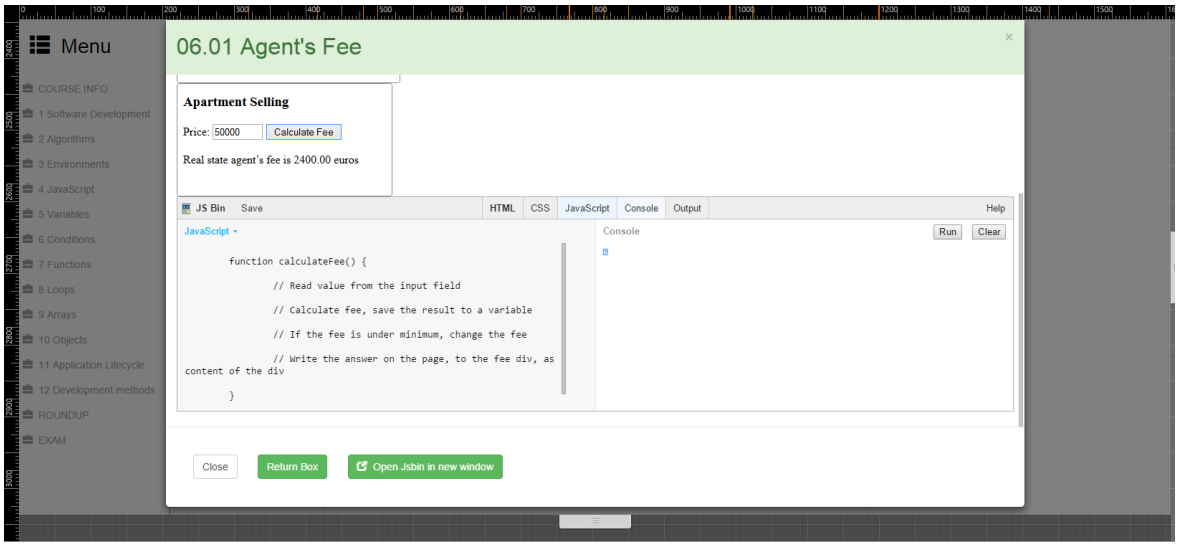
Liite 1 - Ajankäytön hallinta suunnitelma



Liite 2. Kurssisivu kokonaisuudessaan



Liite 3. Modal wide yli 768 pikseliä



Liite 4 - Modaali maksimileveys 1200 pikseliä