

Komponenttipohjaisen käyttöliittymän kehittäminen

Case: Musicinfo Finland Ltd.

Toni Matilainen

Opinnäytetyö

Maaliskuu 2016

Luonnontieteiden ala

Tradenomi(AMK), tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t) Matilainen, Toni	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Huhtikuu 2016
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi Komponenttipohjaisen käyttöliittymän kehittäminen Case: Musicinfo Finland Ltd.		
Tutkinto-ohjelma Tietojenkäsittelyn tutkinto-ohjelma		
Työn ohjaaja(t) Tommi Tuikka		
Toimeksiantaja(t) Musicinfo Finland Ltd.		
Tiivistelmä <p>Opinnäytetyön tavoitteena oli löytää sopivin komponenttipohjainen käyttöliittymäratkaisu Musicinfo Finland Oy:lle. Tutkimus toteutettiin kehittämistutkimuksena, jossa kartoitettiin nykytilanne sekä tutkittiin ja testattiin työkaluvaihtoehtoja. Tutkimuksen perusteelta valittiin tekniikka, jolla voitaisiin toteuttaa Musicinfo-sovelluksen uudistunut käyttöliittymä.</p> <p>Tutkimus toteutettiin testaamalla nykyistä ja kahta uutta tekniikkaa. Tekniikoilla luotiin esimerkkisivu, jonka avulla voitiin nähdä, miten tekniikka toteuttaa kyseisen ratkaisun. Vertailukriteereinä olivat REST-kutsut, design, applikaation reititys, hakukoneystävällisyys, tekniikan oppimiskynnys, mobiilituki, komponenttiratkaisun laatu ja tekniikan suorituskyky.</p> <p>Tutkimuksen tuloksena valittiin React, joka erottui muista erityisesti selain- ja mobiililutensa sekä Pattern Lab yhteensopivuuden ansiosta.</p>		
Avainsanat (asiasanat) Ohjelmointi, ohjelmistokehitys, web-kehitys, Polymer, React, Google, Musicinfo		

Author(s) Matilainen, Toni	Type of publication Bachelor's thesis	Date April 2016 Language of publication: Finnish
	Number of pages 45	Permission for web publication: X
Title of publication Web-component based user interface development Case Musicinfo Finland Oy		
Degree programme Business Information Systems		
Supervisor(s) Tuikka, Tommi		
Assigned by Musicinfo Finland Oy		
Abstract Goal of the thesis was discover best component based user interface solution for Musicinfo Finland Oy. Research included mapping enterprise's current state, and then researching new solutions and testing them. Based on conclusions, technology was selected which can be used to build new user interface for Musicinfo's application. Research was conducted by testing current technology, and two new alternative ones. Example page was developed with these technologies, in order to see how the pages can be built with these technologies. Reference criterions were REST –methods, design, application routing, search engine optimization, technology's learning curve, mobile compatibility, performance, and quality of web component. As a result of research, React was best option. It stood out from others especially in browser- and mobile compatibility and in Pattern Lab compatibility.		
Keywords/tags (subjects) Polymer, React, Google, modular, web development, web components, Musicinfo		

Sisältö

Termit	7
1 Johdanto	9
2 Tutkimusasetelma	10
2.1 Toimeksiantaja ja tutkimuksen tavoite	10
2.2 Tutkimusmenetelmä ja tutkimuskysymykset.....	11
2.2.1 Tutkimuskysymykset.....	11
2.2.2 Rajaus.....	11
2.3 Tutkimuksen toteutus	11
3 Web-komponentit	12
3.1 Web-komponenttien hyödyt.....	12
3.2 Web-komponenttien (nykyiset) haitat.....	13
3.3 Web-komponenttien arkkitehtuuri.....	13
3.3.1 Sivupohjat	14
3.3.2 Varjo-DOM.....	14
3.3.3 Kustomoidut elementit.....	15
3.3.4 Tuonti-HTML.....	15
3.3.5 Selaintuki	15
3.4 Tekniikoiden yleiskatsaus	16
3.4.1 Google Polymer	16
3.4.2 React/ React JS.....	16
3.4.3 AngularJS	19
3.4.4 REST	20
3.4.5 Pattern Lab	23
3.5 Komponenttitekniikoiden mobiilituki	24
3.5.1 Polymer.....	24
3.5.2 ReactJS.....	25

3.5.3	AngularJS	25
4	Tekniikan valinta.....	27
4.1	Polymer.....	27
4.1.1	Rest-kutsut.....	27
4.1.2	Design	28
4.1.3	Reititys	29
4.1.4	Hakukoneystävällisyys	29
4.1.5	Renderöinti/ koodin haastavuus	30
4.1.6	Mobiilituki.....	30
4.1.7	Web-komponentit	31
4.2	React.....	31
4.2.1	Rest-kutsut.....	31
4.2.2	Design	32
4.2.3	Reititys	32
4.2.4	Hakukoneystävällisyys	32
4.2.5	Mobiilituki.....	33
4.2.6	Renderöinti/ koodin haastavuus	33
4.2.7	Web-komponentit	34
4.3	AngularJS	34
4.3.1	Rest-kutsut.....	34
4.3.2	Design	35
4.3.3	Reititys	35
4.3.4	Hakukoneystävällisyys	35
4.3.5	Renderöinti/ koodin haastavuus	36
4.3.6	Mobiilituki.....	36
4.3.7	Web-komponentit	36
4.4	Nopeus.....	37

5 Johtopäätökset.....	38
Lähteet.....	40
Liitteet	43
Liite 1. Kuvankaappaukset rakennetuista sivuesimerkeistä	43

Kuviot

Kuvio 1. Flux-kaavio (Flux, n.d).....	18
Kuvio 2. Pattern Labin metodologia (About Atomic Design n.d)	23
Kuvio 3 Pattern Labin tuottama koodi diskografiaelementistä.....	24
Kuvio 4. Polymerin REST-pyyntö (Using Polymer's iron-ajax to load json files n.d.)	27
Kuvio 5. Polymerin designer-työkalu (Designer, n.d).....	28
Kuvio 6. Designerin tuottama koodi (Designer, n.d)	29
Kuvio 7. Polymerin datan renderöinti (Data binding helper elements, n.d)	30
Kuvio 8. Reactin REST-pyyntökokonaisuus (React n.d)	31
Kuvio 9. Reactin datan renderöinti (Tutorial. n.d)	33
Kuvio 10 Reactin datan renderöinti etusivulle (Tutorial. n.d)	33
Kuvio 11 Angularin REST-kutsu (Requesting JSON data with AJAX. N.d).....	34
Kuvio 12 Angular renderöinti/koodin haastavuus	36
Kuvio 13. Tekniikoiden latausajat	37
Kuvio. 14 Johtopäätökset.....	38

Taulukot

Taulukko 1 Web-komponenttien osat	13
Taulukko 2. Selaintuki (WebComponents n.d.).....	15
Taulukko 3. Yleisimmät REST-pyynnöt esimerkkitalanteessa	20

Termit

AJAX = Tekniikka, jolla voidaan siirtää tietoa palvelimen ja asiakkaan välillä ilman koko web-sivun lataamista.

CSS3 = Cascading Style Sheets. Www-sivujen tyyliohjetiedostot. Niillä määritellään, miten sivujen värit, lokaatiot ja osien tyyli toimivat. CSS3 on CSS:n uusin versio.

Datamalli = Malli, joka sisältää datan organisoidusti ja kertoo, kuinka elementit ovat toisiinsa yhteydessä.

DOM = Document Object Model. Dokumenttioliomalli on tapa kuvata sivujen rakennetta rakennepuuna. Sen tarkoitus on määrittää, miten sivun eri oliot määräytyvät sivulla ja miten niihin voi viitata.

HTML5 = Hyper Text Markup Language. Yleisin www-sivuilla käytetty merkintäkieli. Tällä koodilla kirjoitetaan sivut. HTML5 on HTML-versioista uusin.

Iframe = HTML-elementti, jolla luodaan sivu toisen www-sivun sisälle.

JSON = JavaScript Object Notation. Tiedostomuoto, jolla voidaan välittää verkkosivuilla tietoa JavaScript-objekteina.

Latenssi = Se aika, mikä tietoverkossa siirrettävältä paketilta kuluu matkaan lähettäjältä vastaanottajalle ja takaisin.

Olio = Yksikkö, jonka sisällä on tietoja ja toiminnollisuuksia, jotka kuuluvat loogisesti yhteen. WWW-sivut toimivat siten, että erilaiset oliot kommunikoivat keskenään ja näyttävät käyttäjälle dataa.

PHP = Hypertext PreProcessor. Palvelinympäristöissä käytetty kieli, jolla voidaan luoda dynaamisesti latautuvia sivuja.

Polyfill = Koodin pätkä, joka tarjoaa käyttöön teknologian, jota selain ei alun perin tue.

Reaktiivinen = Kun ohjelmiston yksi osa on riippuvainen toisesta ja kun toinen osa muuttuu, niin muuttuu toinenkin.

Scope = Angularissa käytettävä datamalli, joka yhdistää kontrollerin ja käyttäjälle näytetyn näkymän.

SEO = Hakukoneoptimointi. Tekniikka, jolla parannetaan sivuston näkyvyyttä hakukoneiden tuloksissa.

URL = Uniform Resource Locator. Merkkijono ja standardi, jolla internetissä olevia resursseja voidaan paikantaa ja antaa ohjeita. Perinteinen WWW-osoite on URL.

XML = Kieli, jolla datan joukkoon voidaan lisätä sen merkitystä kuvaavaa tietoa.

1 Johdanto

Viime vuosina web-sovelluskehityksen trendiksi ovat nousseet komponenttipohjaiset käyttöliittymät. Komponenttipohjainen tekniikka tarkoittaa sivun rakentamista erinäisistä komponenteista, joita voidaan käyttää uudelleen ja järjestää halutulla tavalla. Ennen tämä ei ollut kovin helposti mahdollista, koska sivun osat olivat vahvasti kytköksissä toisiinsa. Web-komponenttien arkkitehtuuria ja tekniikoita itsessään käsitellään tarkemmin luvussa 3.

