jamk.fi

SPA-sovellusten hakukoneoptimointi

ReactJS-pohjaisen sovelluksen indeksoituminen

Henna Sauranen

Opinnäytetyö
Joulukuu 2017

Liiketaloudenala
Tradenomi (AMK), Tietojenkäsittelyn koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Sauranen, Henna	Opinnäytetyö, AMK	Joulukuu 2017
	Sivumäärä 50	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
SPA-sovellusten hakukoneoptimointi
ReactJS-pohjaisen sovelluksen indeksoituminen

Tutkinto-ohjelma
Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)
Tommi Tuikka

Toimeksiantaja(t)
Solteq

Tiivistelmä

Hakukoneet on alun perin rakennettu ymmärtämään vain perinteisiä palvelinrenderoituja sivuja. SPA-sovellukset ovat keränneet suosiotaan perinteisten verkkosivujen rinnalla nopean käytettävyytensä ja kevyen rakenteensa ansiosta. SPA-sovelluksissa näkymien muodostuminen tapahtuu selaimessa JavaScriptin avulla. Tätä kutsutaan selainrenderoinniksi. Selainrenderoinnin vuoksi on kyseenalaistettu hakukoneiden kykyä löytää, tulkita ja indeksoida SPA-sovelluksia.

Tutkimuksen tavoitteena on selvittää, onko SPA-sovellusten indeksoituminen Googleen sellaisenaan mahdollista vai onko tarve turvautua palvelinpuolen renderointiin selainpuolella näkymien muodostamisen ohella.

Opinnäytetyö toteutetaan kehittämistutkimuksena, sillä tutkimuksessa pyritään kehittämään Solteqin toimintaa SPA-sovellusten hakukoneoptimoinnin osalta.

Opinnäytetyön tuotoksena valmistunut ReactJS-testisovellus indeksoitui Googleen, kun Googlen indeksoitumisenkriteerit otettiin huomioon.

Opinnäytetyön tuloksena tuotettiin kaksi ReactJS sovellusta: tavallinen sekä isomorfinen ReactJS-sovellus, joka kykenee tarjoamaan palvelinpuolen renderoinnin SPA-toimintojen ohelle. Työssä tutkittiin myös sitä, kuinka isomorfinen SPA-sovellus rakennetaan.

Työn lopputuloksena päädyttiin siihen, että Googlen lisäksi muut hakukoneet eivät kykene suorittamaan JavaScriptia tarpeeksi hyvin. Ainoa mahdollisuus indeksoitua näiden hakukoneiden tietokantoihin on tehdä isomorfinen SPA-sovellus, joka takaa hakukoneoptimoitavuutensa lisäksi myös parhaan käytettävyyden SPA-tekniikoin. Selainrenderoinnin pettäessä sovellus perääntyy palvelinrenderointiin.

Avainsanat (asiasanat)
SPA, single page application, ReactJS, SEO, hakukoneoptimointi, isomorfinen JavaScript, universaalinen sovellus

Muut tiedot

jamk.fi

Kuvailulehti

Author(s)	Type of publication	Date
Sauranen, Henna	Bachelor's thesis	December 2017
		Language of publication: Finnish
	Number of pages 50	Permission for web publication: x

Title of publication

Search engine optimization for SPA

Indexing ReactJS-based application for SEO

Degree programme

Business Information Technology

Supervisor(s)

Tuikka Tommi

Assigned by

Solteq

Abstract

Search engines were originally built to understand only websites that use server-side rendering. Single page applications (SPA) have gathered plenty of attention due to providing a fluid user experience with their efficient usability and lightweight structure. SPA technology uses JavaScript to create views in the browser, which is called client-side rendering. Because of client-side rendering, the ability of search engines to find, crawl and index SPAs has been questioned.

The goal of the research was to find out if it is possible to index SPAs into Google's database as they are, or does the application need server-side rendering alongside client-side rendering.

This thesis was carried out as a development research, as the research aims to improve Solteq's knowledge about the search engine optimization of SPAs. The ReactJS test application created as a part of this research was able to index fully into Google as long as Google's indexing criteria were taken into consideration.

Two ReactJS applications were produced as a part of this thesis: a regular one as well as an isomorphic application that uses server-side rendering alongside the traditional client-side rendering. Building an isomorphic structure was researched in this thesis.

The research resulted in the discovery that other search engines besides Google were not able to execute JavaScript properly. The only possibility to index SPA in their databases was to use an isomorphic application structure that ensures not only its ability to render content for all search engines to be able to reach it but it also gives the best user experience with SPA's client-side rendering.

