jamk.fi

Musiikkisoitinsovelluksen toteuttaminen musiikkipalveluun

Juho Suomi

Opinnäytetyö
Huhtikuu 2017
Luonnontieteiden ala
Tradenomi (AMK), Tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Suomi, Juho	Opinnäytetyö, AMK	Huhtikuu 2017
	Sivumäärä 34	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Musiikkisoitinsovelluksen toteuttaminen musiikkipalveluun

Tutkinto-ohjelma
Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)
Tommi Tuikka

Toimeksiantaja(t)
Music.Info Finland Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa Web-pohjainen musiikkisoitinsovellus Music.Info Finland Oy:lle. Tutkimus toteutettiin kehittämistutkimuksena.

Toimeksiantajan kanssa tehdyn vaatimusmäärittelyn jälkeen suunniteltiin arkkitehtuurimalli sovelluksen komponenteista ja rajapinnoista. Tämän pohjalta ryhdyttiin kehittämään Web-sovellusta. Vaihtoehtojen kartoituksen jälkeen sopivammaksi musiikkikirjastoksi valikoitui SoundManager2-kirjasto.

Sovellus toteutettiin käyttämällä AngularJS-sovelluskehystä. Sovelluksen käyttöliittymä rakennettiin komponenteista, jotka koottiin yhdeksi kokonaisuudeksi Pattern Lab -työkalulla. MusicInfon tietokannassa olevan datan hakeminen sovellukseen toteutettiin REST-kutsujen avulla HTTP-protokollaa käyttämällä.

Työn lopputuloksena syntyi toimiva musiikkisoitinsovellus, joka tulee olemaan lisäpalvelu nykyiseen. Toimeksiantaja voi jatkaa sovelluksen edelleen kehittämistä.

Avainsanat (asiasanat)

JavaScript, AngularJS, Pattern Lab, SoundManager2, REST

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Suomi, Juho	Bachelor's thesis	April 2017
		Language of publication: Finnish
	Number of pages 34	Permission for web publication: x

Title of publication

Developing music player application for music service

Degree programme

Business Information Systems

Supervisor(s)

Tuikka, Tommi

Assigned by

Music.Info Finland Oy

Abstract

The objective of this thesis was to produce a web-based music player application for Music.Info Finland Oy, the company that assigned the thesis. The study was implemented as development research. Requirements analysis performed with the client was followed by the design of an architecture model of the application's components and interfaces. Based on this, the development of the web application started. SoundManager2 music library was selected as the most suitable option amongst the alternatives available.

The application was developed with AngularJS framework and the user interface of the application was built using components that were compiled into a single page with Pattern Lab tool. The data retrieval from MusicInfo's database and display of the information in the user interface were implemented with REST-methods by using the HTTP-protocol.

The final product was a functional music player application, which will be an additional feature on the current service. The company may continue further development of the application.

Keywords/tags (subjects)

JavaScript, AngularJS, Pattern Lab, SoundManager2, REST

Miscellaneous

Sisältö

Kuviot

TERMIT

API	Application programming interface. Ohjelmointirajapinta, jolla ohjelmat voivat lähettää pyyntöjä, vaihtaa tietoja eli keskustella keskenään.
CSS	Cascading Style Sheets. HTML-dokumenttien tyylittelykieli.
DOM	Document Object Model. Tapa jolla kuvataan dokumentti rakennepuuna, jonka eri olioita voi hakea, tutkia ja manipuloida esimerkiksi JavaScriptin avulla.
HTTP	HyperText Transfer Protocol. Käytetään tiedonsiirtoon selaimen ja palvelimen välillä.
JSON	JavaScript Object Notation on tiedostomuoto, jota käytetään tiedonvälitykseen.
MVC	Model-view-controller on ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on käyttöliittymän erottaminen sovellusaluetiedosta.
Web-sovellus	Web-sovellukset ovat ohjelmia, joihin pääsee käsiksi Web-selaimella. Ne vaativat merkintäkielen (esim. HTML, XML, XHTML) tai skriptikielen (esim. JavaScript), mitä Web-selain osaa tulkita.

1 Johdanto

1.1 Opinnäytetyön tausta, tavoitteet ja rajaukset

Musiikkipalvelut ovat tällä hetkellä suosittuja ja uusia sovelluksia kehitetään jatkuvasti (Perälä 2016, 2). Musiikkia kuunnellaan suoratoistona Internetin, tabletin tai kännykän kautta. Tämän opinnäytetyön toimeksiantaja Music.Info Finland Oy tarjoaa palvelua musiikkiin liittyvään taustatietoon. Se kokoaa hajallaan olevaa musiikkitietoa yhdelle sivustolle ja tarjoaa laajan musiikkitietokannan. Toimeksiantajalla on tarve laajentaa palvelua musiikin suoratoistoon ja tarjota näin käyttäjille musiikkitiedon lisäksi mahdollisuutta kuunnella musiikkia.

Tämän opinnäytetyön tavoite on suunnitella ja toteuttaa toimiva selainpohjainen musiikkisoitin MusicInfo-palvelulle. Kehitystyön avulla musiikkipalvelu pystyy jatkossa tarjoamaan kuunneltavaa musiikkia. Kehittämistyössä kehitetään jotain asiaa tai toimintaa kohti parempaa. Se edellyttää nykytilan kartoituksen, vaihtoehtojen etsinnän ja arvottamisen, tavoitteiden määrittämisen ja keinojen valinnan tavoitteisiin pääsemiseksi. (Kananen 2010, 159.)

Toimeksiantajan kanssa käytiin yhdessä läpi sovelluksen määrittelyt, luotiin visio ulkoasusta ja toiminnallisuudesta. Sovelluksen vähimmäisvaatimukset ovat:

- sen täytyy tukea useita formaatteja ja pystyä toistamaan eri tiedostotyyppejä
- se toimii eri alustoilla (käyttöjärjestelmät)
- käyttäjä voi lisätä ja poistaa kappaleita soittolistasta
- soitettavasta kappaleesta näytetään lisätietoja
- sovellus pystyy hakemaan kappaleet Musicinfon tietokannasta ja lisäämään ne käyttäjän soittolistaan.

Musikkisovelluksen kehittämisen edellytyksenä on, että valittu tekniikka ja työkalut ovat yhteensopivia MusicInfo-palvelun arkkitehtuurin kanssa. Toimeksiantajan kanssa päätimme yhdessä, millä tekniikoilla soittimen teko toteutetaan palveluun.

1.2 Tutkimuskysymykset

Opinnäytetyössä tutkin musiikkisoittimen kehittämistä Web-tekniikoilla. Kehitystyön kysymykset liittyvät sovelluksen arkkitehtuuriseen suunnitteluun ja toteuttamiseen JavaScript-kirjastoa käyttämällä.

Tutkimuskysymys:

- Miten kehitetetään Web-pohjainen musiikkisoitinsovellus?

Alakysymykset

- Miten suunnitellaan Web-sovelluksen arkkitehtuuri?
- Mikä kirjasto on sopivin musiikkisoitinsovelluksen kehittämiseen?
- Miten toteutetaan musiikkisoitinsovellus?

1.3 Music.Info Finland Oy

Music.Info Finland Oy on 2012 perustettu musiikkialan startup-yritys. Se ylläpitää MusicInfo-palvelua, joka kokoaa internetissä olevaa musiikkitietoa. Palvelu hakee tietoja eri lähteistä. Tietokanta pitää sisällään tietoa yli seitsemästä miljoonasta artistista. MusicInfo-palvelu toimii hakukonetyyppisesti. Palvelun kautta kuluttajan on mahdollista löytää tietoa musiikin esittäjistä, heidän historiastaan, kappaleista, niiden säveltäjistä ja sanoittajista. Palvelun avulla on mahdollista löytää myös harvinaisempaa ja vähemmän tunnettua musiikkia. Sieltä voi hakea haluamaansa esittäjää ja selata hänen musiikkiinsa liittyvää tietoa. Lisäksi palvelu kertoo, mistä kyseistä musiikkia voi ostaa.

2 Arkkitehtuuri

Arkkitehtuurista suunnittelua käytetään Web-sovellusten kehittämisessä. Suunnittelu tarjoaa yhden tai useamman vaihtoehdon ohjelmiston toteuttamiseksi. Ohjelmiston suunnittelussa on kaksi vaihetta: arkkitehtuurisuunnittelu ja oliosuunnittelu. Arkkitehtuurisuunnittelu määrittää ohjelman rakenteen karkealla tasolla. Siinä kuvataan, mistä isoista rakennekomponenteista ohjelma koostuu ja kuinka ne yhdistetään (rajapinnat). Oliosuunnittelussa keskitytään suunnittelemaan yksittäisiä komponentteja. (Luukkainen 2011, 10.)

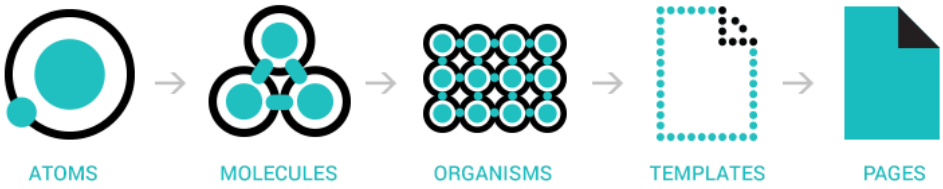
Arkkitehtuurisessa kuvauksessa käytetään malleja sovelluksen rakenteen määrittämiseksi. Malleissa kuvataan eri komponentteja ja niiden välisiä rajapintoja. (Luukkainen 2011, 10.)