Vuonna 2013 Facebook julkaisi React-komponenttikirjastonsa, jolla he kehittävät Facebookia ja Instagram-palveluitaan. Se herätti paljon puhetta monimuotoisuudellaan, helppoudellaan ja tekniikallaan, jossa koko sivu toteutetaan Javascript-tekniikkaa käyttäen. Samana vuonna Google kehitti Polymerin, kirjaston, joka käyttää Googlen luomaa Material Design -kieltä. Material Design on Googlen luoma ratkaisu, jolla he haluavat pitää kaikki heidän luomansa applikaatiot saman tyylin ja designin alla. Polymerillä voidaan käyttää tätä design-ratkaisua kehitettäessä applikaatioita.

Opinnäytetyön toimeksiantajana on MusicInfo Finland Ltd, joka halusi uudistaa palveluaan ja tehdä sen rakenteesta paremman. Nykyään käytössä on AngularJS-ohjelmistokehitys, mutta tarkoituksena olisi siirtyä komponenttipohjaiseen ratkaisuun.

2 Tutkimusasetelma

2.1 Toimeksiantaja ja tutkimuksen tavoite

Toimeksiantajana toimii vuonna 2012 perustettu startup-yritys Music.Info Finland Oy (myöhemmin Musicinfo tai toimeksiantaja). Yrityksen tuotteena toimii internetsovellus, josta käyttäjä voi hakea artistista, albumista tai kappaleesta tietoa mahdollisimman kattavasti ja helposti. Sovellus pitää nyt jo sisällään miljoonia tietueita. Musicinfo on voittanut vuonna 2013 San Francisco MusicTech Summitissa parhaan startup-palkinnon ja myös samana vuonna venäläinen Russian Startup Rating antoi heille BBB-luokituksen, mikä on heidän neljänneksi ylin luokitus. Suomalaisittain menestystä Musicinfo keräsi Keski-suomen kauppakamarin järjestämässä KasvuOpen-kisassa, jossa se voitti vuonna 2013 parhaan startup-palkinnon. Musicinfo sijaitsee Jyväskylässä ja se työllistää kahdeksan henkilöä.

Tavoitteena selvittää paras mahdollinen toteutustekniikka MusicInfon uutta käyttöliittymää varten. Tutkimuksen tehtävänä ovat menetelmien tutkinta ja toteutus esimerkkisivun avulla.

Aihetta tarkastellaan sekä yrityksen tarpeiden että sovelluskehityksen kannalta. Tähän saakka yrityksen sivujen toteutus on ollut työlästä ja designin suhteen ei ole ollut minkäänlaista prosessia. Tulevien muutosten myötä saataisiin paljon järjestelmällisempi prosessi ja käytettävyys sivuille. Tekniikan myötä sivujen ylläpito olisi dynamisempää ja käyttäjäystävällisempää.

Aikaisempi sovellus on olemassa, mutta uuden tekniikan myötä pitää mahdollisesti aloittaa puhtaalta pöydältä. Uuden tekniikan valinnassa otetaan huomioon yrityksen design ja kriteerit valinnan kannalta.

Konkreettisena tavoitteena ja tuloksena on valita tekniikka, jonka avulla Musicinfo voi lähteä toteuttamaan käyttöliittymäänsä.

2.2 Tutkimusmenetelmä ja tutkimuskysymykset

Opinnäytetyöni menetelmänä on kehittämistyö. Tavoitteena on kartoittaa Musicinfon nykytilanne, tutkia miten sitä voisi kehittää komponenttitekniikoilla ja mikä olisi parhain mahdollinen tekniikka tilanteeseen.

Kehittämistyössä kehitetään jotain asiaa tai toimintaa. Kehittämistyössä mennään kohti parempaa, ja siinä on kyse tietoisesta toiminnasta. Se edellyttää nykytilan kartoituksen, vaihtoehtojen etsinnän ja arvottamisen, tavoitteidenmäärittämisen ja keinojen valinnan tavoitteiden pääsemiseksi (Kananen 2010, 159)

2.2.1 Tutkimuskysymykset

Tutkimuskysymys

- Mikä toteutustekniikoista soveltuu parhaiten Musicinfoon soveltamiseksi?

Alakysymykset

- Miten tekniikat käyttävät REST-rajapintaa?
- Miten ylläpito muuttuu tekniikan myötä?
- Mikä tekniikka valitaan?
- Kuinka komponenttipohjaisia tekniikat ovat tarkemmin katsottuna?

2.2.2 Rajaus

Opinnäytetyön tutkimus rajataan kolmeen suurimpaan tekniikkaan dokumentoinnin ja laajuuden vuoksi. Muita tekniikoita ei tulla ottamaan työssä huomioon. Näitä ovat esimerkiksi Mozillan X-tag –tekniikka.

2.3 Tutkimuksen toteutus

Tutkimuksen tavoitteena on kehittää Musicinfon palvelua löytämällä sille komponenttipohjainen käyttöliittymä, joka tekisi heidän sovelluksestaan modernin ja mahdollisimman käytettävän.

3 Web-komponentit

Web-komponentit ovat W3C:n kehitteillä oleva standardi, jonka tarkoituksena on antaa malli itsenäisille web-komponenteille. Web-komponenteilla voidaan rakentaa yksittäinen toimiva kokonaisuus sivulle, vaikka esimerkiksi navigointipaneeli, ja sitä voidaan sitten uudelleen käyttää jokaisella tarvittavalla sivulla. Web-komponentit eivät ole vielä täysin virallinen keino, mutta virallistamisprosessia vie eteenpäin W3C:n standardisointi. Standardisointi tuo web-komponentit enemmän tuetuiksi ja niitä voidaan käyttää vapaammin. Standardisointi ja sen tuoma dokumentaatio säästää aikaa ja vaijaa sekä lisää koodin luotettavuutta. (Web Components. N.d.)

3.1 Web-komponenttien hyödyt

Web-komponentit ovat mullistamassa web-kehitystä ja tarjoavat paljon hyötyjä (Learning Web Component Development, 2015):

- *Uudelleenkäytettävyys.* Niitä voidaan siirtää eri sovellusten välillä, jonka myötä kanssakäynti rajapintojen kanssa on saumatonta.
- *Ylläpidettävyys.* Web-komponentit upotetaan sivupohjaan erillisinä tägeinä, erillisinä kokonaisuuksina jonka myötä virheiden etsintä on helpompaa.
- *Alustariippumattomuus.* Web-komponentit luodaan HTML, CSS ja JavaScript -tekniikoilla, joten ne voivat toimia kaikilla selaimilla.
- *Varjo-DOM.* Tekniikka tarjoaa kapsulointimekaniikan tyyliille, skripteille ja HTML-koodille. Tämä mahdollistaa yksityisen näkyvyysalueen, minkä avulla koodi ja sisältö voidaan pitää erossa toisistaan.

3.2 Web-komponenttien (nykyiset) haitat

Web-komponentit eivät ole vielä täysin standardisoituja vaihtoehtoja, vaan niissä on vielä ongelmia (Learning Web Component Development, 2015):

- *Selaintuki.* W3C:n web-komponenttimäärittelykset ovat vielä keskeneräiset, joten tekniikkaa ei ole täysin voitu sisällyttää selaimiin.
- *Jaetut resurssit.* Web-komponenteilla on omat yksityiset, omaan näkyvyysalueeseen sidotut resurssit, joten samoja resursseja voi mennä sekaisin komponenttien välillä.
- *Suorituskyky.* Kun web-komponenttien määrä kasvaa, niiden sijoittaminen DOMiin tulee viemään enemmän aikaa.
- *Polyfillin muistinkäyttö.* Kun polyfillit varastoivat sisään dataa, ja käyttävät sitä, se aiheuttaa hitautta raskautensa vuoksi.
- *SEO-ongelma.* Kun sivupohjien sisällä olevat elementit ovat staattisia, se luo ongelmia nettisivujen hakuoptimointiin.

3.3 Web-komponenttien arkkitehtuuri

W3C määrittelee web-komponentit siten, web-komponentit rakentuvat neljästä eri osasta.

Taulukko 1 Web-komponenttien osat

Sivupohjat	Varjo-DOM	Kustomoidut elementit	Tuonti-HTML tiedostot	
------------	-----------	-----------------------	-----------------------	--

3.3.1 Sivupohjat

Web-komponentin `<template>` -elementti on komponenttien varsinainen komponenttiosa. Siihen sisällytetään kaikki se data ja tyyli, jota halutaan uudelleenkäyttävän. Polymerissä käytetään myös `template`-elementtiä luomaan sisäkkäisiä komponentteja. (Learning Web Component Development 2015.)

3.3.2 Varjo-DOM

Ennen web-komponentti tekniikoiden määrittelyä web-kehitys sisälsi monia ongelmia. Suurimmat niistä olivat (Learning Web Component Development 2015):

- *Tyylien ylikirjoitus.* Dokumentin oma tyylitiedosto voi vaihtaa web-komponenttien tyyliä
- *Skriptien ylikirjoitus.* Dokumentin JavaScript-tiedosto voi muuttaa web-komponenttien osia.
- *Limittäiset ID:t.* Dokumentti voi sisältää monta duplikaatti-ID:tä, jotka aiheuttavat ongelmia.

Näiden lisäksi myös näkyvyysalueet tuottivat ongelmia. Varjo-DOM auttaa näissä ongelmissa kapselointimekaniikallaan. Varjo-DOM:n avulla voidaan koteloida HTML, CSS ja JavaScript kokonaiseksi web-komponentiksi. Varjo-DOM tuo myös kyvyn sisällyttää toisia DOM-puun osia dokumentin sisälle. Varjo-DOM myös nopeuttaa huomattavasti sivun latausta, kun kaikkia turhia DOMin sisältäviä elementtejä ei tarvitse ladata käyttäjälle asti. Kaikki selaimet eivät tätä silti tue, ja sen lataaminen polyfillien kautta on raskasta. Tämän takia Google Polymer on keksinyt välimuodon. Shady DOMin eli ratkaisun, jolla voidaan saada nyt jo varjo-DOMin ratkaisut kaikilla näkyville. Ratkaisu parantaa varsinkin Safarin mobiiliselaimen nopeutta huomattavasti. (Learning Web Component Development 2015.)