Keywords/tags (subjects)

SPA, Single Page Application, ReactJS, SEO, Isomorphic JavaScript

Miscellaneous

Sisältö

1 Johdanto 6

1.1 Toimeksiantaja, työn tavoitteet ja rajaukset 6

1.2 Tutkimusmenetelmät 7

1.3 Tutkimuskysymykset 8

1.4 Opinnäytetyön tietoperusta ja rakenne 8

2 Hakukoneet ja hakukoneoptimointi 8

2.1 Hakukoneoptimoinnin hyötyjä 10

2.2 Haussa sijoittumiseen vaikuttavat tekijät 11

3 SPA-sovellus 14

3.1 Reititys ja hakukoneet 15

3.2 SPA vs. perinteiset verkkosivut ja sovellukset 17

4 SPA-sovellus ja hakukoneoptimointi 20

5 Sivustokartta 22

5.1 XML-sivustokartan tiedostomuotona 22

5.2 Yleiset ohjeet sivustokartan luomiseen 23

5.3 Sivustokartan lisääminen Googleen ja sivustokarttojen hallinta 24

6 Hakukoneystävällisen ReactJS-sovelluksen luominen 25

6.1 Node.js ja npm 25

6.2 ReactJS 26

6.3 Testisovellus ja sen rakenne 28

6.4 Testisovelluksen indeksoituminen Googleen 29

7 Isomorfinen/universaali web-sovellus 33

8 Isomorfisen/universaalin sovelluksen kehittäminen 34

9 Tutkimustulokset ja johtopäätökset 36

10 Pohdinta 38

Lähteet 40

Liitteet 43

Liite 1. Komponentti, johon tiedot haetaan asynkronisesti. 43

Liite 2. JSON-tiedosto, josta sovellus hakee asykronisesti sisältöä 44

Liite 3. Palvelin saa pyynnön ja käsittelee sen Express.js:lla 45

Liite 4. Redux Actionit kutsutaan retityksessä 46

Liite 5. Testisovellus indeksoitui Googlen tietokantaan 47

Kuviot

Kuvio 1. Title -tagin esiintyminen koodissa ja välilehden otsikkona 11

Kuvio 2. Title -tagin ja metatagin esiintyminen SERP-sivulla 12

Kuvio 3. Canonical atribuutin esiintyminen sivuston head osiossa 13

Kuvio 4. Näkymien muodostuminen palvelinrenderointia käytettäessä 18

Kuvio 5. Näkymien muodostuminen SPA-tekniikoin 19

Kuvio 6. Esimerkki yhden osoitteen XML-sivustokartasta 22

Kuvio 7. Esimerkki monimutkaisemmasta sivustokarttarakenteesta. 23

Kuvio 8 Esimerkki index-sivustokartasta 25

Kuvio 9. Kuva Fetch as Google vertailun tuloksesta 30

Kuvio 10. Helmet -tagin esiintyminen koodissa 31

Kuvio 11. Titlen ja meta kuvauksen muuttaminen asynkronisilla arvoilla 32

Kuvio 12. Sivun esiintyminen Googlen hakutulossivulla 32

Käsitteet

Asynkroninen kutsu	Kun suoritetaan asynkroninen kutsu palvelimelle, sovellukselle palautetaan lupaus tulevasta datasta ja se ei jää odottamaan datan paluuta (kuten synkronisessa kutsussa) vaan ajaa itsensä loppuun. Kun data saapuu, se renderoidaan paikoilleen erikseen.
CSS (Cascading Style Sheet)	CSS on tyylittelykieli, jolla voidaan määrittää, kuinka HTML-elementit näytetään sivulla. CSS-kielellä on mahdollista tehdä sivulle myös animaatioita.
Digitaalinen markkinointi	Digitaalisessa muodossa tehtyä markkinointia. Digimarkkinointiin kuuluu esimerkiksi web-, mobiili- ja sähköpostimarkkinointi, hakukoneoptimointi, hakusanamainonta ja sosiaalisessa mediassa markkinointi.
Google Search Console	Palvelu, jonka avulla voidaan seurata ja pitää yllä sivuston näkyvyyttä Googlen hakutuloksissa.
Hakukone	Hakukone on työkalu, jonka tehtävä on paikantaa verkkosivuja ja julkaisuja, jotka liittyvät käyttäjän hakukoneelle syöttämään hakulauseeseen, joka muodostuu hakusanoista.

Heroku	Pilvipalvelinympäristö, jossa voi kehittää ja säilyttää sovelluksia.
HTML	Hyper text Markup Language. Se tunnetaan erityisesti kielenä, jolla internetsivut on kirjoitettu.
Isomorfinen/universaali JavaScript	Isomorfisella/universaalilla JavaScriptilla tarkoitetaan JavaScript -koodia, joka voidaan ajaa sekä selaimessa, että palvelimella.
JavaScript	Heikosti tyypitetty kieli, joka suoritetaan selaimen tulkilla tai Node.js:ssä, joka on selaimen JS-moottori irrotettuna selainympäristöstä ja siirrettynä palvelimelle.
JSON (JavaScript Object Notation)	JSON on kevyt, helposti luettava formaatti tietojen järjestämiseen. Sitä käytetään pääasiassa selaimen ja palvelimen välisessä datan siirrossa.
Node.js	Node.js mahdollistaa palvelinohjelmoinnin JavaScriptilla. Se on selaimen JS-moottori irrotettuna ja siirrettynä palvelimelle.
Pay-Per-click marketing	Tapa käyttää hakukonemainontaa, jotta sivusto saa maksua vastaan enemmän klikkauksia ja kävijämäärä nousee.