Mallit ovat tärkeitä kirjoitettaessa ylläpidettävää ja uudelleenkäytettävää koodia. Erilaisia malleja löytyy paljon. Mallia valittaessa on otettava huomioon käyttötapaukset, laatuvaatimukset ja valittu tekniikka. Web-sovellukseen voidaan suunnitella oma tai käyttää hyväksi olemassa olevia malleja. (Luukkainen 2011, 23 - 24.)

2.1 Atominen suunnittelu

Atominen suunnitelu (Atomic Design) on menetelmä luoda vankkarakenteisia järjestelmiä kokonaisuuden huomioiden (Adiseshiah 2016). Se tarkoittaa järjestelmän luomista palasista. Näitä palasia yhdistämällä voidaan luoda elementtejä ja malleja, joita voidaan käyttää yhä uudelleen ja uudelleen (Toscano 2015). Atomisessa suunnittelussa yhdistetään alemman tason toiminta korkean tason toiminnan kanssa Web-sovellukseksi (Bloom 2014).

Atominen suunnittelu jakaa käyttöliittymän yksinkertaisiin peruselementteihin. Elementit voidaan jakaa rakennuspalikoihin (atomit, molekyylit, organismit) ja säiliöihin (mallit ja sivut). Keskeinen periaate rakentamisessa on käyttää peruselementtejä seuraavalle tasolle etenemiseksi. (Adiseshiah 2016.)



Kuvio 1. Atomisen suunnittelun elementit

Atomit ovat pienimpiä elementtejä. Käyttöliittymässä niitä ovat mm. otsikot, kappaleet, painikkeet ja syötekentät. Ne sisältävät myös abstrakteja elementtejä, kuten väripaletin, fontteja ja animaatioita. (Bloom 2014.)

Molekyylit ovat käyttöliittymässä ryhmä elementtejä, jotka toimivat yhdessä yhtenä yksikkönä. Hakulomake on esimerkki molekyylistä käyttöliittymässä. Atomit, kuten lomakkeen otsikko, syötekenttä ja painike yhdistettynä muodostavat lomakkeen, joka on molekyyli. (Bloom 2014.)

Organismit ovat molekyyliryhmiä, esimerkiksi ylätunniste (header) koostuu hakukentästä, valikosta ja logosta. Ne ovat liittyneet yhteen muodostaen erillisen osan rajapintaa. Rakennettaessa molekyyleistä organismeja luodaan itsenäisiä, siirrettäviä ja uudelleenkäytettäviä komponentteja. (Bloom 2014.)

Mallipohjat koostuvat enimmäkseen organismeista, jotka yhdistetään sivutason objekteihin. Malleja käytetään sivun toiminnan testaamiseen. Sivut ovat mallipohjia, joiden sisällöksi vaihdetaan oikea esiteltävä sisältö. Se antaa tarkan kuvan siitä, mitä käyttäjälle lopulta näkyy. Sivut tarjoavat mahdollisuuden testata ja nähdä, kuinka kaikki elementit sopivat yhteen. (Bloom 2014.)

2.2 Suunnittelumallit

Suunnittelumallit ovat uudelleenkäytettäviä ratkaisuja yleisesti esiintyviin ongelmiin ohjelmistojen suunnittelussa. Suunnittelumalleilla on kaksi käyttötarkoitusta ohjelmistokehityksessä. Ne tarjoavat yhteisen termistön ja ne on määritelty tietyille skenaarioille. Ongelman kuvaus on yksinkertaisempaa suunnittelumallin avulla kuin esittää ratkaisu koodin muodossa. Jokainen suunnittelumalli keskittyy tiettyyn

oliosuunnitteluun, ongelmaan tai kysymykseen. Suunnittelumalleja voidaan jakaa useisiin eri luokkiin. Seuraavassa tarkastelen kolmea näistä luokista. (Osmani 2012.)

Luovat suunnittelumallit

Nämä mallit keskittyvät olioiden luomismekanismeihin. Olioita luodaan työskentelytilanteeseen sopivalla tavalla. Olioista voidaan tehdä usein liian monimutkaisia. Nämä mallit pyrkivät ratkaisemaan tämän ongelman ja ovat osa luomisprosessia. (Osmani 2012.)

Rakenteelliset suunnittelumallit

Rakenteelliset mallit keskittyvät olioiden koostumukseen. Ne pyrkivät tunnistamaan eri olioiden välisiä suhteita. Nämä mallit auttavat varmistamaan sen, että yhden järjestelmän osan muuttuessa, koko rakenteen ei tarvitse muuttua. (Osmani 2012.)

Käyttäytymisen suunnittelumallit

Käyttäytymisen suunnittelumallit keskittyvät parantamaan tai virtaviivaistamaan erilaisten olioiden välistä viestintää järjestelmässä. (Osmani 2012.)

2.3 REST

REST (Representational State Transfer) on arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen (Reini 2012). Sitä käytetään kehittämään nopeita, kevyitä, skaalautuvia ja helposti ylläpidettäviä Web-palveluita. Nämä palvelut käyttävät usein HTTP-protokollaa viestintävälineenä. Palveluja, jotka pohjautuvat REST-arkkitehtuurimalliin, kutsutaan RESTful -palveluiksi. RESTful -sovellukset käyttävät HTTP-pyyntöjä lähettämään, lukemaan, päivittämään ja poistamaan tietoja. Perusoperaatiot ovat GET, POST, PUT ja DELETE.

REST:n avainkäsite on resurssi. Resurssiin viitataan osoitteella, URI (Uniform Resource Indicator). (Kumar 2015.) Jokainen järjestelmä käyttää resursseja. Nämä resurssit voivat olla mm. kuvia, videotiedostoja tai Web-sivuja. RESTful -palvelun tarkoituksena on tarjota asiakkaille mahdollisuus päästä käyttämään näitä resursseja. (Vaqqas 2014.)

2.4 Model View Controller

Ohjelmat, jotka ovat vuorovaikutuksessa käyttäjän kanssa tarvitsevat käyttöliittymän. MVC-malli (Model-View-Controller) on yleinen suunnittelumalli, jolla integroidaan käyttöliittymä sovellusalueiden logiikkaan. Siinä toimialueen malli ja ohjaimen logiikka on erotettu käyttöliittymästä. Tämän avulla sovelluksen ylläpito ja testaus on helppoa. MVC erottaa sovellusalueet ja esittää ne kolmena eri komponenttina: malli (model), näkymä (View) ja ohjain (Controller). Mallin tehtävä on tallentaa data, ohjain käsittelee käyttäjän toimia ja näkymä määrittää ulkoasun. (Syromiatnikov 2016.)

HTML-sivut tarjoavat syötteitä, joiden avulla käyttäjät voivat olla vuorovaikutuksessa järjestelmän kanssa, esim. hyperlinkkejä, lomakkeita, painikkeita. Tiedot käsitellään palvelimen puolella. Ohjaimella on pääsy sovellusalueiden logiikkaan: sen on mahdollista lukea ja muokata dataa, tehdä laskelmia jne. MVC erottaa näkymän ja ohjaimen vastuut. Tieto näytetään asiakaspäässä HTML-sivuna, kun taas logiikka palvelinpuolella, jossa käsitellään käyttäjän syötteitä. (Syromiatnikov 2016, 25.)

Malli on paikka, johon sovelluksen data tallennetaan. Se on usein dynaamista eli peräisin tietokannasta. Mallista tiedot lähetetään näkymiin. Näkymät näyttävät saadun tiedon. Ohjain toimii mallin ja näkymän välissä ja suorittaa toiminnot. (Mohammad & Golrakh 2015.)

3 Käytetyt tekniikat

3.1 AngularJS

AngularJS (tai Angular) on avoimen lähdekoodin Web-sovelluskehys. Sitä ylläpitävät Google ja yksittäiset kehittäjät sekä heidän yhteisönsä. Sillä pyritään yksinkertaistamaan asiakaspään MVC ja MVVM (model-view-viewmodel) -arkkitehtuureja noudattavien sovellusten kehittämistä ja testaamista. (Introduction to AngularJS Javascript Framework 2016.)

Se tarjoaa kaksisuuntaisen datan sitoutumisen mallien ja näkymien välille. Tämä mahdollistaa automaattisen päivityksen molemmin puolin aina, kun dataan tulee muutos. (Sayar 2015.) Se on suunniteltu luomaan yhden sivun Web-sovelluksia. Sen avulla pystytään kehittämään pieniä, kevyitä sovelluksia, joita on helppo rakentaa, testata ja ylläpitää niiden kasvaessa. (AngularJS 2016.)

AngularJS sisältää räätälöityjä HTML-tageja, joihin tiedot tuodaan suoraan mallista (Mohammad & Golrakh 2015). AngularJS tarjoaa toiminnallisuuden hallita käyttäjän syötteitä selaimessa ja tietoja asiakaspäässä (Dayley 2015). AngularJS voidaan liittää Web-sivuille <script> -tagilla. (AngularJS Introduction 2016).

3.2 HTML5

HTML (HyperText Markup Language) on avoin standardi kuvaamaan rakennetta ja sisältöä Web-sivuilla. HTML tarjoaa rakenteen Web-selaimille, jotta ne osaavat jäsentää sivun sisällön oikeanlaiseen muotoon. (HTML 2016.)

HTML 5 on uusin versio HTML-standardissa. Se sisältää elementtejä, määritteitä, käyttäytymisiä ja tekniikoita, jotka mahdollistavat monipuolisempien ja tehokkaampien Web-sivustojen ja -sovelluksien luomisen. (HTML5 2016.)

HTML 5:n avulla voidaan käsitellä graafista sisältöä verkossa ja siihen voidaan sisällyttää multimediaa. Kaikki sisältö voi olla upotettuna tai linkitettynä yhteen HTML-tiedostoon. Lisäksi se tarjoaa tuen JavaScript -ohjelmointirajapinnoille (API). (Segal 2016.)