3.3.3 Kustomoidut elementit

Kustomoidut elementit ovat elementtejä, joilla voi olla muista riippumaton tyyli ja rakenne. Näitä elementtejä voidaan siten käyttää kaikilla sivuilla ja muotoilla miten vaan. Kustomoidulla elementillä on paljon etuja. Ohjelmoijan tarvitsee koodata vähemmän, tagi-kirjasto on paljon semanttisempi ja luettavampi ja DIV-tagit vähenevät suuresti. (Learning Web Component Development 2015.)

3.3.4 Tuonti-HTML

Tuonti-HTML mahdollistaa toisen HTML-dokumentin tuonnin toiselle sivulle. Se antaa vaihtoehdon nykyiselle iframe-tekniikalle, joka on kankea ja epävaka ratkaisu ulko-puoliselle sisällölle. (Learning Web Component Development 2015.)

3.3.5 Selaintuki

Web-komponenttitekniikat ovat sen verran uusi ilmiö, että kaikki selaimet eivät ole ehtineet mukaan tukemaan sitä

Taulukko 2. Selaintuki (WebComponents n.d.)

	Chrome	Opera	Firefox	Safari	IE/Edge
Sivupohjat	■	■	■	■	■
Kustomoidut elementit	■	■	■	■	■
Varjo-DOM	■	■	■	■	■
Tuonti-HTML	■	■	■	■	■

■ = Tukee ■ = Ei tue /harkinnassa

■ = Kehitteillä tai olemassa testiversioissa

3.4 Tekniikoiden yleiskatsaus

Opinnäytetyötä kirjoitettaessa web-komponenttitekniikoita on tullut useampia, mutta niistä valittiin vain merkittävimmät ja tuetuimmat dokumentaation sekä käyttöönoton vuoksi.

3.4.1 Google Polymer

Polymer on kirjasto, jota voidaan käyttää ohjelmiston kehitystyössä, joka hyödyntää web-komponentteja. Polymerin tavoitteena on seurata heidän kehittämänsä Material Design -designmallia ja rakentaa selkeitä, helposti komponenteista rakennettavia sivuja. Material Design on monelle tuttu Androidista. Polymer käyttää web-komponenttipohjaista tyyliä, jossa voidaan rakentaa omia HTML-osien tapaisia elementtejä. Tämä menetelmä edistää uudelleenkäytettävyyttä ja yksinkertaisuutta. (Material design n.d.) Polyfill-tekniikka auttaa Polymeria ja sen ominaisuuksia toimimaan uusimmissa, sekä vanhoissa selaimissa.

Polymerillä on rakennettu muun muassa Google Music -musiikkipalvelu, Googlen Zeitgeist-palvelu; palvelu, jolla voidaan paikallistaa käyttäjien tekemät Google-haut ja lähdekoodien pilvipalveluihin perehtynyt GitHub. (Who's using Polymer?, 2016)

3.4.2 React/ React JS

Reactin tarkoitus on olla MVC-mallin V eli View, näkymä. Tämän takia sitä onkin helppo myös kokeilla nopeasti, koska ei tarvitse huolehtia muusta. Myös itse kirjasto on tämän yksinkertaisuuden takia pieni, todella nopea ja dynaaminen. Kehittäjät loivat Reactin ratkaistakseen ongelman isojen sivujen dynaamisen datan kanssa. Dynaaminen data on käytännössä sitä, että voidaan olla jatkuvassa kanssakäymisessä sivun kanssa ja että sivu lataa itsensä aina uudelleen muutosten myötä. React haluaa erottaa web-komponenttitekniikoista tekemällä puhtaasti kokonaisia komponentteja eikä vain sivupohjia, joissa ajetaan koodia niiden sisällä tai niiden kautta. (Why React? n.d.)

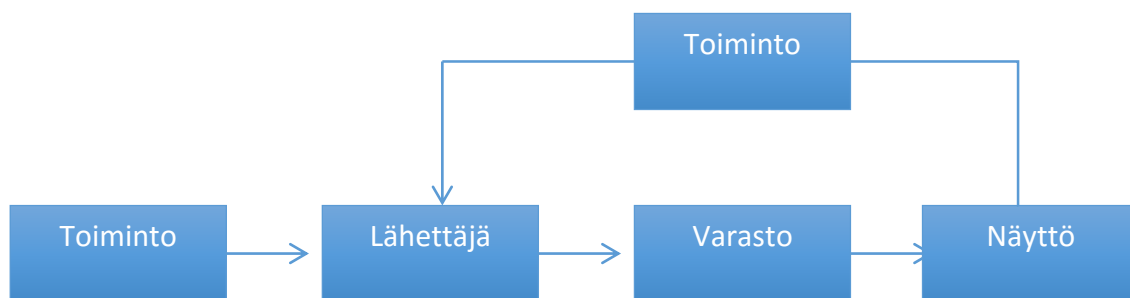
Reactin jatkuvan datan malli toimii siten, että applikaatiot luovat kevyen renderöinti-metodin, joka toimii prototyyppinä tehdystä applikaatiosta. Siitä mallista luodaan sivuille näytettävät elementit. Muutoksien tapahtuessa sama sykli kutsutaan uudelleen. (Working With the Browser n.d.)

Nopeudessa React loistaa Virtual DOMin avulla. Se ei eroa paljoakaan normaalista tutusta DOM-rakenteesta, vaan se pyrkii olemaan yksinkertaistettu, suorasukaisempi versio perinteisestä. Se ei ole heidän keksimänsä tekniikka, mutta he käyttävät sitä suoraan ratkaisuisaan ilman lisäratkaisuja tai lisäosia. React luo virtuaalisen DOM-puun, joka ohittaa perinteisen raskaan DOM-puun ja käyttää sitä sivun rakenteena. Etuna on myös se, että kun tehdään muutoksia, päivitetään ainoastaan muutokset eikä koko DOM-puuta. Kyseiset ratkaisut ovat nopeita, virtaviivaistettuja, mutta eivät täysin aukottomia. React pitää sisällään kopiota DOM-puusta, mikä syö muistia enemmän kuin muut ratkaisut. (Working With the Browser n.d.)

Reactilla on kehitetty muun muassa nuorison keskuudessa suosittu valokuvapalvelu Instagram, videostriimauspalvelu Netflix, Klarna, Suomessakin paljon käytetty laskutuspalvelu ja kommenttijärjestelmiä rakentava Disqus. Kyseiset palvelut on myös rakennettu Flux-arkkitehtuurin avulla.

Pelkällä Reactilla voidaan rakentaa yksinkertaisia applikaatioita, mutta jos applikaatiossa on dynaamista dataa eli alati vaihtuvaa dataa, niin silloin Flux tuo paljon etua nopeuteen ja arkkitehtuuriin. Fluxin avulla Reactista Jos näytetään vain staattisia sivuja, joissa tieto ei muutu koko ajan, niin silloin Flux ei tuo paljoakaan etua. (Flux n.d.)

Flux-arkkitehtuuri koostuu kolmesta tärkeästä osasta: lähettäjä, varasto ja näyttö. Näitä kolmea osaa yhdistää toiminto-osa (ks. kuvio 1) (Flux n.d.).



Kuvio 1. Flux-kaavio (Flux, n.d)

- **Näyttö:** Osa, jonka käyttäjä näkee. Tässä osassa renderöity data näkyy käyttäjälle.
- **Varasto:** Pitää sisällään kaiken applikaation datan ja sovelluslogiikan. Muistuttaa paljon MVC:n mallia (model). Varasto on dataobjektien kokoelma, kun MVC-arkkitehtuurin malli on yhden objektin varasto.
- **Lähettäjä:** arkkitehtuurin keskus, jonka läpi menevät varaston kautta tulevat kutsut.

Flux-arkkitehtuuri toimii käytännössä siten, että ensimmäiseksi käyttäjä luo toiminnan tekemällä jotain käyttöliittymän sisällä. Nämä toiminnot menevät lähettäjän läpi prosessointia varten. Toiminto herättää varastossa olevan callbackin, joka muuttaa datan sisältöä. (Learning Web Component Development 2015).

Sen jälkeen varasto päästää muunnellun datan takaisin, jolloin näyttöosa tarkkailee datan muuttumista ja tapahtumakäsittelijän avulla pääsee käsiksi dataan. Sen jälkeen näyttö asettaa itselle setState-tilan, joka saa käyttöliittymän uudestaan lataamaan itsensä. (Learning Web Component Development 2015.) Kaikki datan liike Fluxin sisässä on yksisuuntaista, ja koska Fluxin arkkitehtuuri ei salli kaksisuuntaista interaktiivista datan käsittelyä, niin se aiheuttaa monia peräkkäisiä päivityksiä.

Fluxin avainperiaatteina toimivat neljä kohtaa (Learning Web Component Development 2015):

- *Synkronia*. Kaikki callback-toiminnot tapahtuvat rinnakkaisesti, mutta toiminnot itsessään tapahtuvat asynkronisesti lähteestä käsin.
- *Kontrollin käänteisyys*. Varasto voi toimia itsenäisesti ja suorittaa tekoja itsenäisesti ilman erinäistä moduulia/kontrolleria. Tällöin kontrollin kulku ei ole perinteinen.
- *Semanttiset toiminnot*. Kun toiminto suoritetaan, se sisältää semanttisia attribuutteja, jotka auttavat varasto-objektia tietämään oikean suoritustavan.
- *Ei rinnakkaisia toimintoja*. Flux ei salli toimintoja, jotka aiheuttavat toisia toimintoja. Näistä voisi aiheutua rinnakkaisia päivityksiä, jotka hidastaisivat järjestelmää.