ReactJS	JavaScript -kirjasto, jolla voidaan rakentaa SPA-sovelluksia.
Renderointi	Renderoinnilla opinnäytetyössä tarkoitetaan sivunmuodostumista. Sivu voi muodostua, joko palvelimella tai selaimessa.
Robots.txt	Sivuston juuressa oleva tekstitiedosto, joka estää indeksointirobotteja indeksoimasta ei haluttuja sivuja tai sivun osia.
SPA-sovellus (Single Page Application)	Sovellus, joka muodostaa näkymänsä selaimessa eikä käytön aikana vaadi sivun uudelleen latausta palvelimelta.
URL-osoite	URL-osoite ilmoittaa verkkosivuston tai tiedoston sijainnin internetissä. Hakukoneet yksilöivät sivut URL-osoitteiden mukaan.
Web-palvelin	Työssä palvelimesta puhuttaessa tarkoitetaan web-palvelinta. Se tarkoittaa tietokonetta tai ohjelmistoa, joka jakaa dokumentteja asiakasohjelmille ja koneille käyttäen HTTP-protokollaa.
404-virheilmoitus	404-virheilmoitus näytetään käyttäjälle, jollei sivuston URL-osoitteen mukaista sivua löydy.

1 Johdanto

Hakukoneet on alun perin rakennettu ymmärtämään vain palvelinpuolella renderoituja sivuja, jolloin selaimelle palautetaan valmis HTML-sivu, jonka hakukoneet osaavat lukea, tulkita sekä indeksoida tietokantoihinsa. Teknologiat kuitenkin kehittyvät ja JavaScript-pohjaiset SPA-sovellukset ovat keränneet suosiotaan perinteisten verkkosivujen rinnalla nopean käytettävyytensä ja kevyen rakenteensa ansiosta tarjoten käyttäjälle sulavan käyttökokemuksen.

Haittapuolena hakukoneiden näkökulmasta SPA-sovelluksissa on niiden vahvasti JavaScript-painoitteinen dynaaminen näkymien muodostuminen ja tämän vuoksi on kyseenalaistettu hakukoneiden kyky löytää, tulkita ja indeksoida SPA-sovelluksia. Tutkimuksen tavoitteena onkin selvittää onko SPA-sovellusten hakukoneoptimointi sellaisenaan mahdollista vai onko tarve turvautua palvelinpuolen renderointiin selainpuolen näkymien muodostamisen ohella. Tutkimuksessa tarkastellaan, mitä sovellukselta hakukoneeseen indeksoituakseen vaaditaan, sekä keinoja kuinka nämä asiat on mahdollista toteuttaa.

Työssä käytetyt testisovellukset on ladattavissa Githubista osoitteessa: https://github.com/hennasauranen/opinnaytetyo-2017

1.1 Toimeksiantaja, työn tavoitteet ja rajaukset

Työn toimeksiantaja on vuonna 1982 perustettu ohjelmistopalveluja tuottava yritys Solteq Oyj, jonka ydinosaamista ovat asiakaskohtaamiseen liittyvän liiketoiminnan teknologiset ratkaisut, -palvelut sekä liiketoiminnan tuki. Solteqin markkina-asema Suomessa johtavien kaupanalan toimijoiden keskuudessa on vahva. Lähivuosien tavoitteena on kasvaa strategisesti erityisesti pohjoismaissa.

Opinnäytetyön tavoiteena on tutkia kuinka SPA-sovellus (Single-page Application) voidaan toteuttaa teknisesti siten, että se on mahdollista hakukoneoptimoida ja että se saadaan indeksoitua hakukoneiden tietokantoihin. JavaScript-pohjaisten SPA-sovellusten yleistyessä on tärkeää, että hakukoneet eivät löydä ja indeksoi ainoastaan serveripuolen renderointiin luottavia perinteisiä verkkosivuja.

SPA-sovellusta mietittäessä hakukoneiden näkökulmasta on huomioitava se, että sovelluksen sisältö renderoidaan JavaScript-ohjelmointikielellä selaimessa. Perinteinen verkkosivu, joita hakukoneet on alunperin rekennettu ymmärtämään, renderoivat sisällön serverillä ja lähettävät sen valmiina selaimelle, jolloin hakukonetta vastassa on heti valmis sivusto eikä JavaScriptillä koottava kehys.

Sivun laadukas sisältö ja hakukoneen sisältöön käsiksi pääseminen vaikuttavat kriittisesti siihen kuinka sivusto sijoittuu hakutuloksessa. Opinnäytetyössä tutkitaan, kuinka hakukoneet pääsevät SPA-sovelluksen sisältöön käsiksi ja onnistuuko SPA-sovelluksen näkymien indeksointi. Hakukoneiden osalta tutkimus keskittyy lähinnä Googleen, mutta tutkimuksessa sivutaan myös muita hakukoneita. Tutkimuksen aikana rakennettiin ReactJS-kirjaston avulla kaksi SPA-sovellusta, joista ensimmäinen on normaali ReactJS-sovellus ja toinen käyttää avukseen isomorfista/universaalia JavaScriptia, jolloin koodi voidaan käsitellä selaimen lisäksi myös palvelimella.