Semanttisten tunnisteiden, kuten <article>, <section> ja <aside>, avulla voidaan jäsentää sisältöä niin, että tärkeimmät elementit voidaan sijoittaa lohkoiksi. Tämä on hyödyllinen hakukoneille ja syötteenlukijoille. HTML5 -tunnisteet ovat taaksepäin yhteensopivia niin, että HTML4 -pohjainen sisältö ei voi tuhota HTML5 -sisältöä. (Segal 2016.)

3.3 JavaScript

JavaScript on kevyt oliopohjainen kieli (Object-oriented language). Sen syntaksi perustuu Java- ja C -kieleen. JavaScript on ohjelmointikieli, joka ajetaan Web-selaimessa. JavaScript on laajalti käytetty Web-pohjaisissa ja itsenäisissä sovelluksissa. JavaScript sisältää kirjaston olioita, kuten Array ja Date, sekä kielen elementtejä, kuten operaattorit, ohjausrakenteet ja poikkeukset. JavaScript:n avulla pystytään lisäämään sivuille dynaamista sisältöä. (What is JavaScript 2016.)

Web-selaimeen on sisäänrakennettu tulkki, joka voi jäsentää ja suorittaa kieltä. JavaScriptin avulla voidaan kirjoittaa monimutkaisia sovelluksia, joilla on suora pääsy selaimen tapahtumiin ja Document Object Model (DOM) -olioihin. Pääsy DOM:iin tarkoittaa, että Web-sivulla voidaan lisätä, muokata tai poistaa osia lataamatta sitä uudelleen. (Dayley 2015)

JavaScript on asiakaspään (Client-side) ohjelmointikieli. Se on nopea, koska kaikkia funktioita voidaan ajaa ilman, että joudutaan lähettämään palvelimelle pyyntöjä tai odottamaan palvelimelta vastausta. JavaScript mahdollistaa korkean tason responsiivisen käyttöliittymän luomisen, koska käyttäjän ei tarvitse odottaa palvelimen vastausta. (Gandhi 2015.)

JavaScript on olio-ohjelmointikieli. Se eroaa muista oliopohjaisista kielistä (esim. C++, PHP, Java) siten, että siinä ei käytetä luokkia. (Gandhi 2015.)

JavaScript kuuluu dynaamisesti tyypitettyihin ohjelmointikieliin (Meijer & Drayton, 2004). Dynaamisesti tyypitetyssä ohjelmakoodissa olioiden tietotyyppiä ei määritetä muuttujalle. Muuttujan data voi olla missä tahansa muodossa ja tietotyyppi voi vaihtua ajon aikana. (Loui, 2008.) Tietotyypit voivat olla mm. numeroita, merkkijonoja, taulukoita tai olioita (Data Types 2016).

3.4 Mustache

Mustache on yksinkertainen mallijärjestelmä, jota voidaan käyttää Web-sivustoja ja sovelluksia rakennettaessa. Käyttämällä mallijärjestelmää voidaan pitää taustakoodi (back-end) erillään etupäästä (front-end), joka näkyy käyttäjälle. Näiden täysin erottaminen tarjoaa monia etuja. Sen avulla on helppo mm. suunnitella sivuston ulkoasu ilman, että vaikuttaisi sivuston koodiin. Suunnittelua voidaan vaihtaa myöhemmässä vaiheessa vaikuttamatta taustakoodiin (back-end). Mustache sisältää hyvin vähän logiikkaa, mikä tarkoittaa, että malli pysyy selkeänä ja siistinä. (Doyle 2012.)

Mustachessa ei käytetä if-else-lausekkeita tai -silmukoita, ainoastaan tageja. Tagit merkitään kahdella aaltosulkeella {{}}. Haluttu data tuodaan tagin sisälle. (Mustache Manual 2014.) Kaikki logiikka on piilossa oliossa, joka tuodaan aaltosulkujen sisään, ja koodissa, joka luo tai hakee niitä. Mustache ei ole sidottu mihinkään tiettyyn kieleen. Mallin prosessoreja on saatavilla useille eri kielille, kuten JavaScript, PHP, Perl, Ruby, Python ja Java. (Doyle 2012.)

3.5 Pattern Lab

Pattern Lab on työkalu staattisten Web-sivujen luontiin. Se on PHP-kieleen pohjautuva generaattori, joka kokoaa UI-komponentteja yhteen. Komponentit noudattavat atomista suunnittelua. Pattern Lab käyttää Musctache-mallipohjaa, johon HTML-koodi kirjoitetaan. Pattern Labiin voidaan tuoda dataa JSON tai YAML -muodossa. (Pattern Lab 2016.)

Pattern Lab on uusi työkalu. Se ei ole UI-sovelluskehys, kuten Bootstrap tai Foundation, eikä se korvaa sisällönhallintajärjetelmää esim. WordPress tai Joomla!. Pattern Lab ei ole kielestä, kirjastosta tai tyylistä riippuvainen. (Frost 2016.)

3.6 Sass

Sass (Syntactically Awesome Style Sheets) on laajennus CSS-kielestä. Se manipuloi muuttujia, kuten sisäkkäisiä sääntöjä, sekoituksia, valitsimia ja periytymisiä. Sass:lla on kaksi syntaksia SCSS ja Sass. Vanhempi syntaksi tunnetaan sisennettynä syntaksina tai vain Sass:na. Sulkujen ja puolipisteiden sijaan se käyttää rivien

sisennystä määrittämään lohkoja. Vaikka se ei ole enää ensisijainen syntaksi, se tulee olemaan tuettu jatkossa. (Sass 2016.)

Uusin syntaksi tunnetaan nimellä SCSS (Sassy CSS). Se käyttää sulkuja ja puolipisteitä niin kuin CSS-kieli. Se ei välitä sisennystasoista tai tyhjästä tilasta. SCSS-syntaksi sisältää kaikki CSS-kielen ominaisuudet ja myös Sass:n ominaisuuksia. (Long 2011).

Laajennukset, Sass ja SCSS, muunnetaan hyvin muotoiltuun CSS-kieleen käyttämällä komentorivityökalua tai Web-sovelluskehyksen liitännäistä. (Long 2011).

3.7 SoundManager2

SoundManager2 on ohjelmointirajapinta (API), joka käyttää HTML5 -audiota ja Flash:ia. Se mahdollistaa audion toistamisen JavaScriptin avulla. (How SoundManager Works 2016.) Javascript API:n avulla voidaan lukea dyynaamisesti asetettuja audio-asetuksia, esim. tiedoston koko, kesto ja kuinka suuri osa tiedostosta on latautunut. SoundManger -objektissa on pääsy selaimen tapahtumiin. Näitä voidaan liittää osaksi koodia silloin, kun muutoksia tapahtuu audiossa esim. toistetaan, pysäytetään tai audioraita päättyy. (Odell 2009.)

SoundManger tarjoaa myös ylimääräisiä toimintoja loppukäyttäjille, joilla on asennettuna Flash Player versio 9 tai uudempi. Silloin käyttäjälle on mahdollista näyttää enemmän visuaalista infromaatiota mm. soitetun audioraidan aaltomuotoja tai taajuuskorjain. (Odell 2009.)

SoundManager on avoimen lähdekoodin ohjelmisto ja vapaa käytettäväksi sekä henkilökohtaisissa että kaupallisissa hankkeissa. Sen on kehittänyt Scott Schiller ja see toimii BSD-lisenssin alla. (Odell 2009.)

4 Sovelluksen suunnittelu

4.1 Arkkitehtuurinen suunnittelu

Toteutin sovelluksen noudattamalla perinteisen vesiputousmallin (waterfall model) vaihejakoa. Sen vaiheet ovat:

- Määrittely
  vaatimusmäärittely eli asetetaan kehitettävälle sovellukselle
  vähimmäisvaatimukset ja rajat
- Suunnittelu
  miten sovellus toteutetaan, arkkitehtuurinen suunnittelu
- Toteutus
  ohjelmointi
- Testaus
- Käyttöönotto
  Osien yhteen liittäminen: Integrointi- ja järjestelmätestaus
- Ylläpito (Rintamäki 2017, 5-7).

Vaatimusmäärittely tehtiin yhdessä toimeksiantajan kanssa ja nämä rajasivat tehtävääni. Sovelluksen vaatimukset on kuvattu tarkemmin kappaleessa 1.1.

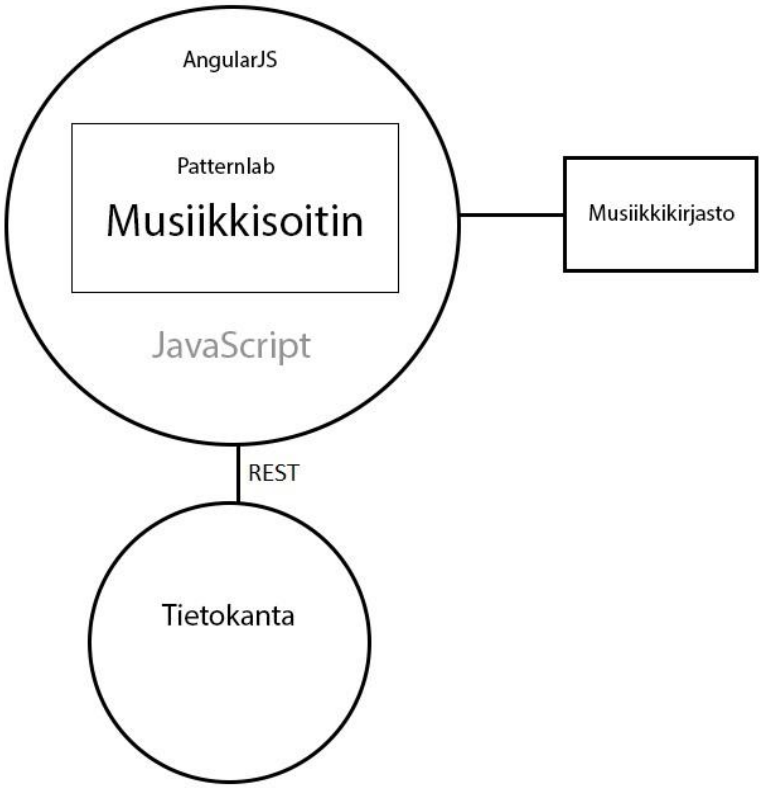
Suunnitteluvaiheessa tutkin erilaisia tapoja toteuttaa responsiivinen musiikkisoitin. Musiikkisoittimen kehittäminen on mahdollista käyttämällä perinteisiä Web-tekniikoita, kuten HTML5, JavaScript ja CSS.