3.4.3 AngularJS

AngularJS on myös kehitetty Googlen toimesta, ja se sai alkunsa vuonna 2009, kun Googlen insinööri oli tyytymätön siihen, että sen aikaiset tekniikat eivät olleet helposti uudelleenkäytettäviä, koska ne sisälsivät liikaa riippuvaisuuksia ja pakollisia vaikeasti ylläpidettäviä koodipätkiä. Siitä motivoituneena hän kehitti ratkaisun, joka sitten vuosien saatossa kehittyi viralliseksi AngularJS-kirjastoksi. (Web Component Architecture & Development with AngularJS, 2015)

AngularJS ei ole varsinaisesti web-komponenttitekniikka, mutta niitä muistuttavia komponentteja voidaan rakentaa Angular-direktiivien avulla. Direktiivit ovat toteutus-tapa, jolla voidaan luoda ja kapseloida itsenäisiä liitännäisiä, jotka ovat riippumattomia ulkoisista riippuvuussuhteista. Angular loi myös alustan dynaamisille sivuille. Ennen Angularia JQueryllä voitiin tehdä dynaamisia ja interaktiivisia sivuja, jotka kommunikoivat DOM:n kanssa. Angularin kaksisuuntaisen datavirran avulla voidaan rakentaa sivuja, jotka latautuvat käyttäjän muutosten alaisena. Syyt AngularJS:n erittäin isoon suosioon ovat monet. Angular käyttää JSONia datamallissaan, mikä mahdollistaa mutkattoman ominaisuuksien vaihdon. Dokumentaatio on laajaa ja tuettua Googlen ansiosta. (Web Component Architecture & Development with AngularJS, 2015).

Angular eroaa muista verrattavissa olevista tekniikoista eniten siinä, että se on ainoa kokonainen sovelluskehys. Polymer on kirjasto, jolla voidaan toteuttaa web-komponenttiratkaisuja joka hyödyntää selaimen natiiveja toimintoja, ja React toimii MVC-mallin V-osana eli näkymänä (view). (Tivi, 2016)

3.4.4 REST

REST on arkkitehtuurityyli, joka pyrkii siihen, että eri osapuolet voivat kehittyä ja muuttua toisistaan riippumatta. RESTin avainkäsite on resurssi ja sen esitysmuodot. RESTin esitysmuotoina toimivat URLit, joilla dataa hallitaan. State tarkoittaa, että palvelun resursseilla ja tiedoilla on tiloja, joihin vaikutetaan erilaisilla kutsuilla. Kutsuja ovat haku, poisto, lisäys ja päivitys. Transfer tarkoittaa sitä, että asiakkaan tilaa myös voidaan muuttaa REST-kutsuja käyttämällä. (REpresentational State Transfer (REST) n.d.)

RESTissä käytetään yleisimmin JSON-kieltä datamuotona. JSON on kuin yleinen XML, mutta kevyempi ja selkeämpi. (Web Services: JSON vs. XML. n.d)

Taulukko 3. Yleisimmät REST-pyynnöt esimerkkitalanteessa

	GET	PUT	POST	DELETE
/esimerkkejä/	Hakee listauksen esimerkeistä.	-	Lisää uuden resurssin listaan.	-
/esimerkkejä/esim1	Hakee tiedon esimerkki1-resurssista.	Korvaa esimerkki1-resurssin toisella resurssilla.	Lisää uuden resurssin tietyllä tunnisteella.	Poistaa esim1-resurssin.

REST-arkkitehtuuria kehitettäessä oli kuusi tavoitetta, jotka haluttiin täyttää (REST Architectural Goals n.d.)

Suorituskyky

Tiedonsiirto järjestelmästä toiseen hajautetussa ympäristössä on tärkein. Suorituskykyä verkossa pitää optimoida niin, että latenssit saataisiin mahdollisimman pieniksi. Yksi tärkeimpiä tekijöitä verkon suorituskyvyn kannalta ovat optimoinnit, joilla saadaan ap- plikaatio mahdollisimman vähän käyttämään itse verkkoa. Sitä voidaan vähentää väli- muistien käytöllä ja siirtämällä data lähemmäksi lähdettä.

Skaalautuvuus

Järjestelmän komponenttien asetukset ovat aktiivisia ja joustavia. Kun järjestelmä ja sen asetukset kasvavat, järjestelmä voi silti olla tehokas pienillä kanssakäymismäärillä.

Yksinkertaisuus

Komponenteista tehtäisiin mahdollisimman yksinkertaisia ja selkeitä, jotta niitä olisi helppo sisällyttää muualle ongelmitta.

Muunneltavuus

Järjestelmän pitää alati olla valmis muutoksiin ilman, että koko järjestelmää pitäisi käynnistää uudelleen. Vaikka järjestelmä tehtäisiin täysin käyttäjän vaatimusten mu- kaan, vaatimukset ovat silti alati muuttuvia, joten järjestelmän pitää olla valmis muu- toksiin, eikä niiden pidä olla vaikeasti toteutettavissa.

Selkeys

Kahden järjestelmän välisen kommunikaation ja tietoliikenteen pitäisi olla niin sel- keätä, että kolmas osapuoli ymmärtäisi sitä. Tämä mahdollistaisi palomuurien ja jaet- tujen välimuistien toteutuksen.

Siirrettävyys

Komponenttien pitää olla mahdollisimman helppoja siirtää ympäristöstä toiseen ilman suurempia muokkauksia.

Luotettavuus

Järjestelmän pitää olla sietokykyinen, jotta yksittäinen virhe ei kaada koko järjestelmää.

RESTin rajoittavat piirteet (What Is REST? n.d.):

Yhtenäinen käyttöliittymä

Yhtenäinen käyttöliittymä on kokoelma määrittämiä, joilla saadaan käyttöliittymä yhtenäiseksi. Se sisältää syntaksin resurssien tunnistamiselle (URL) sekä tavat esittää resurssit (MIME/mediatyypit) ja kutsut.

Tilattomuus

Tilattomuus voi kuulostaa ristiriitaiselta siksi, että REST:n yksi osa on juuri tilallisuus (state). Tämä tarkoittaa sitä, että itse protokolla on tilaton. Mitään tärkeää informaatiota ei ole piilotettu, vaan kaikki on näkyvillä kutsuissa.

Välimuistikelpoinen

Mahdollisuus välimuistien käyttöön parantaa suorituskykyä ja vähentämään turhien toistuvien pyyntöjen hakua, kun ne voidaan hakea läheltä välimuistista.

Asiakaspalvelin

Asiakas ja palvelin pidetään erossa toisistaan. Asiakkaan ei tarvitse huolehtia palvelimen tietovarastoista, eikä palvelimen tarvitse huolehtia käyttöliittymistä. Ne voidaan jopa korvata ja kehittää omana kokonaisuutenaan, kunhan kanssakäymiskäytäntöjä ei muuteta.

Kerrostettu järjestelmä

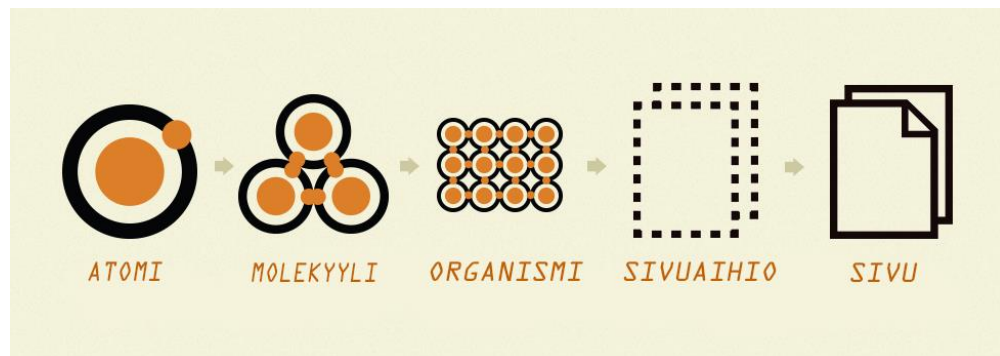
Järjestelmän taakan tasoittamiseksi ja turvallisuuden parantamiseksi on hyvä olla välissä eri ratkaisuja. Niitä ovat esimerkiksi palomuuuri, välityspalvelin ja välimuisti. Nämä eivät kuitenkaan saa muuttaa käyttöliittymää asiakas-palvelinvälillä.

HATEOAS

HATEOAS eli Hypermedia as the Engine of Application State, joka selvimmin erottaa RESTin muista arkkitehtuureista. Tekniikan ideana on kykenevyys ajaa hypermedioita

eli suoria linkkejä REST-vastausten sisällä. Tämä parantaa arkkitehtuurin dynamiikkaa ja sujuvuutta.

3.4.5 Pattern Lab



Kuvio 2. Pattern Labin metodologia (About Atomic Design n.d)

Pattern Lab perustuu menetelmäoppiin, jonka avulla rakennetaan design-järjestelmiä. Tätä menetelmäoppia kutsutaan atomiseksi designiksi. Atominen design on muutakin kuin tapa, miten järjestellä elementtejä ja ymmärtää niitä. Se on kokonainen PHP-sovelluskehys nimeltä Pattern Lab, jolla voidaan luoda oikeanlainen merkistö täysin interaktiiviselle tyyliohjeistukselle ja prototyyppijärjestelmille. Pattern Lab on myös responsiivinen sovelluskehys, joten toimivuus on sama työpöytä ja mobiiliympäristöissä. (About Atomic Design n.d.)

Pattern Lab koostuu viidestä eri osasta (Ks. kuvio 2) (About Atomic Design n.d.):

- Atomeja ovat sivun HTML-tagit eli input-tagit, button-tagit ja linkkitagit. Kuten luonnossakin, atomit itsessään eivät ole kovin hyödyllisiä yksinään.
- Molekyylejä tulee, kun yksinään hyödyttömiä atomeja yhdistetään toisiinsa, ja niistä muodostuu jotain suurempaa, kuten esimerkiksi sivun form/haku-lomakkeet.