1.2 Tutkimusmenetelmät

Opinnäytetyö toteutetaan kehittämistutkimuksena, sillä tutkimuksessa pyritään kehittämän Solteqin toimintaa SPA-sovelluksen hakukoneoptimoinnin osalta. Kehitystutkimus koostuu erilaisista kehitysmenetelmistä, jotka muodostuvat tukemaan tutkimuksen tarpeita yhdistellen kvalitatiivisia ja kvantitatiivisia tutkimusmenetelmiä. Se ei siis ole valmis tutkimusmenetelmä vaan muotoutuu aina omanlaisekseen tutkimuksen tarpeita varten. (Kananen, 2015, 33.)

Solteqilla SPA-sovellusten hakukoneoptimoitavuutta ei ole ennen tutkittu ja nyt siihen kaivataan muutosta. Tutkimuksessa pyritään rakentamaan hakukoneille luettavissa oleva SPA-sovellus. Tämän lisäksi pyritään ymmärtämään kuinka hyvin hakukoneet saavat jo tällä hetkellä luettua JavaScript-pohjaisten sovellusten sisältöä ja kuinka sovelluksen luettavuutta voidaan parantaa. Tutkimus käy läpi useita alan osaajien kirjoittamia artikkeleita luotettavista lähteistä ja siinä tehdään johtopäätökset näiden lähteiden perusteella. Tämän lisäksi tutkimuksen aikana kehitetyn ReactJS testisovelluksen indeksoitumista Googleen tarkastellaan hakubottien silmin Googlen

työkalujen avulla. Tutkimuksessa testataan myös isomorfisen/universaalin sovelluksen rakentamista.

1.3 Tutkimuskysymykset

Tutkimuskysymykset on johdettu tutkimuksen tavoitteista ja toimeksiantajan tarpeista. Seuraaviin tutkimuskysymyksiin pyritään saamaan vastaukset tutkimuksen aikana.

1. Millainen sivu on teknisesti hyvin hakukoneoptimoitavissa ja miksi hakukoneoptimointi kannattaa?
2. Miten SPA-sovellukset ja niiden hakukoneoptimointi eroavat perinteisistä sivuista?
3. Kuinka parantaa hakukoneiden indeksointimahdollisuuksia ReactJS-pohjaisissa sovelluksissa?

1.4 Opinnäytetyön tietoperusta ja rakenne

Tutkimusta varten koottu tietoperusta muodostuu suurelta osin alan osaajien tuottamista kirjoista ja artikkeleista sekä ohjelmointikielien ja palveluiden tukisivujen ohjeista.

Opinnäytetyö alkaa johdannolla, jossa kerrotaan työn motiivit ja tutkimusperiaatteet. Teoriaosassa, eli viitekehyksestä käydään läpi tutkimuksen kannalta oleelliset tekijät. Esimerkiksi hakukoneoptimointi, haussa sijoittumisen kriteerit sekä perinteiset- että SPA-sovellukset. Tutkimuksen käytännön toteutuksessa luodaan kaksi hakukoneystävällistä ReactJS-sovellusta: normaali ja isomorfinen. Normaalin sovelluksen indeksoitumista Googleen seurataan. Tulokset ja johtopäätökset -osiossa kirjataan vastaukset tutkimuskysymyksiin ja pohdinnassa ilmaistaan, miten tutkimus sujui ja mitä olisi voinut tutkia vielä lisää.

2 Hakukoneet ja hakukoneoptimointi

Hakukone on työkalu, jonka tehtävä on paikantaa verkkosivuja ja julkaisuja, jotka liittyvät käyttäjän hakukoneelle syöttämään hakulauseeseen, joka muodostuu hakusanoista. Hakukone palauttaa sille syötettyihin hakusanoihin täsmäävän

tulossivun, jota kutsutaan lyhenteellä SERP ja jonne on listattu hakulauseeseen liittyviä verkossa sijaitsevia dokumentteja. Google Search, Bing, and Yahoo! Search ovat yleisesti kolme kaikista merkittävintä hakukonetta. (Shenoy, Prabhu, 2016.) Maantieteellisellä sijainnilla on kuitenkin vahva vaikutus siihen mitkä hakukoneet koetaan merkittäviksi, sillä esimerkiksi venäjällä Yandex on yksi suosituimmista hakukoneista Googlen rinnalla. (Barysevich, 2016.)

Rand Fishkin, hakukoneoptimoinnin konsultointi yrityksen Moz:in perustaja kuvailee maailmanlaajuista verkkoa ison kaupungin metrokarttana, jossa jokainen ainutlaatuinen dokumentti, esimerkiksi verkkosivu, on oma pysäkkinsä. Hakukone tarvitsee näiden pysäkkien välille yhteyden, jonka avulla se pääsee jokaiselle pysäkille. Tämä yhteys muodostuu sivujen välisistä linkityksistä. Hakukoneiden automatisoidut botit, joita kutsutaan yleisemmin nimillä ”crawlers” tai ”spiders” yltävät linkkien avulla moniin biljooniin toisiinsa sidottuihin dokumentteihin verkossa, jolloin ne tulkitsevat niiden koodin ja tallettavat valitsemansa palat massiviseen tietokantaan, josta niitä voidaan hakea hakukoneella. (Fishkin, 2017.) Kun käyttäjä tekee hakukoneella haun, hakukone vastaa siihen käymällä läpi tietokannastaan löytyvät dokumentit ja palauttaa niistä kävijämäärältään suurimmat, jotka ovat relevantteja käyttäjän lähettämään hakulauseeseen verrattaessa. (Shenoy & Prabhu, 2016.)