Kaikkein yksinkertaisin tapa toteuttaa soitin olisi käyttää pelkästään HTML5-audiota, joka tukee MP3, Ogg ja Wav-formaatteja (HTML5 Audio 2016). Soittimesta tulisi tällä tekniikalla toiminnan kannalta kevyt, koska se olisi nopea ladata. HTML5-audio tukee kuitenkin vain yleisimpiä formaatteja ja sille on rajoittunut tuki selaimissa.

Soitin ei voi kuitenkaan olla rajoittunut pelkästään yleisimpiin tiedostomuotoihin. Musiikki pitäisi pystyä toistamaan eri selaimilla ja eri tiedostotyyppien pitäisi olla

tuettuja. Järkevämmäksi tavaksi toteuttaa sovellus osoittautui erillisen sovelluskirjaston (API) käyttäminen, mikä mahdollistaa useiden formaattien toistamisen.

Sovelluksen suunnittelin toteutettavaksi seuraavan mallin mukaisesti.



Kuvio 2. Sovelluksen arkkitehtuurikaavio

Musiikkisoitinsovelluksen toiminnallisuus luodaan AngularJS-sovelluskehystä käyttämällä ja Pattern Lab -työkalun avulla kootaan elementit yhteen. Data haetaan tietokannasta REST:in välityksellä. Musiikkikirjasto tarjoaa tarvittavat soittimen toiminnot musiikin toistamiseen.

Testaaminen liittyy olennaisesti kehitystyöhön. Sovelluksen komponenttien toimintaa tullaan testaamaan, jotta ne saataisiin toimimaan yhdessä. Toimeksiantajan kanssa sovittiin, että soitinsovelluksen integroiminen MusicInfo-palveluun jää toteutettavaksi myöhemmin. Ylläpito ja edelleen kehittäminen jäävät myös toimeksiantajalle.

4.2 Musiikkikirjaston valinta

Musikkisovelluksen kehittämisen edellytyksenä on, että valittu tekniikka ja työkalut ovat yhteensopivia MusicInfo-palvelun arkkitehtuurin kanssa. MusicInfo-palvelu käyttää AngularJS-sovelluskehystä toimintojen suorittamiseen. Aloitin työn tutkimalla, millaisia musiikkikirjastoja on olemassa ja mitkä niistä soveltuisivat tähän kehitystyöhön.

Web-selaimet ovat viime vuosina kehittyneet paljon uuden HTML5-teknologian avulla. Musiikkikirjastoja on useita, mutta ongelmana on, että vain uusimmat selaimet tukevat audiota. JavaScript-kirjastojen avulla pystytään hallitsemaan audion olion metodeja, ominaisuuksia ja tapahtumia, joiden avulla voidaan manipuloida ääni ja videoelementtejä. JavaScript-kirjastot ovat yhteensopivia AngularJS:n kanssa.

Puhtaalla JavaScriptillä kirjoitetuista kirjastoista Howler.js oli yksi hyvistä vaihtoehdoista. Se tarjoaa tuen monelle tiedostomuodolle ja on testattu toimivaksi yleisimmissä selaimissa.

Päädyin kuitenkin valitsemaan AngularJS:lle tehdyn musiikkikirjaston, koska AngularJS:n tarjoaa ominaisuuksia, joiden avulla voidaan kehittää toimintoja pidemmälle ja tehdä niistä sopivia omiin tarpeisiin. Löysin kaksi potentiaalista kirjastoa: Angular media player ja SoundManager2.

Vertailussa päädyin SoundManager2-kirjastoon. Tämä kirjasto on hyvin dokumentoitu. Koin dokumentaation vahvaksi tueksi sovelluksen kehittämisessä.

Angular Media playerin heikkoutena koin, että sen toiminta on rakennettu html audio- ja video -tagin ympärille. Tämän vuoksi Web-sivun sisältöä ei voi päivittää dynaamisesti. Sivulla olevaa dataa ei pysty vaihtamaan reaaliajassa.

5 Sovelluksen kehittäminen

5.1 Kehitysympäristö

Ennen soittimen rakentamista täytyy olla palvelin, jolla voidaan ajaa ohjelmaa. Tähän voidaan käyttää esimerkiksi XAMPP- tai WAMP-kehitysympäristöjä, jotka mahdollistavat ohjelmien ajamisen paikallisesti Web-palvelimelta.

Pattern Lab -projektin sain ladattua GitHubin -verkkosivulta, osoitteesta: https://github.com/pattern-lab/patternlab-php. XAMPP-kehitysympäristön pystytys tapahtui kopioimalla projektin juurihakemisto htdocs (C:\xampp\htdocs) -kansion sisälle. Pattern Labin tiedostorakenne on seuraava:

Kuvio 3. Pattern Labin tiedostorakenne

Muokattavat Mustache-tiedostot sijaitsevat hakemistossa source/_patterns. Kansiot on nimetty atomisen suunnittelun mukaisella tavalla. Pattern Lab koostaa source-kansion HTML-tiedostoiksi public-kansioon. Sivun saa näkymään avaamalla /public/index.html -tiedoston verkkoselaimessa. Kaikki muutokset, jotka tehdään source-hakemistossa, täytyy uudestaan koota sivuksi ja ladata verkkoselaimessa.

Sivun kokoamiseksi täytyy ajaa seuraavat komennot:

- call compass compile source
- php core\builder.php -g.

5.1.1 Kirjastojen lisääminen

Pattern Lab sisältää ylä- ja alatunnisteen tiedostot, joihin lisätään halutut kirjastot. Nämä vastaavat samoja tageja kuin HTML-sivun <head> ja <footer>. Tiedostot löytyvät kansiosta source/_patterns/00-atoms/00-meta.

Kehitystyössä käyttämäni JavaScript -kirjastot ovat:

- angular
- angular-soundmanager2
- angular-drag-and-drop-lists
- mustache.

Angular ja mustache-kirjastot lisätään _00-head.mustache -tiedostoon. Kaikki muut kirjastot lisätään _01-foot.mustache-tiedoston loppuun, koska nämä tiedostot ladataan viimeisenä.

5.2 Sovelluksen toimintojen kehittäminen

Kehitysympäristön lisäksi on sovelluksen kehittämiseksi oltava tietokantapalvelin tai erillinen testitiedosto, josta voidaan tuoda dataa käyttöliittymään. Tässä kehitystyössä käytin MusicInfo-palvelun valmista tietokantaa datan hakemiseen.

5.2.1 AngularJS-moduuli

Pattern Labin tiedostorakenteessa JavaScript-tiedostot löytyvät source/js-kansiosta. Musiikkisoittimen tiedosto on nimetty player-app.js-tiedostoksi, jonka sisälle toteutin kaikki sovelluksen toiminnallisuudet.

AngularJS:lle täytyi määrittää moduuli, jotta sovelluksen toimintoja voidaan käyttää. Moduuli on ikään kuin kontti, joka sisältää sovelluksen ohjaimet, palvelut, suodattimet, direktiivit, jne.

Moduulin nimesin MusicInfoksi alla olevalla tavalla:

```
var MusicInfo = angular.module('MusicInfo', []);
```

Kuvio 4. AngularJS-moduulin luonti

5.2.2 Angular Factory

Jotta päästään käsiksi musiikkitietoon, täytyy kommunikoida palvelimen kanssa HTTPS-yhteyden välityksellä. AngularJS:ssä voidaan määrittää palvelu tähän tarkoitukseen. AngularJS:ssä on kaksi tapaa määrittää palvelu, service ja factory. Näistä käytin jälkimmäistä. Oletusmetodien sijaan käytin datan hakemiseen HTTP-palvelua. HTTP-palvelu on yksi AngularJS:n palvelu datan lukemiseksi etäpalvelimelta. HTTP-palvelu tekee pyynnön palvelimelle ja palauttaa vastauksen.

HTTP-palvelu on rakennettu REST-verbejä käyttämällä. Metodit datan hakuun ovat: $http.get(), $http.post(), $http.put(), $http.delete(). Tässä kehitystyössä käytin ainoastaan $http.get() -metodia.

Factoryn sisään voidaan kirjoittaa funktioita. Olen toteuttanut datan haun tekemällä oman funktion julkaisulle (getAlbum) sekä artistille (getArtist). Palvelun olen nimennyt musicFactory:ksi. Julkaisun haku tapahtuu seuraavalla tavalla.

```
MusicInfo.factory("musicFactory", function ($http, $q) {

                // Palauttaa arvon
                return({
                    getAlbum: getAlbum,
                });

                // Haetaan julkaisu tietokannasta
                function getAlbum(id) {
                    var request = $http({
                        method: "get",
                        url: 'https://musicinfocloud.net/preview/v1/release/' + id + '/',
                         headers: {
                            'Content-Type': 'application/json',
                            'Accept': 'application/json'
                        },
                        params: {
                            action: "get"
                        }
                    });

                    return( request.then( handleSuccess, handleError ) );
                }

});
```

Kuvio 5. Julkaisun hakeminen tietokannasta

5.2.3 Angular Controller

Kun sain HTTP-palvelun luotua, vein datan ohjaimeen (controller), joka käyttää palvelua toimintojen suorittamiseksi. Ohjain suorittaa kaikki sovelluksen toiminnot. Lopullinen tulos tuodaan HTML-sivulle. Ohjaimen nimesin MainCtrl:ksi.