- Organismit ovat itsenäisiä tai koostuvat monesta molekyylisestä rakentaen yhden ison loogisen kokonaisuuden, esimerkiksi sivuston valokuvaosion. Organismit ovat myös uudelleenkäytettäviä komponentteja.
- Sivuaihiot ovat koko sivun levyisiä kokonaisuuksia, joissa monta organismia on sidottu yhteen ja voidaan nähdä sivun rakentuva design. Sivuaihiosta voidaan nähdä, minne organismit asettuvat ja miltä ne näyttäisivät sivun rakenteessa. Sivuaihio koostuu organismeista tai molekyyleistä, joiden paikan sivuaihio määrittää. Sivuaihiot ovat myös uudelleenkäytettäviä osia.
- Sivut ovat kuten sivuaihioita, mutta paljon yksityiskohtaisempia. Kun halutaan tehdä todella spesifinen sivu, mikä ei toistu Pattern Labissa, valitaan sivu eikä sivuaihio.

```

<div class="Discography">
  <div class="Discography-header">
    <h2>Discography</h2>
    <div class="Discography-filters">{{> molecules-DiscographyFilters }}</div>
  </div>
  <slick infinite=true slides-to-show=3 slides-to-scroll=1 class="slider multiple-items">
    {{#releases}}
      <div class="Discography-item">
        {{> molecules-ReleaseItem }}
      </div>
    {{/releases}}
  </slick>
</div>

```

Kuvio 3 Pattern Labin tuottama koodi diskografiaelementistä

3.5 Komponenttitekniikoiden mobiilituki

3.5.1 Polymer

Polymerin sivuaihioita voidaan yhdistää mobiilikehysten avulla mobiilisovelluksiksi. Esimerkiksi Apache Cordovalla on saatu aikaiseksi sovellus Polymerin kanssa. Apache Cordova on sovelluskehys, joka mahdollistaa hybridisovellusten, eli HTML5–elementtien ja mobiilin yhdistämisen toimivaksi mobiilisovellukseksi. (Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options, n.d).

3.5.2 ReactJS

Mobiiliapplikaatioiden kehittäjät ovat pohtineet, tehtäisiinkö applikaatio Androidin tai iOS-käyttöjärjestelmien omien kehitystyökalujen avulla vai HTML5:n avulla jonkinlainen versio, joka toimii sekä mobiililla että tietokoneella. Jälkimmäisessä on etunsa, sillä samalla voidaan tehdä molemmat, mutta parhaimman käyttökokemuksen ja tehokkuuden kannalta olisi hyvä tehdä applikaatiot niille tarkoitetuilla työkaluilla. Natiiveilla tekniikoilla tehtäessä ongelmaksi jää vielä se, että pitäisi tehdä Androidille ja iOS:lle sovellukset, missä vaaditaan kahta täysin erillistä koodausta. (Yet another reason your favorite iPhone app isn't on Android, 2015)

Tammikuussa 2015 Facebook julkaisi React Nativen. React Native antaa keinon luoda suoraan mobiilisovelluksia, joita kehitetään ReactJS:n avulla. Applikaatiot luodaan JavaScriptiä käyttäen, joka toimii kaikissa mobiililaitteissa, ja silti voidaan käyttää natiiveja ratkaisuja. Kehittäjien pitää silti tehdä eri sivut työpöytäympäristöille ja mobiililaitteille, mutta heidän tarvitsee vain oppia yksi koodiympäristö, jota voidaan käyttää molempiin. Tekniikka myös parantaa työtahtia, koska React Native nopeuttaa sitä poistaessaan viiveitä, jotka aiheutuvat koodin kääntämisestä ja valmistumisesta. Reaktiivisella ratkaisulla saadaan suoraan muutokset näkymään, ja on mielekkäämpää kehittää applikaatiota tietokoneella, koska silloin nähdään suoraan muutokset mobiililaitteessa. (React Native: Bringing modern web techniques to mobile. 2015) React Nativella on rakennettu muun muassa Discoveryn virtuaalitodellisuussivusto, Facebookin Ryhmät-applikaatio ja pelaajille tarkoitettu keskustelusovellus Discord. (React Native n.d.)

3.5.3 AngularJS

Angularille on kehitetty Ionic-rajapinta, joka pyrkii olemaan Twitter Bootstrapin kaltainen valmis UI-pohja, josta on helppo kehittää jokaiseen kännykkään miellyttävän näköinen ja käyttöystävällinen mobiiliapplikaatio. Ionic ei toimi yksinään vaan vaatii mobiilikääntäjän, joka kääntää koodin mobiileille yhteensopivaksi. Tällaisia ratkaisuja ovat mm. PhoneGap ja Apache Cordova. Ionicin vahvuuksia ovat Angularin monimuotoinen ja joustava alusta, liitännäisten suuri laajuus sekä se, että Ionic on käytetyin mobiilisovelluskehys. (Comparing The Top Frameworks For Building Hybrid Mobile Apps 2015;

Ionic n.d.) Ionicilla on rakennettu muun muassa Lemon Pie -reseptiapplikaatio, Swor-
kit-treeniohjelma ja Estimated, ketterän kehityksen tarkkailuun tarkoitettu applikaatio.

4 Tekniikan valinta

Tekniikan valintaan käytettiin seuraavia kriteerejä:

- Miten ne lukevat REST-rajapintaa?
- Miten design/tyylien muokkaus onnistuu?
- Miten sivujen reititys eli käyttäjän ohjaaminen oikealle sivulle onnistuu?
- Miten hakukoneystävällisiä tekniikat ovat, eli miten Googlen kanssa indeksointi sujuu?
- Kuinka helppoa on käyttää komponentteja uudelleen, ja kuinka komponentteja itse osat ovat?
- Mobiilituki
- Nopeus

Jokaisesta kriteeristä annettiin arvosana 1–3.

1 = monimutkainen/ ei (vielä) toimi, 2 = toimii jotenkuten, 3 = toimii erinomaisesti

4.1 Polymer

4.1.1 Rest-kutsut

Polymer lukee REST-rajapintaa AJAX-elementtiä käyttäen. Luodaan oma kustomoitu elementti, jonka läpi ajetaan RESTiä lukevat ajax-kutsut.

```
<template>
<div >
  <iron-ajax auto
    handle-as="json",
    url="http://musicinfo.io/json/v1/artist/87c5dedd-371d-4a53-9f7f-80522fb7f3cb/releases",
    method: 'GET',
    last-response="{{artistsLoaded}}" ></iron-ajax>
```

Kuvio 4. Polymerin REST-pyyntö (Using Polymer's iron-ajax to load json files n.d.)

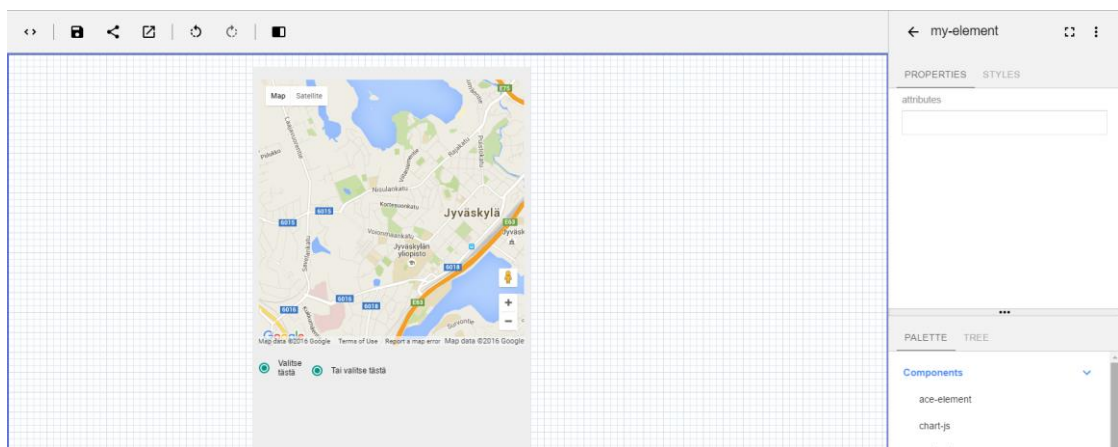
REST-pyyntö suoritetaan AJAX-funktiolla, jossa ajetaan dataa käsitteleviä parametreja. Ensimmäiseksi määritellään funktio automaattisesti päivittyväksi auto-parametrillä. Seuraavaksi määritellään sitä, minkälaisena data käsitellään, tässä tapauksessa JSON-muodossa. Osoite, josta tieto halutaan saada, määritellään url-parametrilla. Jotta tieto saataisiin RESTistä loppukäyttäjälle, määritetään metodi, miten sitä käsitellään, tässä

tapauksessa GET, eli haetaan tietoa. Lopuksi annetaan muuttuja, jonka avulla voidaan käsitellä tullutta dataa ja manipuloida sitä. (Using Polymer's iron-ajax to load json files n.d.)

Arvosana 3, helppo ymmärtää ja toteuttaa

4.1.2 Design

Polymer on kehittänyt designille kokonaan oman Designer-netissä käytettävän työkalunsa, jolla voidaan raahaa ja pudota -tyylillä suunnitella design-ratkaisuja ja nähdä, millaista koodia siitä syntyy. Käyttäjä voi tallentaa luomuksensa GitHubiin myöhempää käyttöä varten. (Learning Web Component Development 2015.)