Nykypäivänä sivuston sisäisten ja ulkoisten linkitysten laatu, tuore ja intuitiivinen sisältö sekä selkeä navigointi ovat päätekijötä määriteltäessä järjestystä, jossa hakukone tarjoaa käyttäjälle hakutulokset tulosivulla. Muita edellä mainittuun vaikuttavia tekijöitä ovat verkkosivun suosio, tekstisisällön laatu ja käyttäjäystävällisyys. Hakukoneoptimointi on siirtymässä koko ajan kohti käyttäjäkokemusmallia, jossa käyttäjäystävällisyydellä on suuri painoarvo haussa sijoittumiseen. Hakukoneoptimoinnin tavoite on parantaa verkkosivujen näkyvyyttä hakutuloksissa. Parhaimmillaan hakukoneoptimoinnilla saavutetaan paikka hakutulosten kärjessä, kun käyttäjä hakee sivustoon liittyvillä hakusanoilla tietoa hakukoneiden kautta. (Shenoy & Prabhu, 2016.)

2.1 Hakukoneoptimoinnin hyötyjä

Suuren yleisön saavuttaminen: Kun hakukone optimointi on tehty oikein ja sivusto nousee hakutulosten kärkisijoille, saadaan sivustolle aikaan lisää liikennettä. (Shenoy & Prabhu, 2016.)

Pysyvät tulokset: Kun sivun hakukoneoptimointia pidetään yllä, saadaan aikaan pysyviä tuloksia, jotka eivät ole riippuvaisia ulkopuolisista tekijöistä kuten esimerkiksi mainoksista. Toisin kuin esimerkiksi pay-per-click-lähestymistavassa, jossa hakutulosten paikka tippuu heti sillä hetkellä, kun lakataan maksamasta mainoksista. (Shenoy & Prabhu, 2016.)

Matalat kustannukset: Verkkosivun luonti ja ylläpito sekä hakukoneoptimoinnin asiantuntijan palkkaaminen tulee usein halvemmaksi kuin muut mainontakeinot, sillä pitkällä tähtäimellä ja pienellä ylläpidolla on mahdollista stabilisoida sivuston paikka hakutulosten joukossa. (Shenoy & Prabhu, 2016.)

Data ja sen analyysit: Google Analytics ja Google Search Console mahdollistavat datan keräämisen, joka auttaa ymmärtämään avainaspektit liittyen sivuston näkyvyyteen ja käyttäjien toimintaan sivustolla. On myös mahdollista nähdä konversioita sekä tutkia kuinka moni käyttäjä vain käy sivuilla ja lähtee pois ja kuinka moni puolestaan viipyy sivuilla pidempään. (Shenoy & Prabhu, 2016.)

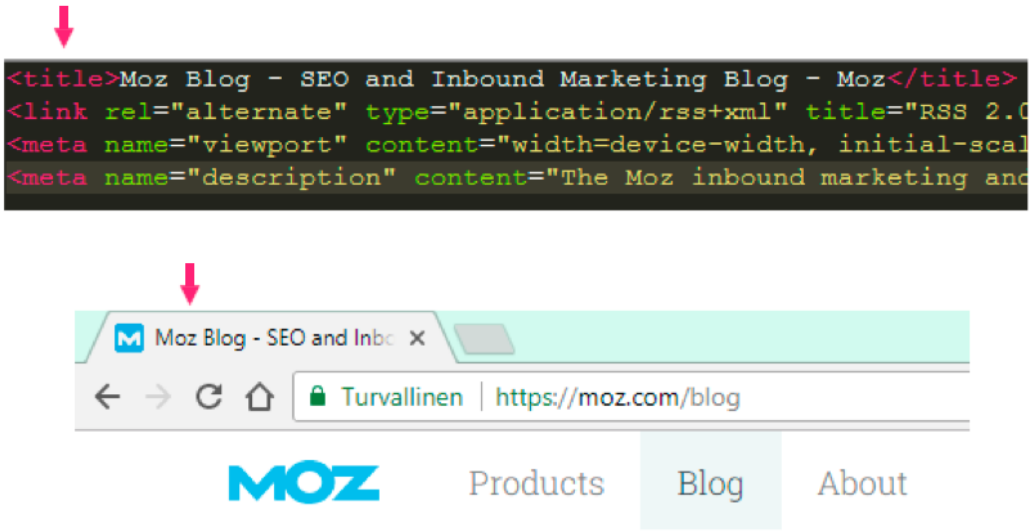
Kilpailussa mukana pysyminen: Hakukoneoptimoinnin erilaiset työkalut, esimerkiksi Google Analytics ja Google Search Console, antavat tarkastella tuloksia ja raportteja oman sivuston lisäksi myös kilpailijoiden sivuilta. Siksi on mahdollista vertailla tuloksia oman haussa sijoittautumisensa parantamiseksi. (Shenoy & Prabhu, 2016.)