Ohjaimeen tein funktiot, joiden avulla sain haettua julkaisun ja artistin. Julkaisun saa haettua kutsumalla musicFactory.getAlbum() ja artistin kutsumalla musicFactory.getArtist().

```
MusicInfo.controller('MainCtrl', ['musicFactory','$scope' , function (musicFactory, $scope){

    // Haetaan julkaisu
    musicFactory.getAlbum(album).then(function (tracks) {
                $scope.albumData = tracks;
        })
    // Haetaan artisti
    musicFactory.getArtist(artist).then(function (albumArtist) {
            $scope.artistData = albumArtist;
        })
    }


}]);
```

Kuvio 6. Datan vienti ohjaimeen

Näin sain julkaisun ja artistin datan käyttööni. Niistä pystyin erittelemään tietoja muuttujiksi.

5.2.4 Uuden julkaisun lisääminen

Toistettavien kappaleiden tulee näkyä käyttöliittymässä ja niitä on pystyttävä toistamaan. SoundManager2:lla on oma taulukko, johon toistettavat kappaleet lisätään. Käyttöliittymässä näkyville tiedoille tein oman taulukon, tracks.

Julkaisun kappaleiden data on lisättävä sekä SoundManger2 että tracks -taulukkoon. Kappaleiden lisäämiseksi tein oman funktion. Ensin kappaleiden data loopataan eli silmukka käy läpi tiedot. Kappaleet lisätään sen jälkeen SoundManager2:n taulukkoon eri muuttujineen (id, title, artist, length, url). Lopuksi samat tiedot lisätään tracks-taulukkoon.

```
function addNewAlbum(data) {

    $timeout(function () {
        for (var i = 0; i < data.length; i++) {
            // Luodaan objektit kappaleista
            $scope.newTracks = soundManager.createSound({
                id: data[i].id,
                title: data[i].name,
                artist: data[i].artist_display_name,
                length: data[i].preview_length,
                url: data[i].preview_url
            });
            // Lisätään kappaleet tracks-taulukkoon
            $scope.tracks.push($scope.newTracks.options);
        }
    }, 30);
}
```

Kuvio 7. Kappaleiden lisääminen

5.2.5 Virheidenkäsittely

Jotta voidaan käsitellä palvelimen virheilmoituksia haettaessa dataa, oli määriteltävä virheidenkäsittely. Interceptor on factoryn palvelu, jota voidaan käyttää HTTP-pyyntöjen ja -vastauksien käsittelyyn. Interceptor palauttaa objektin, joka sisältää neljä ominaisuutta:

- request: kutsutaan ennen kuin pyyntö lähetetään
- requestError: kutsutaan, jos pyyntö epäonnistuu
- response: kutsutaan, kun HTTP-pyyntö onnistuu
- responseError: kutsutaan, jos HTTP-metodi epäonnistuu. ($Http 2017.)

Sain objektin käyttöön rekisteröimällä sen config-osioon.

```
MusicInfo.config(['$httpProvider', function ($httpProvider) {

var interceptor = ['$q', function($q) {
        return {
            request: function (config) {
                return config;
            },
             requestError: function(rejection) {
                return $q.reject(rejection);
            },
            response: function (response) {
                return response || $q.when(response);
            },
            responseError: function(rejection) {
                console.log(rejection);
            }
        };
    }];

    $httpProvider.interceptors.push(interceptor);
}]);
```

Kuvio 8. Virheidenkäsittely

5.3 SoundManager2-kirjaston toimintojen kehittäminen

SoundManager2-kirjaston ominaisuudet sain käyttöön AngularJS-sovelluskehyksessä määrittämällä tämän kirjaston moduulin MusicInfo-moduulin sisälle. Moduulin nimi on angularSoundManager.

```
MusicInfo = angular.module('MusicInfo', ['angularSoundManager']);
```

Kuvio 9. SoundManager2-moduulin lisääminen

Kirjaston tarjoamia valmiita funktioita ovat esim. play(), pause(), nextTrack(), prevTrack(). Käytin näitä valmiita funktioita musiikkisoittimen toisto-painikkeissa.

```
<div class="PlayControls">
    <button class="PlayControls-previous" prev-track>
        <svg viewBox="0 0 100 100"><use xlink:href="#skip-backward"></use></svg>
    </button>
    <button class="PlayControls-play" play-pause-toggle>
        <svg viewBox="0 0 100 100" ng-if="isPlaying==false">
            <use xlink:href="#play"></use>
        </svg>
        <svg viewBox="0 0 100 100" ng-if="isPlaying==true">
            <use xlink:href="#pause"></use>
        </svg>
    </button>
    <button class="PlayControls-next" next-track>
        <svg viewBox="0 0 100 100">
            <use xlink:href="#skip-forward"></use>
        </svg>
    </button>
</div>
```

Kuvio 10. Toisto/tauko, seuraava ja edellinen -painikkeet

SoundManager2-kirjasto tarjoaa myös tapahtumia (event), joiden avulla voidaan seurata soittimen toimintaa. Tapahtumia käytetään yhdessä toimintojen kanssa. Toimintoja ei suoriteta ennen kuin tapahtuma esiintyy, esimerkiksi kun käyttäjä painaa painiketta. Näiden tapahtumien arvot määritetään muuttujiksi ja tuodaan soittimeen. Hyödynsin kirjaston valmiita tapahtumia, kuten currentTrack:position, currentTrack:duration, music:volume ja track:progress.

```
<div class="DurationInfo">
    <div class="DurationInfo-position" ng-bind="currentPosition"> </div>
    <div class="DurationInfo-duration" ng-bind="currentDuration"> </div>
</div>
```

Kuvio 11. Toistettavan kappaleen ajankohta ja kesto

Kirjasto tarjoaa myös valmiita direktiivejä. Direktiivien liittäminen HTML-sivulle tekee sivusta interaktiivisen. Soittimessa näytetään edistymispalkki. Seek-track -direktiivi seuraa kappaleen tarkkaa ajankohtaa.

```
<div class="seekBase" seek-track>
    <div class="seekLoad" ng-style="{ 'width': progress + '%' }"></div>
</div>
```

Kuvio 12. Seek-track -direktiivi

Volume-palkissa asteikko on 0-100 välillä. Kehitin volume-slider -direktiivin seuraamaan äänenvoimakkuuden muutoksia. Palkin leveyden määritin prosentteina.

```
<div class="volumeBarContainer">
  <div class="progress-box">
    <div class="progress-bar-container">
      <div class="volume-text" ng-bind="volume"></div>
        <div class="progress-bar progress-bar-slider">
          <input class="progress-slider" type="range" min="0" max="100" ng-model="volume" volume-slider>
            <div class="inner" ng-style="{ width: volume + '%' || '0%' }"></div>
          </div>
      </div>
    <button class="volume-button">
      <svg viewBox="0 0 512 512">
        <use xlink:href="#volume-medium"></use>
      </svg>
    </button>
  </div>
</div>
```

Kuvio 13. Volume-slider -direktiivi

Mukautin myös sekoitus (shuffle) sekä jatkuvan toiston (repeat) -toimintoja.

```
<div class="OptionalPlayControls">
    <button class="shuffle-btn" shuffle-music>
        <svg viewBox="0 0 512 512">
            <use xlink:href="#shuffle"></use>
        </svg>
    </button>
    <button class="repeat-btn" repeat-music>
        <svg viewBox="0 0 512 512">
            <use xlink:href="#loop"></use>
        </svg>
    </button>
</div>
```

Kuvio 14. Shuffle ja repeat -direktiivit

5.4 Sivupalkin tietojen esittäminen

Musiikkisoitinsovellus pitää sisällään sivupalkin, jossa näytetään julkaisun tietoja. Sivupalkkiin ladataan lisätietoja julkaisusta ja artistista.

Toin ensimmäiseksi sivupalkkiin yleiset tiedot: kansitaide (cover), kappaleen nimi (title) ja artisti (artist). Mukautin AngularJS:n syntaksia niin, että merkintätavan {{}} sijasta käytin {$ $} -merkintää. Tällä tavalla pystyin erottamaan AngularJS:n tiedot mustache-mallipohjan tiedoista.

```
<section class="AlbumInfo">
    <figure class="AlbumInfo-cover">
        <img class="AlbumInfo-image" ng-src="{$ selectedTrackImage $}"
            on-error-src="{$ default_image $}"/>
    </figure>
        <div class="nowPlaying">
            <p class="np-title" ng-bind="selectedTrackName"></p>
            <p class="np-artist" ng-bind="selectedTrackArtist"></p>
        </div>
```

Kuvio 15. Sivupalkin yleiset tiedot

Sen jälkeen liitin yksityiskohtaisemmat julkaisun tiedot: julkaisu, vuosi, kappaleiden määrä, formaatti, tyyppi, kotimaa, genre ja levy-yhtiö. Lisäsin myös muita tietoja, kuten tagit, ostolinkit, artistin historia ja linkit sosiaalisen median kanavista.

```
<section class="Social-networking">
    <h2 class="Social-networking-header"> Social Profiles </h2>
        <ul class="social" >
            <li ng-repeat="sMedia in results" id="{$ sMedia.name $}">
                <a class="social-icon" ng-href="{$ sMedia.url $}" target="_blank" >
                  <svg><use xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="{$ sMedia.svg $}"></use></svg>
                </a>
            </li>
        </ul>
</section>
```

Kuvio 16. Esimerkki, sosiaalisen median linkkien lisääminen

5.5 Käyttöliittymän kokoaminen

Kun olin saanut tarvittavat musiikkisoitinsovelluksen osat rakennettua, yhdistin ne mallipohjaan. Aiemmissa kappaleissa kerroin molekyylitason osien rakentamisesta. Yhdessä ne muodostavat isomman kokonaisuuden, organismin.