Kuvio 5. Polymerin designer-työkalu (Designer, n.d)

```

1 <link rel="import" href="../../ace-element/ace-element.html">
2 <link rel="import" href="../../core-animat-ed-pages/core-animat-ed-pages.html">
3 <link rel="import" href="../../core-animat-ed-pages/transitions/hero-transition.html">
4 <link rel="import" href="../../core-animat-ed-pages/transitions/cross-fade.html">
5 <link rel="import" href="../../core-animat-ed-pages/transitions/slide-down.html">
6 <link rel="import" href="../../core-animat-ed-pages/transitions/slide-up.html">
7 <link rel="import" href="../../core-animat-ed-pages/transitions/tile-cascade.html">
8 <link rel="import" href="../../yt-video/yt-search-video.html">
9 <link rel="import" href="../../paper-radio-button/paper-radio-button.html">
10 <link rel="import" href="../../paper-toast/paper-toast.html">
11 <link rel="import" href="../../paper-toggle-button/paper-toggle-button.html">
12 <link rel="import" href="../../google-map/google-map-directions.html">
13 <link rel="import" href="../../google-map/google-map.html">
14
15 <polymer-element name="my-element">
16
17 <template>
18 <style>*</style>
131 <ace-element id="ace_element">function test() {
132 var x = true;
133 }</ace-element>
134 <core-selector selected="0" id="core_selector"></core-selector>
135 <core-animat-ed-pages selectedindex="0" notap id="core_animat-ed_pages">
136 <section id="section2" active>
137 <section id="section1">
138 <yt-search-video id="yt_search_video"></yt-search-video>
139 <core-selector selected="0" id="core_selector1"></core-selector>
140 <paper-radio-button checked label="Valitse tästä" id="paper_radio_button"></paper-radio-button>
141 <paper-toast text="Toast!" id="paper_toast" touch-action="none" class="core-transition-bottom core-transition"></paper-toast>
142 <paper-toggle-button id="paper_toggle_button" touch-action="pan-y"></paper-toggle-button>
143 </section>
144 <google-map-directions id="google_map_directions"></google-map-directions>
145 <google-map id="google_map"></google-map>
146 <google-map latitude="62.24259510932172" longitude="25.7264082470703" zoom="13" id="google_map1"></google-map>
147 <paper-radio-button checked label="Tai valitse tästä" id="paper_radio_button1"></paper-radio-button>
148 </section>
149 </core-animat-ed-pages>
150 </template>

```

Kuvio 6. Designerin tuottama koodi (Designer, n.d)

Arvosana: 3, erottuu muista täysin omalla design-työkalullaan

4.1.3 Reititys

URLien luonti eli applikaation sisällä tapahtuva reititys onnistuu app-router-nimisen lisäosan kanssa, joka on tarkoitettu juuri web-komponenttien reititystä varten. Myös paljon suosiota on kerännyt flatiron-reititin, mikä toimii myös saumattomasti Polymerin kanssa. (Router for Web Components n.d. ; Building single page apps using web components. 2014.)

Arvosana 3

4.1.4 Hakukoneystävällisyys

Hakukoneystävällisyys Polymerin kohdalla on kehittymässä. Vaikka Polymer piilottaa kaiken sisällön varjo-DOMiin, Googlen haku on kehittynyt sen verran, että se osaa etsiä sisältöä sieltä, mutta ei vielä täydellisesti. (Introducing Web Components and What It Means for Search Engine Optimization and Privacy 2014; What about SEO? 2015.)

Arvosana 2

4.1.5 Renderöinti/ koodin haastavuus

Itse Polymerin koodi on yksinkertaista HTML-koodausta, joten oppimiskäyrä kehittäjillä on suhteellisen alhainen tämän tekniikan kanssa.

```
<p><b>Discography:</b></p>
<div style="height: 200px; overflow: scroll; margin-bottom: 20px; width: 610px;">
<template is="dom-repeat" items="{{artistsLoaded.releases}}" as="release">
<div style=""><span><table><tr> <td style="border: 1px solid black;">{{release.name}}
```

Kuvio 7. Polymerin datan renderöinti (Data binding helper elements, n.d)

Luodaan ensin palikka, johon tieto tulostuu, eli <div>-elementti ja sen sisälle aihio eli <Template>, jonka sisälle Polymer ymmärtää tulostaa REST-pyyntöillä haetun datan. Items-kohdassa määritetään, mitä sieltä datasta otetaan, ja as-attribuutilla annetaan muuttuja datalle. Sitten <Template>:lle määritellään "dom-repeat"-arvo, joka määrittää templaatin sellaiseksi, että se ymmärtää käydä läpi RESTistä tulevaa dataa. Sen sisälle luodaan erinäisiä elementtejä, jotka voivat olla taulukkoja tai jopa perus <p>-leipätekstielementtejä, joiden sisälle tuplakaarisulkein data tulostetaan. (Data binding helper elements, n.d)

Arvosana 3

4.1.6 Mobiilituki

HTML5-tekniikka soveltuu hyvin mobiilikehitykseen. Apache Cordova –mobiilikehityksen avulla HTML-elementit voidaan kääriä mobiililla tunnistettavaksi elementeiksi. Ne eivät ole kännykän natiivein ominaisuuksien rakennettuja sovelluksia, vaan hybridisovelluksia, eli HTML5-elementtien ajamista mobiilikehityksessä. (Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options, n.d; How to Create a To-Do List App With Polymer and Cordova 2016)

Arvosana 2

4.1.7 Web-komponentit

Polymer tekee juurikin W3C:n standardisoimia web-komponentteja, mutta kyseinen prosessi on vielä kesken ja selaintuki vaihteleva, joten pitäisi odottaa vielä hetki, jotta web-komponentit olisivat täysin tuettuja ja parhaimmassa kannattavuudessaan. (Are We Componentized Yet?. 2015)

Arvosana 2

4.2 React

4.2.1 Rest-kutsut

Reactissa voidaan hakea RESTistä tiedot komponenttiin, joka siirtää ne state-tilaan, joka aktivoi käyttöliittymän uudelleenrenderöinnin.

```
var Discography = React.createClass({
  getInitialState: function() {
    return {data: []};
  },
  componentDidMount: function() {
    var self = this;
    $.get(this.props.url, function(result) {
      var discoq = result.releases;
      if (this.isMounted()) {
        this.setState({
          data: discoq
        });
      }
    }).bind(this);
  },
});
```

Kuvio 8. Reactin REST-pyyntökokonaisuus (React n.d)

Reactin REST-funktio aloitetaan sillä, että tehdään toinen funktio, jolla haetaan datan kyseinen state eli Reactissa oleva tiedoston tila. Tämä pitää hakea, jotta myöhemmin voidaan käyttää sitä tilaa eri muuttujissa. Kun tila on haettu ja se on saatavilla, tehdään funktio, jonka avulla React hakee kaiken datan. Aluksi haetaan määritelty osoite, josta data haetaan. Se on monesti määritelty tiedoston alalaidassa tehdyn renderöintikäskyn ohessa. Saatua tietoa voidaan tarkentaa result-parametrin avulla, johon tässä

tapauksessa liitetään releases-liite, jolla tarkennetaan, että halutaan hakea juuri disko-
grafia palvelimelta. Kaikki tämä tallennetaan discoq-nimiseen muuttujaan, jota voi-
daan käyttää myöhemmin dataa tulostettaessa. Jos REST-pyyntö onnistuu, saatu si-
sältö siirretään Reactin ymmärtämäksi dataksi, joka haettiin aiemmin InitialState-pyyn-
nön avulla. Lopuksi koko funktio sidotaan "this"-muuttujaan, jotta sen käyttö helpot-
tuisi. (React n.d.)

Arvosana 2, Reactin oppimiseen pitää käyttää muita tekniikoita enemmän aikaa.

4.2.2 Design

Tyylilien kirjoitus onnistuu Reactissa muuttujien avulla. Ensin määritellään muuttuja,
jonka sisällä ovat tyylimäärittelyt ja joihin sitten viitataan "style"-attribuutin sisällä.
Reactissa määrittelyt eivät ole merkkijonoja vaan objekteja, joita sitten renderöidään
käyttöliittymässä. (Inline Styles. n.d)

Arvosana 3, helppoa ja tehokasta

4.2.3 Reititys

Reititykseen on olemassa Javascript-paketteja, jotka ovat kevyitä ja Reactia varten
suunniteltuja. Varteen otettavana reititys lisäosana on Routie-niminen erittäin kevyt
ohjelma, jolla voidaan hoitaa Reactin reititys yksinkertaisen tehokkaasti. Toisena
vaihtoehtona on käyttää React-tiimin kehittämää React-Routeria, mikä on täydellinen
reitityspaketti Reactille. (Using Routie for React.js n.d.; react-router n.d)

Arvosana 3, lukuisia eri paketteja, joita käyttää.

4.2.4 Hakukoneystävällisyys

Reactin hakukoneystävällisyys voidaan hoitaa koa-prerender-nimisellä osalla, jolla
voidaan esirenderöidä sivusto HTML:ksi, jotta hakukoneet voivat löytää sivun.
Javascript-pohjaisten käyttöliittymien huono puoli onkin juuri siinä, että hakukoneet
löytävät niitä todella heikosti, koska ei ole staattista sisältöä vaan lennossa
renderöityä. (From AngularJS to React: The Isomorphic Way 2014.)

Arvosana 2.5, on olemassa keino toteuttaa, mutta suoraa keinoa ei ole.

4.2.5 Mobiilituki

Reactilla on React Native, jolla voidaan natiivisti toteuttaa applikaatioita, eli erinomainen valinta mobiilikehittämiseen. Lisää luvussa 3.5

Arvosana 3

4.2.6 Renderöinti/ koodin haastavuus

Reactissa on testatuista tekniikoista suurin oppimiskynnys. Suurin osa ohjelmista tehdään puhtaalla JavaScriptillä, jota ei ole totuttu käyttämään näin suurissa määrin.