Käytettävyys: Hakukoneoptimointi tukee parempaa käyttäjäkokemusta. Esimerkkinä verkkosivuston sisäiset linkit, jotka osoittavat tärkeisiin asiaan liittyviin sivuihin. Nämä linkitykset auttavat sekä hakukoneita indeksoimaan sivun, että käyttäjää navigoimaan sivuilla. Hakukoneoptimointi kannattaa rakentaa pitäen mielessä sivustolle suunnitellut käyttäjävaatimukset, sillä käyttökokemukseltaan hyvät sivustot sijoittuvat haussa paremmin. (Shenoy & Prabhu 2016.)

2.2 Haussa sijoittumiseen vaikuttavat tekijät

Verkkosivustoilla on useita tekijöitä, jotka ovat kontrolloitavissa ja joilla on tekemistä hakutuloksessa sijoittumisen kanssa. Näitä tekijöitä ovat mm. title -tagi, metatagit, mielenkiintoinen sisältö ja sen järkevä otsikointi sekä sisäiset että ulkoiset linkitykset. (Shenoy & Prabhu 2016.)

Sivun HTML -koodissa olevaan title -tagiin on hyvä tiivistää sivun sisältö selkeäksi ja sisältöä kuvaavaksi otsikoksi. Title -tagi merkitään dokumentin head-osaan ja se esiintyy sivuilla selainvälilehden otsikkona. (Ks. kuvio 1.) (Shenoy & Prabhu 2016.)

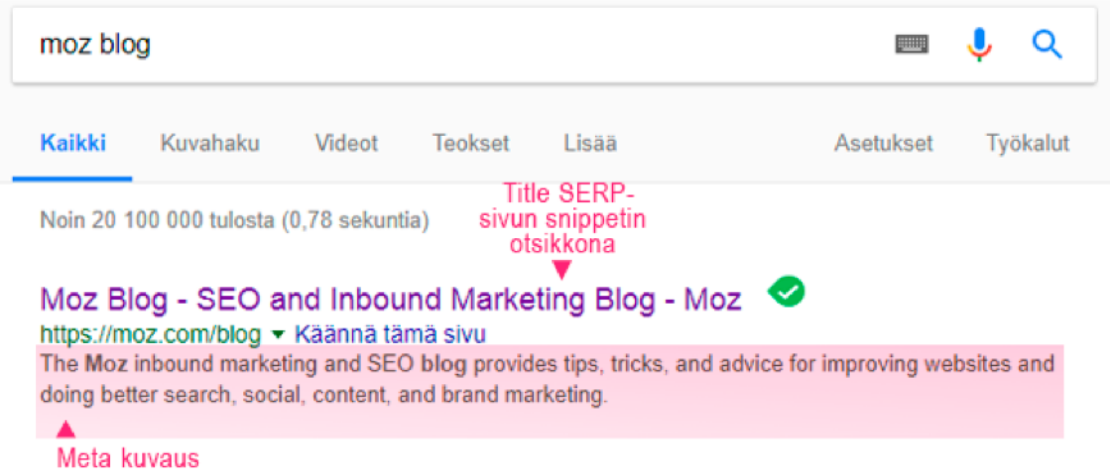


Kuvio 1. Title -tagin esiintyminen koodissa ja välilehden otsikkona

Hakukoneet esittävät usein sivun title -tagin myös otsikkolinkkinä kappaleelle, joka palautetaan hakutulos- eli SERP-sivulla. (Ks. kuvio 2.) On kuitenkin myös mahdollista, että hakukone muokkaa otsikkoa käyttäjän hakulauseeseen sopivammaksi. Title -tagissa tulisi käyttää lausetta, joka tiivistää sivun sisällön informatiiviseksi otsikoksi, jonka lukemalla saa käsityksen mitä aihetta sivusto käsittelee. Pituudeltaan title -tagin olisi hyvä olla noin 55-65 merkin mittainen. Title -tagin tulee olla ainutlaatuinen

sivuston jokaisella sivunäkymällä, jotta se saadaan parhaiten kuvaamaan juuri sen sivunäkymän sisältöä. (Shenoy & Prabhu, 2016.)

Google on varmistanut, että se ei enää pidä meta kuvauksia (meta description) ja -avainsanoja (meta keywords) suurina haussa sijoittumisen tekijöinä, mutta jotkut hakukoneet, kuten esimerkiksi Yahoo!, ottavat ne vielä huomioon (Edwards, 2014). Vaikka meta kuvaus ei suoranaisesti vaikutakaan Googlen haussa sijoittumiseen, se saatetaan joskus esittää SERP-sivulla, jolloin niiden avulla voidaan vaikuttaa käyttäjän haluun valita kyseinen sivu. Tämän vuoksi niitä ei kannata jättää huomioimatta. (Ks. kuvio 2.) Kuvausta kirjoittaessa on pidettävä mielessä, että se kirjoitetaan käyttäjälle eikä hakukoneelle. On myös mahdollista, että Google rakentaa itse sivuston tekstistä sopivan sisällön. (Shenoy & Prabhu, 2016.)



Kuvio 2. Title -tagin ja metatagin esiintyminen SERP-sivulla

Avainsana tagiin kirjataan sivun keskeisimmät hakusanat ja termit, jotka esiintyvät sivulla. Google antaa painoarvoa niille vain vähän, mutta jotkut hakukoneet, kuten Yahoo!, ottavat avainsanat vielä huomioon. (Shenoy & Prabhu, 2016.)