Mustache-mallipohjaan saadaan sisällytettyä malleja syntaksilla {{> malli }}. Molekyylitason mallit tuodaan organismitasolle. Esimerkiksi sivupalkin tuonti organismiksi tapahtuu seuraavalla tavalla:

```
<aside id="Sidebar">
    <div class="Sidebar-menu">
        <div class="Sidebar-content">
            {{> molecules-AlbumInfo }}
            {{> molecules-TagsList }}
            {{> molecules-Affiliates }}
            {{> molecules-ArtistDescription}}
            {{> molecules-SocialNetworking }}
        </div>
    </div>
</aside>
```

Kuvio 17. Sivupalkin kokoaminen organismiksi

Kokosin lopuksi organismitason mallit yhdeksi sivuksi tuomalla ne lopulliseen sivupohjaan (Pages).

```
<div class="Player-page" ng-controller="MainCtrl">
    <div class="Player-wrapper">
        {{> organisms-Sidebar }}
        {{> organisms-Playlist }}
        {{> organisms-Player }}
    </div>
</div>
```

Kuvio 18. Organismitason mallien tuominen sivulle

6 Sovelluksen testaaminen

Testaaminen oli kehitystyön jatkuva prosessi. Valtaosin tein yksikkötestaamista eli testasin käyttöliittymän komponenttien toimintaa. Kun kehitin musiikkisoitinsovelluksen toiminnallisuutta AngularJS:lle, testasin samalla luomiani toimintoja käyttöliittymässä. Komponenttien testaaminen oli mahdollista aina, kun sain käyttöliittymän sivulla näkymään tiedon ja painikkeen kullekin toiminnolle. Käytin kappaleiden toistamiseen ja tiedon hakemiseen ensin fyysistä testitiedostoa, josta sain testattavaa dataa tuoduksi käyttöliittymään. Kehitystyön kuluessa pystyin Pattern Labin sivujen avulla testaamaan ja näkemään, kuinka sivun palaset sopivat ja toimivat yhteen.

Kun kehitystyössä etenin mallitasolle ja loin toimivan mallin soittimesta, pystyin tuomaan käyttöliittymään oikeaa esiteltävää sisältöä. Testasin soittimen toimintaa hakemalla MusicInfon tietokannasta JSON-muodossa tietyn artistin julkaisuja. Tietokantaan pääsin käsiksi HTTP-kutsuja käyttäen. Osoite, josta hain dataa, on muodossa https://musicinfocloud.net/preview/v1/release/:id.

Musiikkisoittimen käyttöä testattaessani, hakiessani dataa, havaitsin, että tietokannan rakennetta oli muokattava järkevämpään muotoon ja lisättävä uusia viittauksia. Tämä tietokannan viittausten korjaaminen hidasti käyttöliittymän kehittämistä. Soittimen toimintojen testaamista voitiin jatkaa vasta, kun tietokannan viittaukset oli korjattu. Testatessani tutkin myös, onko tietokannassa samaa tietoa useassa paikassa.

Musiikkisoitinsovelluksen suorituskyvyn testaaminen oli myös tärkeää. Testaamisella pyrin selvittämään, kuinka:

- nopesti sivu latautuu
- suurta määrää tietoa sovellus pystyy käsittelemään sitä käytettäessä
- nopeasti data saadaan ladattua sivulle
- nopeasti sivun tiedot päivittyvät.
- käyttäjää tiedotetaan, jos dataa ei löydy.

7 Tietoturvan huomioiminen

Aina, kun sovellus tekee pyyntöjä palvelimelle, altistutaan tietoturvariskeille. Tämä on otettava huomioon ja pyrittävä estämään, kun sovellusta kehitetään. Uhkien torjumiseksi palvelimen sekä asiakkaan täytyy tehdä yhteistyötä. AngularJS-sovelluskehyksen konfiguraatiossa on osa tietoturvariskeistä jo huomioitu.

7.1 Cross Site Scripting

Cross Site Scripting (XSS) on haavoittuvuus, joka mahdollistaa sen, että hakkerit voivat lisätä omaa koodia esim. ponnahdusikkunoita sovelluksen sivuille tai ohjata käyttäjiä linkeistä omille sivuilleen. Scriptit suoritetaan tahallisesti käyttäjän selaimessa, koska selain luottaa sisällön lähteeseen, vaikka se ei tulisikaan sieltä mistä pitäisi. (Booth 2011.)

Content Security Policy (CSP) -asetus mahdollistaa, että palvelimien ylläpitäjät voivat vähentää tai poistaa nämä vektorit, joissa XSS-iskuja voi tapahtua. Tämä onnistuu määrittämällä verkkotunnukset, jotka selain tulkitsee luotettaviksi suorittamaan skriptejä. Tarkoituksena on saada selain suorittamaan vain niitä komentosarjoja, jotka ladataan luotettaviksi merkityistä lähteistä, hyläten kaikki muut skriptit. (Medley 2017.)

Määritin CSP-asetukset sallimaan ainoastaan HTTPS-alkuiset osoitteet lisäämällä seuraavan META-tagin HTML-dokumentin head-osioon:
<meta http-equiv="Content-Security-Policy" content="default-src https:">

7.2 Cross Site Request Forgery

Cross Site Request Forgery (CSRF) suoritetaan käyttäjän verkkoselaimessa. Haavoittuvuus mahdollistaa, että hyökkääjä voi pakottaa selaimen toteuttamaan sovelluksen toiminnot vilpillisesti. Se käyttää hyväksi käyttäjän istuntoa sovelluksessa. (Booth 2011.)

Omassa musiikkisoitinsovelluksessani CSRF ei ole ongelma silloin, kun resursseja haetaan palvelimelta. Mutta kun käytetään POST, PUT tai DELETE-metodeja, palvelimen tila muuttuu ja silloin CSRF-haavoittuvuus on mahdollinen. Hyökkääjien yritys päästä

käsiksi käyttäjän istuntoon väärentämällä HTTP-pyyntöjä estetään tokenien avulla. Tokenit ovat pitkiä salauksen arvoja, joita on vaikea arvata. Jokaiseen palvelimelle tulevaan pyyntöön sisällytin tokenin. Näitä käytetään tarkistamaan loppukäyttäjän oikeudet. Kun palvelimelle lähetetään pyyntö, palvelin vertaa oikeaa arvoa HTTP-parametrissa olevaan arvoon. Jos arvo ei täsmää, pyyntö epäonnistuu.

7.3 Cross-origin resource sharing

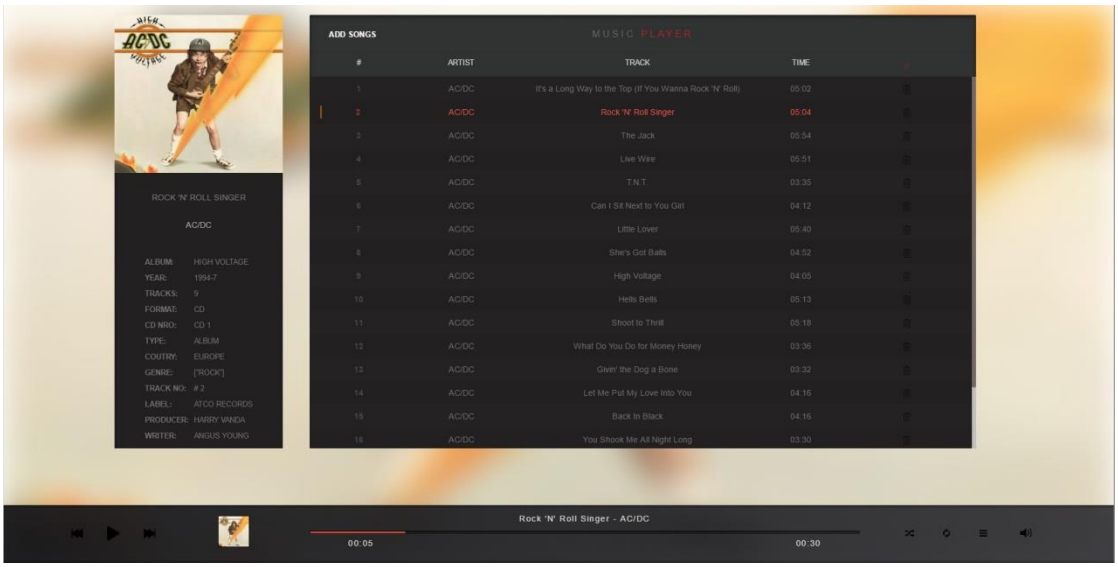
Cross-origin resource sharing (CORS) on mekanismi, joka sallii resurssien pyytämisen toimialueen ulkopuolelta. Se mahdollistaa, että palveluun pääsee eri verkko-osoitteesta kuin missä palvelin sijaitsee. Se toimii lisäämällä uusia HTTP-otsikoita. Nämä kertovat palvelimelle osoitteet, jotka ovat oikeutettuja lukemaan tietoa Web-selaimella. CORS määrittää tavan, jolla Web-selain ja -palvelin voivat olla yhteydessä toisiinsa. (AlexKjes 2017.)