Polymer ja Angular käyttävät renderöinnissään lähes kokonaan HTML-merkintää JQueryn kera kun React käyttää Javascriptiä suurimmaksi osiksi. Itse renderöinti sivulle tapahtuu HTML-merkinnällä, mutta ympäröivät asiat on puhtaasti JavaScriptiä.

```
render: function() {
  Discoq= this.state.data;
  return (
    <div>
    <b>Discography:</b>
    <div style={td2Style}>
      {Discoq.map(function(disco){
        return <div style={urlStyle}>
          <table><tr><td style={tdStyle}>{disco.name}</td></tr></table>
        </div>
      })}
    </div>
  )
}
```

Kuvio 9. Reactin datan renderöinti (Tutorial. n.d)

Reactissa tietojen renderöinti on monimutkaisin, mutta kun kaavan oppii, samaa kaavaa käytetään kaikissa sovelluksissa. Ensiksi määrätään Discoq-muuttuja Reactin datan tilaksi, jossa sitä voidaan käsitellä. Sitten return-komennon myötä tietoja voidaan käsitellä. Discoq-muuttuja pitää ensiksi kartoittaa läpi, eli map-funktiolla ajetaan tietojoukko läpi niin, että jokainen tietue saa arvon, johon viitata. Sitten ajetaan toinen return läpi, jossa muusta poiketen HTML-koodilla viitataan tietoihin. (Tutorial, n.d)

```
React.render(
  <Discography style={urlStyle} url="http://musicinfo.io/json/v1/artist/01809552-4f87-45b0-afff-2c6f0730a3be/releases/" />,
  document.getElementById('disco-content')
);
```

Kuvio 10 Reactin datan renderöinti etusivulle (Tutorial. n.d)

Lopuksi sivun viimeisenä elementtinä määrätään aiemmin käytetty URL ja ohjataan käsitelty data elementtiin indeksisivulle, jossa sitä voidaan käsitellä.

Arvosana 2, suurin oppimiskynnys

4.2.7 Web-komponentit

Reactin komponentteja voidaan täydellisesti käyttää uudelleen, ja ne toimivat kaikissa selaimissa. React tukee jopa vanhempiakin selaimia erinäisten kiertoteiden kautta.

Suurimpia eroja muihin web-komponentteihin on se että Polymer pyrkii käyttämään selaimen natiiveja ominaisuuksia toteuttaakseen ratkaisunsa. React on kokonaan erillinen ratkaisunsa.

Arvosana 3

4.3 AngularJS

4.3.1 Rest-kutsut

Angularilla voidaan hakea tietoa REST-apeista todella helposti ngResource-moduulia käyttäen. Yksinkertaisemmissa tilanteissa se onnistuu käyttämällä http.get-pyyntöä (ks. Taulukko 3).

```
app.controller('discographyCtrl', function($scope, $http){
  $http.get('http://musicinfo.io/json/v1/artist/01809552-4f87-45b0-afff-2c6f0730a3be/releases')
    .success(function(response,status, headers, config) {
      $scope.releases = response.releases;
      console.log($scope.releases);
    }).error(function(data, status, headers, config) {
      // log error
    });
});
```

Kuvio 11 Angularin REST-kutsu (Requesting JSON data with AJAX. N.d)

Tehdään RESTille get-kutsu, määritellään osoite, ja kun tämä kutsu onnistuu, määritellään funktio, joka laittaa saadun datan releases-nimiseen scopeen. Selvyden vuoksi itse laitoin vielä sen perään tulemaan konsoliin viestin, joka näyttää juuri tehdyn releases-scopen sisällön. Näen siitä, tuleeko mitään läpi vai näyttääkö tyhjää. Perässä on virhe-funktio, joka tekee jonkin käyttäjän määrittelemän toimenpiteen, kun virhe ilmaantuu. (Requesting JSON data with AJAX n.d.)

Arvosana 3, yksinkertainen ja helppo

4.3.2 Design

Angularille on tehty Polymerin kaltainen Material Design–designmalli, jota voidaan käyttää rakentamaan tyylikkäitä visuaalisia kokonaisuuksia. Ilman sitä on myös helppo visualisoida CSS/SASS-tyylitiedostoilla. (Angular Material – Introduction, n.d)

Arvosana 2.5

4.3.3 Reititys

Angularille on tarjolla lisäosia ja myös oma ngRoute-moduuli, jolla voidaan hoitaa reititys tehokkaasti natiivisti. (AngularJS Documentation for ngRoute, n.d).

Arvosana 3

4.3.4 Hakukoneystävällisyys

Angularilla on ollut paljon ongelmia hakukoneiden kanssa. Siihen keksittiin avuksi hashbang-tekniikka, jossa #-merkatut urlit ohittivat palvelinpuolella monesti hoidetun renderöinnin ja renderöivät koko sivun yhdeksi isoksi merkkijonoksi, jonka Google pystyi näyttämään hakutuloksissa. Tämä tekniikka, joka oli selvästi väliaikainen ratkaisu, johti monesti sivujen kaatumisiin ja liian monimutkaisiin URLeihin. HTML5:n myötä saapui pushState-tekniikka, jonka avulla sivu ei päivity erikseen, vaikka sivun sisällä navigoidaan, ja tässä tapauksessa hakukoneet voivat indeksoida sen paremmin. (AngularJS: How to setup pushState with html5Mode n.d.)

Arvosana 2, keino on, mutta se ei ole aukoton.

4.3.5 Renderöinti/ koodin haastavuus

```
<div ng-controller="discographyCtrl">
<b>Discography: </b>
<p></p>
<div style="height: 200px; overflow: scroll; width: 610px;">
<table> <tr ng-repeat="release in releases">
<td style="border: 1px solid black; width: 400px;">{{release.name}}</td>
</tr></table>
</div>
```

Kuvio 12 Angular renderöinti/koodin haastavuus

Itse renderöinti on yksinkertaista Angularissa. Tehdään <div>-elementti, jossa määritetään, mitä kontrolleria käytetään. Tässä tapauksessa diskografian hakuun käytetään diskografian kontrolleria. Sitten määritetään jonkinlainen aihio tulostetulle datalle, jonka itse näin aiheelliseksi tehdä taulukkomuodossa, jotta kaikki data tulisi hyvin järjestettynä. Yhdessä taulukon ylläpitävässä elementissä määrittelen, mitä haetaan, eli kaikki diskografian sisältämät yksittäiset julkaisut. Annetaan jonkinlainen alias, tässä tapauksessa 'release', ja scopen nimi eli 'releases'. Sitten voimme tulostaa tuplakaarisulkeita käyttäen. Tässä tapauksessa haetaan jokaisen julkaisun nimi.

Arvosana 3

4.3.6 Mobiilituki

Ionic on erittäin kattava ja tehokas mobiilisovelluskehys, jota voidaan käyttää isoihin-kin applikaatioihin. Käsittelen tätä asiaa lisää luvussa 3.5.

Arvosana 3

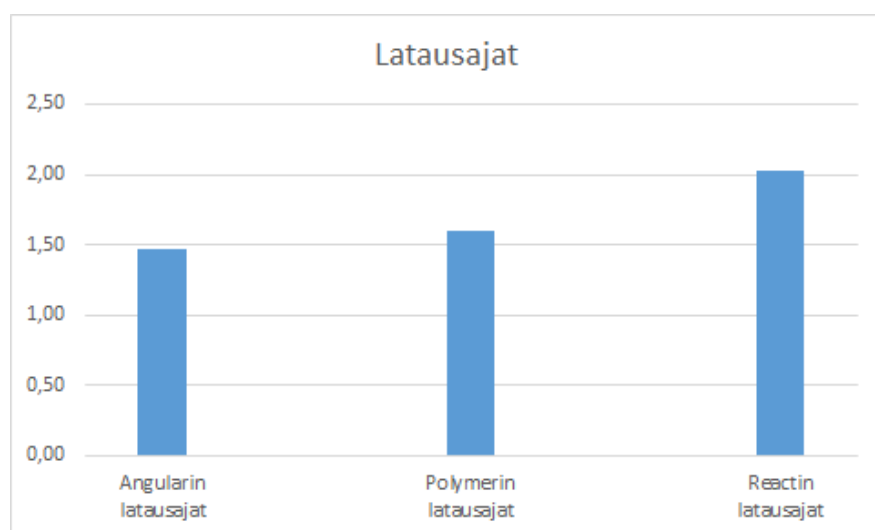
4.3.7 Web-komponentit

Nykyistä Angular-versiota voidaan pitää vanhanaikaisena, ja sillä ei voi suoranaisesti rakentaa web-komponentteja. Direktiivien kautta voidaan rakentaa niitä muistuttavia elementtejä, mutta niitä ei pidä verrata muiden tekniikoiden komponentteihin. Angular on julkaissut opinnäytetyön teon aikana Angular 2.0 beeta -version, joka auttaa luomaan natiiveja web-komponentteja. Angular 2.0:n tarkoitus on tehdä

komponenteista reaktiivisia, joten uuden Angularin kanssa voisi myös käyttää Flux-rakennetta. (Building Angular apps using Flux architecture 2015.)

Arvosana 1.5, Beta-versiossa on kykenevyys web-komponenteille, mutta en vielä miettisi beta-versiota vaihtoehtona.

4.4 Nopeus



Kuvio 13. Tekniikoiden latausajat

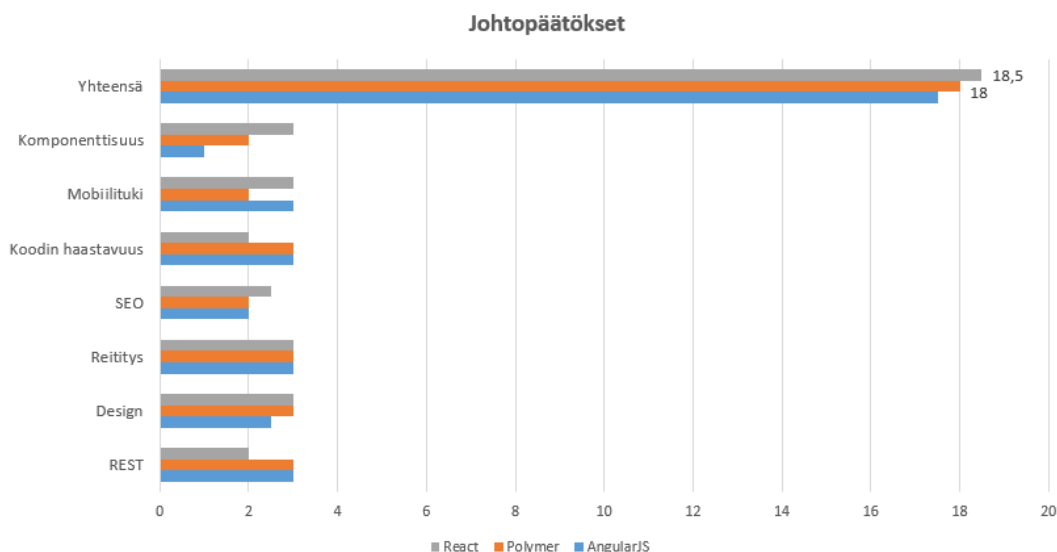
Suoritin 10 latausta jokaisella tekniikalla ja laskin niiden keskiarvon. Kuten huomataan, kovin suuria eroja ei ole. Reactin yksisuuntainen data-kierto voi olla parempi ei-interaktiivisille ohjelmille, kun Angularin tarjoama kaksisuuntainen datavirtaus voisi olla aivan liian raskas.