Meta robots -attribuutti kertoo hakukoneiden boteille pitääkö kyseinen sivu indeksoida vai ei. Meta robots -attribuutti merkitään sovelluksen head-osioon. (Meta tags that Google understands, 2017.)

Google yksilöi verkkosivun URL-osoitteen avulla. Sivujen indeksointi Googlen hakutietokantoihin tehdään siis URL-osoitteiden mukaan, jolloin jokaisella kantaan talletettavalla sivunäkymällä on oltava oma yksilöllinen URL-osoitteensa, jotta se voidaan tunnistaa muista sivuista (How Google Search Works, 2017). Tämän vuoksi Google suosittelee, että eri URL-osoitteiden takana oleva toistuva samanlainen sisältö on hyvä yhdistää linkittämällä ne yhteen Canonical-attribuutin avulla. Tämä kertoo hakukoneille tuplasivuista oikean, jonka hakukone sitten indeksoi. (Use canonical URLs, 2017.)

Googlen tukiforumilla on annettu esimerkki, jossa verkkokaupan tuotesivu löytyy usean eri URL:n takaa. Sivun järkevin URL, jonka mukaan indeksointi halutaan toteuttaa, on seuraava:

https://www.example.com/dresses/green/greendress.html

On kuitenkin mahdollista, että käyttäjä hakee tuotetta sivuilla olevan haun kautta, jolloin URL saattaisi olla seuraava:

https://www.example.com/products?category=dresses&color=green

Tässä tapauksessa hakukone löytää indeksoitavakseen saman sivun kahden URL-osoitteen takaa. Haluamme siis kertoa hakukoneelle, että se indeksoi vain oikean URL-osoitteen tietokantaansa. (Ks. kuvio 3.) (Use canonical URLs, 2017.)

```
name="description" content="Looking to level-up your content game? The free Beginner's Guide to
Content Marketing from Moz has you covered." />

  <link rel="canonical" href="https://moz.com/beginners-guide-to-content-marketing" />
```

Kuvio 3. Canonical atribuutin esiintyminen sivuston head osiossa

Laadukas, aiheeseen liittyvä ja kattava sisältö nousee isoon rooliin hakukoneoptimoinnissa. On tärkeää, että sivuston sisältö käsittelee sivun kannalta olennaisia termejä ja aiheita. Google suosittelee tekemään sivut käyttäjää eikä hakukonetta varten. Luonnoton ja liika avainsanojen toistaminen alentaa sivun hakukonenäkyvyyttä. (Anderson, 2016.) Avainsanoja ja niitä lähellä olevia variaatioita voidaan kuitenkin käyttää sivuilla kahdesti tai kolmesti loogisella tavalla. Sisältöä on hyvä päivittää

usein, sillä se lisää kävijöiden mielenkiintoa palata uudelleen sivuille. Vilkkaasti liikennöidyt sivut saavat suuremman painoarvon hakutuloksissa. (Shenoy & Prabhu, 2016.)

Sivuston sisällön selkeä ja informatiivinen otsikointi auttaa hakukonetta hahmottamaan sisällön rakenteen, jolloin sivu indeksoituu paremmin ja saavuttaa paremmin näkyvyyttä hakukoneiden hakutuloksissa. Ensimmäisen otsikkotagin (h1) sisällön, tulee olla relevantti sivuston aiheeseen nähden, sillä sen avulla hakukoneiden botit ja käyttäjät pysyvät kartalla mistä kyseinen sivu kertoo. On myös tärkeää, ettei otsikkotasojen yli hypätä. Esimerkiksi h1- seuraa h2- ja tätä h3-tagi. (Shenoy & Prabhu, 2016.)

Sivustorakenteen hyvä hierarkia ja linkitykset vaikuttavat näkyvyyteen positiivisesti. Sivuston sisäiset linkit ovat yksi tärkeimmistä tekijöistä hakukoneoptimoinnissa. Sivuston sisäiset linkit ovat linkkejä, jotka kulkevat sivuston sisällä yhdistäen sen sivunäkymät. On suositeltavaa, että sivuston sivunäkymät eivät ole kauempana kuin kolmen linkin päässä etusivusta. Tällöin sivut ovat helposti saatavilla sekä hakukoneita että hyvää käyttökokemusta ajatellen. Sivuston eri sivut kannattaa siis yhdistää linkkien avulla hakokonenäkyvyyden parantamiseksi. Sivujen URL-osoitteiden on hyvä olla käyttäjälle informatiiviset ja selkeät. (Fishkin, 2017.)

3 SPA-sovellus

Single Page Application eli SPA-sovellus toimii selaimessa eikä käytön aikana vaadi sivun uudelleen latausta palvelimelta. Kyseessä on sulavasti toimiva sovellus, joka lataa suurimmat resurssit, kuten esimerkiksi HTML:n, CSS:n ja Scriptit, palvelimelta vain kerran sovelluksen käytön aikana. Näiden resurssien avulla voidaan rakentaa kaikki sovelluksen näkymät ja toiminnot JavaScriptin avulla ilman uutta palvelimelle osoitettua pyyntöä ja sivuston uudelleenlatausta. Edellä mainittu myötävaikuttaa sivunäkymän vaihdosten nopeuteen ja siihen, että sisältö on usein vahvasti dynaamista. Tämän vuoksi SPA-sovellus tarjoaa käyttäjälle hyvän käyttökokemuksen. (Mäkäräinen, 2017.)