HTTP-protokollan otsikko pitää sisällään X-Requested-With -kentän, jota käytetään pääasiassa tunnistamaan Ajax-pyyntöjä. AngularJS-sovelluskehys lisää tämän kentän yleensä HTTP-pyyntöön. Ratkaisin ongelman poistamalla otsikon HTTP-pyynnöstä:

```
myModule.config(function($httpProvider){
   delete $httpProvider.defaults.headers.common['X-Requested-With'];
});
```

8 Valmis musiikkisoitinsovellus

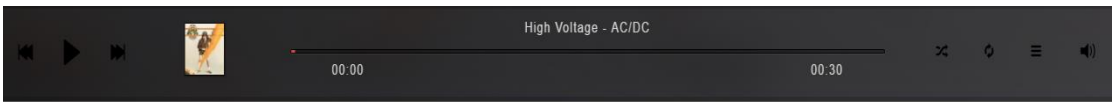
Valmis käyttöliittymä sisältää kolme päänäkymää: soitin, soittolista ja sivupalkki.



Kuvio 19. Sovelluksen käyttöliittymä

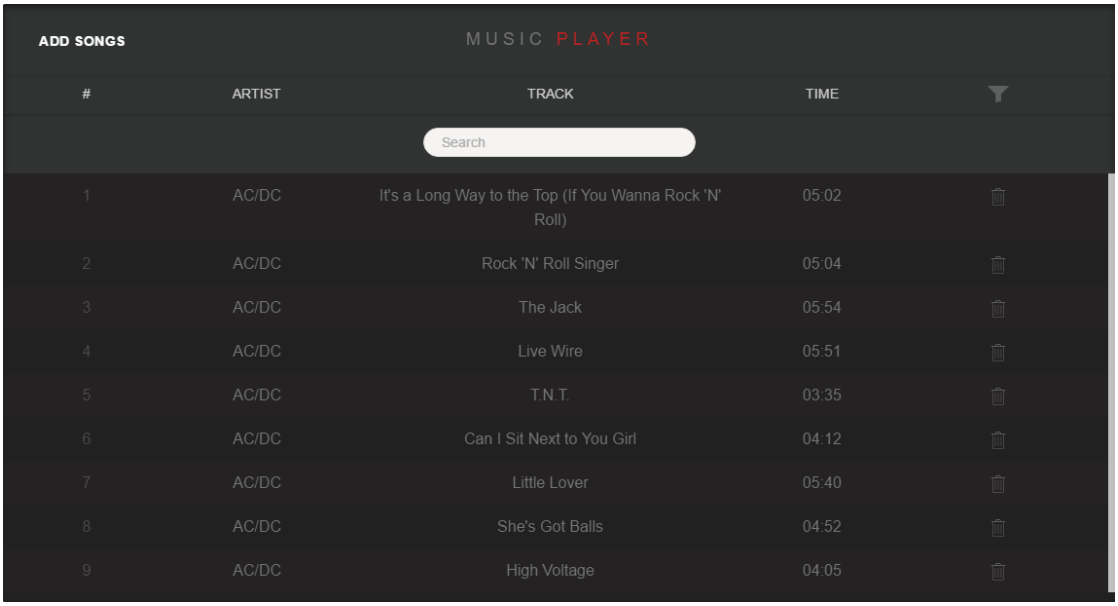
Soitin näyttää kappaleen tilan: nykyisen kappaleen, artistin, kuluneen ajan ja keston. Soitin toistaa oletuksena kappaleita soittolistan mukaisesti. Kappaleiden toistaminen loppuu, kun soittolistan viimeinen kappale päättyy.

Soittimessa voidaan toistamisen lisäksi pysäyttää kappale, valita edellinen tai seuraava ja säätää äänenvoimakkuutta. Edistymispalkin avulla käyttäjä voi siirtyä toistettavassa kappaleessa eri ajankohtaan. Soittolistan kappaleita voidaan sekoittaa tai toistaa jatkuvasti. Soittimessa on valintamahdollisuutena näyttää sivupalkki tai piilottaa se.



Kuvio 20. Soitin

Soittolistaan käyttäjä voi lisätä kappaleita, poistaa niitä, käyttää hakutoimintoa etsimään listasta tiettyä kappaletta tai järjestää kappaleet uuteen järjestykseen.



Kuvio 21. Soittolista

Sivupalkki näyttää julkaisun kansitaiteen sekä lisätietoja esim. julkaisun tuottajan ja kappaleen sanoittajan.

Kuvio 22. Sivupalkki

9 Tutkimuksen tulokset

Tutkimuksen tuloksena kehitin musiikkisoitinsovelluksen, joka on yhteensopiva toimeksiantajan, MusicInfo Finland Oy:n, palveluun. Opinnäytetyötä aloittaessani määrittelimme toimeksiantajan kanssa yhdessä vähimmäisvaatimukset sovellukselle. Palvelu on aiemmin tarjonnut yksinomaan musiikkitietoa. Kehittämistyön avulla luotiin musiikkisoitin täydentämään palvelua.
Kehittämistyöni eteni seuraavasti:

1. listasin yhteistyössä toimeksiantajan kanssa sovelluksen vaatimukset
2. suunnitelin sovelluksen arkkitehtuurimallin
3. valitsin kehitystyöhön sopivan JavaScript-kirjaston
4. kehitin toiminnon, jolla musiikkitiedot saa haettua tietokannasta
5. toteutin sovelluksen käyttöliittymän
6. kehitin tarvittavat soittimen toiminnot kirjastoa hyödyntäen
7. testasin sovelluksen toimintaa
8. korjasin löydetyt ohjelmointivirheet
9. kartoitin mahdolliset tietoturvariskit.

Aloitin kehittämistyön suunnittelemalla mallin sovelluksen karkeasta rakenteesta, sen pääkomponenteista ja rajapinnoista. Tämän pohjalta lähdin toteuttamaan työtä. Tarvitsin soittimen toteuttamiseksi musiikkikirjaston. Valitsin SoundManager2-kirjaston, koska sen avulla sovelluksen toimintoja voidaan kehittää edelleen ja tehdä niistä sopivia omiin tarpeisiin. Kirjasto tukee dynaamisesti päivitettävää sisältöä käyttöliittymässä. Tämä kirjasto on lisäksi hyvin dokumentoitu.

Käyttöliittymän toteutuksessa keskityin rakentamaan yksittäisiä komponentteja, joista myöhemmin kokosin musiikkisoitinsovelluksen. Käytin Pattern Lab -työkalua käyttöliittymän kehittämiseksi. Toteutin tiedon hakemisen ja sen tuomisen käyttöliittymään AngularJS-sovelluskehystä käyttäen. Musiikkitieto haetaan MusicInfon tietokannasta REST-rajapinnan kautta JSON-muodossa.

Kehittämistyö eteni niin, että testasin jatkuvasti toimintoja niitä kehittäessäni. Testaamisen avulla pystyin korjaamaan löytyneitä vikoja ja ongelmakohtia. Lopuksi minun oli ratkaistava sovelluksen käyttöön liittyviä tietoturvaongelmia, joihin törmätään, kun tietoa haetaan palvelimelta.

Musiikkisoitinsovellus tulee julkaistavaksi MusicInfo-palvelun uuden version myötä. Kehittämäni sovelluksen kautta käyttäjä pystyy kuuntelemaan musiikkia ja lukemaan kyseiseen artistiin ja albumiin liittyvää tietoa. SoundManger2-kirjasto tarjoaa soittimen toiminnot: Play, Pause, Repeat, Shuffle, Volume.

Jatkossa soitinta on mahdollista kehittää edelleen niin, että siinä voidaan mm. toistaa MPEG-4 -videota, näyttää kappaleen audioraidan aaltomuotoja ja säätää toistoasetuksia taajuuskorjaimella. Lisäksi siihen voidaan liittää muita sovellusrajapintoja (API) esim. SoundCloud.

Tämä opinnäytetyö toimii dokumentointina yrityksen uusille työntekijöille. Sovelluksen tekniikkaa voidaan hyödyntää samantyyppisessä sovelluksen kehittämistyössä. Tulokset ovat todennettavissa.

10 Pohdinta

Kehittämistutkimus tutkimusmenetelmänä soveltui Music.Info Finland Oy:n toimeksiantoon erinomaisesti, koska yrityksellä ei ollut olemassa olevaa sovellusta musiikin toistamiseen. Kehitystyön avulla palvelun tarjontaa pystytään laajentamaan musiikin kuunteluun. Palvelun arkkitehtuuri ja käytetyt tekniikat asettivat rajat sovelluksen toteuttamiseen.

Kehitystyössä hyödynsin aiemman osaamiseni lisäksi keskustelufoorumeja ja alan kirjallisuutta. Toteutuksessa käytetty musiikkikirjasto on avoimen lähdekoodin ohjelmisto. Sen käyttö vaati minulta lisäperehtymistä JavaScript-kieleen.

Henkilökohtaisena tavoitteena minulla oli oppia toimintatapoja sovellusten kehittämiseksi. Kehitystyössä opin kirjastojen käyttöä AngularJS-sovelluskehyksessä ja JavaScript-kielen osaamiseni vahvistui. Huomasin, että paras tapa oppia käyttämään AngularJS:ää, on ryhtyä rakentamaan käytännön sovellusta. Käytännön oivallus oli, että sovelluksen kehittämisessä on tärkeää nähdä ja hahmottaa laajemmin kokonaisuus.

Toteutin opinnäytetyön työskentelemällä MusicInfon toimitiloissa. Tein ohjelmointia itsenäisesti. Sain apua projektipäälliköltä työn edetessä vastaan tulleissa haasteissa. Hänen kanssaan mietimme mahdollisia toteutusmahdollisuuksia.