AngularJS:n datan renderöinnin (ng-repeatin) raskautta (AngularJS: My solution to the ng-repeat performance problem 2013) voidaan keventää korvaamalla ng-repeat ominaisuus ngReact-komponentilla, mikä käärii Reactin nopean renderöinnin Angular-komponentiksi. (NgReact - React Components in Angular 2013; Improving AngularJS long list rendering performance using ReactJS 2014)

5 Johtopäätökset

Annoin kaikista kriteereistä pisteet ja tein johtopäätöksen laskemalla ne yhteen.

	REST	Design	Reititys	SEO	Koodin haastavuus	Mobiilituki	Komponenttisuus	Yhteensä
AngularJS	3	2,5	3	2	3	3	1	17,5
Polymer	3	3	3	2	3	2	2	18
React	2	3	3	2,5	2	3	3	18,5



Kuvio. 14 Johtopäätökset

Kun laskin pisteet yhteen, React nousi sopivimmaksi tekniikaksi. Polymer on myös vartenotettava vaihtoehto, kunhan W3C onnistuu standardisoimaan web-komponentit ja tekniikan yleistyessä selaintuki laajenee.

Reactina etuna on myös React Native, jolla MusicInfo voi kehittää palvelulleensa mobiilisovelluksen, joka toimii natiivisti kaikilla alustoilla. Reactin takana on Facebookin kehitystiimi, ja React onkin Githubissa yksi kovimmassa nousussa oleva avoimen lähdekoodin projekti. Reactin eduksi voidaan myös laskea sen pienkokoisuus. Kuten Mikko Forsström artikkelissaan mainitsee: ”Pienet kirjastot vastaavat hänen mukaansa uudelleenkäytettävyyden haasteeseen paremmin kuin isot sovelluskehyykset. Pienet osat on helpompi vaihtaa, jos ja kun ne vanhentuvat.” (Tivi 2016.)

On olemassa myös keinoja, miten voidaan käyttää web-komponentteja Reactin sisässä (Combining React, Flux & Web Components 2014.) ja Reactia Angularin sisässä (NgReact - React Components in Angular. 2013)

Reactin etuna oli myös sille kehitetty Patternity-työkalu, joka parantaa huomattavasti yhteistyötä Pattern Labin kanssa. Polymerin ja Angularin toimivuudesta Pattern Labin kanssa en löytänyt mitään dokumentaatiota.

Lähteet

About Atomic Design. N.d. Viitattu 10.2.2016. <http://patternlab.io/about.html>

AngularJS: How to setup pushState with html5Mode. N.d. Viitattu 10.3.2016. <http://www.codelord.net/2015/05/12/angularjs-how-to-setup-pushstate-with-html5mode/>

AngularJS Documentation for ngRoute. n.d Viitattu 15.1.2016 <https://docs.angularjs.org/api/ngRoute>

Angular Material – Introduction. N.d Viitattu 21.3.2016. <https://material.angularjs.org/>

Are We Componentized Yet?. 2015. Viitattu 20.1.2016 <http://ionrimmer.github.io/are-we-componentized-yet/>

Building Angular apps using Flux architecture. 2015. Viitattu 16.3.2016. <http://victorsavkin.com/post/99998937651/building-angular-apps-using-flux-architecture>

Building single page apps using web components. 2014. Viitattu 31.3.2016

Comparing The Top Frameworks For Building Hybrid Mobile Apps. 13.10.2015. Viitattu 15.2.2016. <http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps/>

Designer. n.d Viitattu 21.3.2016 <https://polymer-designer.appspot.com/>

Flux. N.d. Viitattu 15.1.2016. <https://facebook.github.io/flux/>

Inline Styles. n.d <https://facebook.github.io/react/tips/inline-styles.html>

Introducing Web Components and What It Means for Search Engine Optimization and Privacy. 9.3.2014. Viitattu 21.3.2016. <https://medium.com/cool-code-pal/introducing-web-components-and-what-it-means-for-search-engine-optimization-and-privacy-b21bfc1f63c7#.hwc9n5v2l>

Ionic. N.d. Viitattu 15.2.2016. <http://ionicframework.com/>

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

Material Design. N.d. Viitattu 15.2.2016. <https://www.google.com/design/spec/material-design/introduction.html>

Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. N.d Viitattu 21.3.2016 https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Patternity. N.d. Viitattu 10.2.2016. <https://github.com/influitive/patternity>

React. N.d. Viitattu 21.3.2016. <https://facebook.github.io/react/tips/initial-ajax.html>

React Native. N.d, Viitattu 10.2.2016, <https://facebook.github.io/react-native/>

React Native: Bringing modern web techniques to mobile. 2015. Viitattu 20.2.2016 <https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/>

react-router. N.d. Viitattu 31.3.2016 <https://github.com/reactjs/react-router>

Requesting JSON data with AJAX. N.d. Viitattu 16.3.2016 <http://fdietz.github.io/recipes-with-angular-js/consuming-external-services/requesting-json-data-with-ajax.html>

Router for Web Components. N.d. Viitattu 20.2.2016. <https://github.com/erikringsmuth/app-router>

Sandeep, Kumar Partel 2015. Learning Web Component Development. Ebook.

Shapiro, D. 2015. Web Component Architecture & Development with AngularJS. Viitattu 14.3.2016. Ebook.

WebComponents N.d, Viitattu 14.3.2016. <http://webcomponents.org/>

What Is Rest. N.d viitattu 10.1.2016. <http://www.restapitutorial.com/lessons/whatisrest.html>

Why React? N.d viitattu 10.1.2016. <https://facebook.github.io/react/docs/why-react.html>

REST Architectural Goals. N.d. Viitattu 10.1.2016. http://whatisrest.com/rest_architectural_goals/index

Tutorial. n.d. Viitattu 10.1.2016 <https://facebook.github.io/react/docs/tutorial.html>

Using Polymer's iron-ajax to load json files. N.d. Viitattu 16.3.2016. <http://danieltse.com/using-polymer-iron-ajax-to-load-json-files/>

Using Routie for React.js. N.d. Viitattu 16.3.2016, <http://www.addthis.com/blog/2014/12/19/using-routie-for-react-js/#.Vu6GmvmLRhF>

Vänskä, O. 2016. Javascript-kehittäjä valitsee työkalupakin. Julkaisussa. Tivi. Helsinki: Talentum, 40–47.

Web Components. N.d. Viitattu 10.1.2016. <https://github.com/w3c/webcomponents>

Web Services: JSON vs. XML. N.d. Viitattu 10.1.2016. <http://digitalbazaar.com/2010/11/22/json-vs-xml/>

What about SEO? 1.10.2013. Viitattu 21.3.2016.

<https://www.youtube.com/watch?v=inllyR7hN8M>

Working With the Browser. N.d. Viitattu 15.3.2016.

<https://facebook.github.io/react/docs/working-with-the-browser.html>

Yet another reason your favorite iPhone app isn't on Android. 2015. Viitattu

15.1.2016 <http://bgr.com/2015/10/30/ios-vs-android-apps-development/>

Liitteet

Kuten huomataan alla olevissa kuvissa, visuaalisia eroja ei toteutustekniikoilla tapahtunut.

Liite 1. Kuvankaappaukset rakennetuista sivuesimerkeistä

Artist: Elvis Presley

Discography:

Mid South Coliseum Memphis TN 16th March 1974 Evening Show
King Creole (El Barrio Contra Mi) Vol. 2
Elvis Presley
Elvis Talks! The Elvis Presley Interview Record: An Audio Self-Portrait
The Bosom Of Abraham / He Touched Me
Love, Elvis

Reviews:



[A bizarre release that fans of The King probably won't rush to buy.](#)



[He genuinely was one of the greatest vocalists who ever drew breath.](#)



[It's a gift that keeps on giving.](#)

Artist name: Elvis Presley

Discography:

- Mid South Coliseum Memphis TN 16th March 1974 Evening Show
- King Creole (El Barrio Contra Mi) Vol. 2
- Elvis Presley
- Elvis Talks! The Elvis Presley Interview Record: An Audio Self-Portrait
- The Bosom Of Abraham / He Touched Me
- Love, Elvis
- Tryin' To Get To You
- Love Songs

Reviews:



A bizarre release that fans of The King probably won't rush to buy.



He genuinely was one of the greatest vocalists who ever drew breath.



It's a gift that keeps on giving.

Artist: Elvis Presley

Discography:

Mid South Coliseum Memphis TN 16th March 1974 Evening Show

King Creole (El Barrio Contra Mi) Vol. 2

Elvis Presley

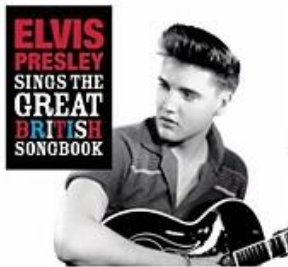
Elvis Talks! The Elvis Presley Interview Record: An Audio Self-Portrait

The Bosom Of Abraham / He Touched Me

Love, Elvis

Tryin' To Get To You

Reviews:



[A bizarre release that fans of The King probably won't rush to buy.](#)



[He genuinely was one of the greatest vocalists who ever drew breath.](#)



[It's a gift that keeps on giving.](#)