Kehitystyössä haastavaa oli SoundManger2-kirjaston toimintojen muokkaaminen juuri toimeksiantajan palvelun tarpeisiin sopiviksi. Suurimmat haasteet nousivat esiin tiedon hakemisessa tietokannasta ja sen tuomisessa käyttöliittymään. Ongelmaksi muodostui, ettei MusicInfon tietokannassa kappaleiden tiedoissa ollut viitteitä artistiin tai julkaisuun. Asian tilan korjaaminen viivästytti musiikkisoitinsovelluksen kehittämistä.

Lopputuloksena sain kehitettyä toimivan selainpohjaisen sovelluksen. Tämä musiikkisoitinsovellus yhdistetään MusicInfo-palvelun seuraavaan versioon. Sovelluksen kehittämistä tullaan jatkamaan tulevaisuudessa.

Lähteet

Adiseshiah, E.G. 2016. Wireframing tools and atomic design: User experience from the bottom up. Blogikirjoitus Justinmind -verkkosivuilla. Viitattu 7.11.2016. Https://www.justinmind.com/blog/wireframing-tool-and-atomic-design-user-experience-from-the-bottom-up/.

AlexKjes. 2017. HTTP access control (CORS). Artikkeli developer.mozilla.org -verkkosivuilla. Viitattu 10.3.2017. Https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS.

AngularJS. N.d. 2016. Developer Guide angularjs.org -sivustolla. Viitattu 30.3.2016. Https://docs.angularjs.org/guide/introduction.

AngularJS Introduction. N.d. 2016. Tutoriaali w3scools -verkkosivuilla. Viitattu 2.11.2016. Http://www.w3schools.com/angular/angular_intro.asp.

Bloom , C. 2014. Your Frontend Methodology Is All of Them: Atomic Design & Pattern Lab. Blogikirjoitus Phase2 Technology -verkkosivuilla. Viitattu 7.3.2017. Https://www.phase2technology.com/blog/your-frontend-methodology-is-all-of-them-atomic-design-patternlab/.

Booth, R. 2011. Introduction to Web Application Security. Powerpoint -esitys Nasan verkkosivuilla. Viitattu 14.12.2016. Https://www.nasa.gov/offices/ocio/itsummit/presentations_2011.html.

Dayley, B & B. 2015. AngularJS, JavaScript, and jQuery All in One, Sams Teach Yourself E-kirja. Viitattu 30.10.2016. Http://ptgmedia.pearsoncmg.com/images/9780672337420/samplepages/9780672337420.pdf

Data Types (JavaScript). N.d. 2016. Web Development MSDN -kirjastossa Microsoftin verkkosivuilla. Viitattu 30.10.2016. Https://msdn.microsoft.com/en-us/library/7wkd9z69(v=vs.94).aspx.

Doyle, M. 2012. Easy HTML Templates with Mustache. Artikkeli elated.com -verkkosivuilla. Viitattu 2.11.2016. Http://www.elated.com/articles/easy-html-templates-with-mustache/.

Frost, B. 2016. Atomic Design. Chapter 3, Tools of the Trade bradfrost.com -verkkosivuilla. Viitattu 6.11.2016. Http://atomicdesign.bradfrost.com/.

Gandhi, V. 2015. Object Oriented JavaScript. Forzia Tech PowerPoint esitys LinkedIn verkkosivuilla. Viitattu 30.10.2016. Http://www.slideshare.net/Forziatech/object-oriented-programming-in-javascript.

How SoundManager Works. N.d. 2016. Ohjeet schillmania.com -sivustolla. Viitattu 30.3.2016. Http://www.schillmania.com/projects/soundmanager2/doc/getstarted/.

HTML. N.d. 2016. Määritelmä HTML-kielestä. Computer Hope -verkkosivuilla. Viitattu 7.11.2016. http://www.computerhope.com/jargon/h/html.htm.

HTML5. N.d. 2016. HTML-kehittäjän opas, developer.mozilla.org -verkkosivuilla. Viitattu 7.11.2016. Https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5.

HTML5 Audio. N.d. 2016. Tutoriaali W3Schools -verkkosivuilla. Viitattu 29.10.2016. Http://www.w3schools.com/html/html5_audio.asp.

$Http. N.d. 2017. Interceptors. Dokumentaatio docs.angularjs.org -verkkosivuilla. Viitattu 18.3.2017. Https://docs.angularjs.org/api/ng/service/$http#interceptors.

Introduction to AngularJS Javascript Framework. N.d.2016. Esittely Angularista Livecoding.tv -verkkosivuilla. Viitattu 9.11.2016. Https://www.livecoding.tv/learn/angular-js/history/.

Kananen, J. 2010. Opinnäytetyön kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulun julkaisuja -sarja.

Kumar, D. 2015. Best Practices for Building RESTful Web services. White Paper. Dokumentti Infosys -verkkosivuilla. Viitattu 7.11.2016. Https://www.infosys.com/digital/insights/Documents/restful-web-services.pdf.

Long, J.W. 2011. Sass vs. SCSS: which syntax is better? Artikkeli thesassway -verkkosivuilla. Viitattu 6.11.2016. Http://thesassway.com/editorial/sass-vs-scss-which-syntax-is-better.

Loui, R. P. (2008). In praise of scripting: Real programming pragmatism. Computer 41 (7), 22-26.

Luukkainen, M. 2011. Ohjelmistojen mallintaminen. Luentomateriaali. Helsingin yliopisto, Tietojenkäsittelytieteen laitos, matemaattis-luonnontieteellinen tiedekunta. Viitattu 9.3.2017. Https://www.cs.helsinki.fi/u/mluukkai/ohmas10/luentokalvot/luento1.pdf.

Medley, J. 2017. Content Security Policy (CSP). Artikkeli developer.mozilla.org -verkkosivuilla. Viitattu 7.3.2017. Https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

Meijer, E. & Drayton, P. 2004. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. Teoksessa Intended for submission to the Revival of Dynamic Languages. Microsoft Corporation (s. 1-6). Citeseer.

Mustache Manual. N.d. 2014. Dokumentaatio mustache.github.io -verkkosivuilla. Viitattu 2.11.2016. Http://mustache.github.io/mustache.5.html

Mohammad, W. M. & Golrakh M. 2015. Mastering AngularJS for .NET Developers. Introduction, E-kirja. Viitattu 7.3.2017. Https://www.packtpub.com/mapt/book/Web%20Development/9781783553983.

Nourie, D. 2006. Java Technologies for Web Applications. Artikkeli Oraclen verkkosivuilla. Viitattu 9.3.2017.Http://www.oracle.com/technetwork/articles/javase/webapps-1138794.html#webapp.

Odell, D. 2009. Pro JavaScript RIA Techniques: Best Practices, Performance and Presentation. Multimedia Playback. p. Apress.

Osmani, A. 2012. Learning JavaScript Design Patterns. Categories Of Design Pattern. 15.–16. p. O'Reilly.

Pattern Lab. N.d. 2016. Documentation Pattern Lab -verkkosivuilla. Viitattu 6.11.2016. Http://patternlab.io/docs/index.html.

Perälä, J. 2016. IT-tuotteiden ja -palveluiden vaihtaminen: musiikin suoratoistopalvelut. Pro gradu tutkielma, Jyväskylän yliopisto, Tietojenkäsittelytieteiden laitos JYX-julkaisuarkisto -verkkosivuilla. Viitattu 29.10.2016. Https://jyx.jyu.fi/dspace/handle/123456789/50250.

Reini, J. 2012. REST-rajapinnat kiinnostavat sovelluskehittäjiä. Lehtileike, paikkatietoikkuna -verkkosivuilla. Viitattu 7.11.2016. Https://www.paikkatietoikkuna.fi/c/document_library/get_file?uuid=97cc3ed7-bbee-4230-96a8-6a0615f14515&groupId=108478.

Rintamäki, M. 2017. Lyhyesti ohjelmistojen kehityksestä. PDF -kurssimateriaali, Jamk. Viitattu 13.3.2017. Https://docs.google.com/presentation/d/1cJwDftttbdNb6uUBDaUoJa2jqdKAkBskEJJFtl_fDTY/edit#slide=id.p4.

Sayar, R. 2015. Top JavaScript Frameworks, Libraries and Tools and When to Use Them. Artikkeli sitepoint.com -verkkosivuilla. Viitattu 2.11.2016. Https://www.sitepoint.com/top-javascript-frameworks-libraries-tools-use/.

Sass. N.d. 2016. Sass Documentation sass-lang -verkkosivuilla. Viitattu 6.11.2016. Http://sass-lang.com/documentation/.

Segal, N. 2016. An Overview of HTML5. Artikkeli Htmlgoodies -verkkosivuilla. Viitattu. 7.11.2016. Http://www.htmlgoodies.com/html5/an-overview-of-html5.html#fbid=YqSlPKrguJX.

Syromiatnikov, A. 2016. A Journey Through the Land of Model-View-* Design Patterns. Degree project Linnaeus University Department of Computer Science. Viitattu 2.11.2016. Http://www.diva-portal.org/smash/get/diva2:738269/fulltext01.pdf.

Toscano, J. 2015. The Unicorn Workflow: Design to Code with Atomic Design Principles and Sketch Artikkeli Medium -verkkosivuilla.Viitattu 7.11.2016. Https://medium.com/re-write/the-unicorn-workflow-design-to-code-with-atomic-design-principles-and-sketch-8b0fe7d05a37#.3yyo94xmb.

Vaqqas, M. 2014. RESTful Web Services. Tutoriaali Dr.Dobbs -verkkosivuilla. Viitattu. 7.11.2016. Http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069.

What is JavaScript. N.d. 2016. Artikkeli Mozilla Developer Network -sivustolla. Viitattu 30.3.2016. Https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.