Palvelimella sijaitsevia tietoja käsitelläkseen SPA-sovellus ottaa palvelimeen tarvittaessa yhteyden asynkronisilla kutsuilla. Kun sisältö palvelimelta kutsutaan sovellukseen asynkronisesti, kutsun vastausta ei jäädä odottamaan. Sovellusnäkymä siis renderoidaan esitettäväksi selaimeen odottamatta kutsun vastauksena saapuvaa dataa. Kun kutsun vastausdata saapuu, se renderoidaan sivunäkymään paikoilleen jälkikäteen. (Mäkäräinen, 2017.)

SPA-sovelluksen kehittäminen on myös helpompaa, koska ei ole tarvetta palvelimella sivun renderoivalle koodille. SPA-tekniikalla on helpompaa myös tehdä mobiilisovelluksia, koska kehittäjä voi käyttää samaa palvelinpuolen koodia web-sovelluksille sekä mobiilisovelluksille. SPA voi tallentaa tiedot selaimen muistiin tehokkaasti, jolloin sovellusta voidaan käyttää myös offline-tilassa. (Mäkäräinen, 2017.)

3.1 Reititys ja hakukoneet

Perinteisesti sivustoissa liikutaan URL-osoitteiden kautta, jolloin jokaisella URL-osoitteella on oma dokumenttinsa, joka haetaan palvelimelta. Koska SPA-sovellukset toimivat yhdellä dokumentilla, jonne sivunäkymät muutetaan selaimessa JavaScriptillä, ne eivät vaadi URL-osoitteen päivittymistä tai uutta sivupäivitystä. (Emmit & Scott, 2016, 86–87.)

Moz:n artikkelin, ADK:n artikkelin sekä Googlen tukiforumin avulla voidaan päätellä, että muutkin hakukoneet Googlen lisäksi yksilöivät sivunäkymän URL-osoitteen avulla tietokantoihinsa. Sivut, jotka omistavat oman URL-osoitteen, on mahdollista indeksoida ja ne pystyvät sijoittumaan haussa. SPA-tekniikoin toteutetun sivuston sivunäkymät tulee siis ohjata omiin URL-osoitteisiinsa reitityksen avulla. (Fishkin, 2017.) (How Google Search Works, 2017) (Cauthorn, 2016.)

Hakukoneiden lisäksi myös käyttäjät olettavat pystyvänsä liikkumaan SPA-sivustolla tavallisen sivuston tavoin URL-osoitteiden kautta. Jotta tämä olisi mahdollista toteuttaa, tarvitaan reititykseä, joilla rakennetaan sisäinen navigaatio SPA-sovellukseen sekä määritellään URL-osoitteet näkymille. SPA-sovellusten reitityksellä luodaan sovelluksen näkymille omat URL-osoitteensa. SPA-sovelluksissa URL-osoite ei siis varsinaisesti määritä yhtä html-sivua kuten perinteisissä sivustoissa vaan sovelluksen tila ja tietyt

toiminnot tallennetaan URL-osoitteen alle. Reitityksen tehtäviin kuuluu myös 404 sivujen määrittäminen, jos osoitepalkkiin syötetylle linkille ei löydy reitittimestä vastinetta. Selainpään reititys hyödyntää yleisesti joko URL-osoitteen erillistä tunnistinta tai HTML5 History -rajapintaa. Molemmat metodit mahdollistavat ilman palvelinta toimivan navigaation hieman eri tavoin. (Emmit & Scott, 2016, 86–94.)

Yksi syy SPA-sovellusten huonoon maineeseen hakukonenäkyvyyden osalta on hakukonenäkyvyyteen negatiivisesti vaikuttava, jo ehkä hieman vanhahtava reititysmalli, joka on toteutettu käyttämällä URL-osoitteen erillistä tunnistinta. Tässä tavassa URL-osoitteessa käytetään ristikkomerkkiä (#) etuliitteenä ja sen jälkeistä osoitteen osaa tunnistimena, kun viitataan osioon dokumentissa. Esimerkiksi perinteisessä web-sovelluksessa yhteystiedot-sivu voisi löytyä osoitteesta *http://domain.fi/yhteystiedot/*, mutta SPA-sovelluksessa saman sivun URL olisi http://domain.fi/#yhteystiedot/. Reititys, joka käyttää erillisenä tunnistimena etuliite #-merkin jälkeistä osoitteen osaa, ei lähetä koskaan tätä tunnistinta palvelimelle osana http-pyyntöä. Tämän vuoksi hakukone, joka ei ymmärrä SPA-sovelluksia, todennäköisesti luulee *http://domain.fi/#galleria* ja *http://domain.fi/#yhteystiedot* olevan sama sivu. Google kuitenkin kiertää ongelman ja tarjoaa hakukoneoptimointia varten tuen URL-osoitteen erillistä tunnistinta käyttävälle reititykselle. Tätä tukea kutsutaan nimellä Hashbang (#!). (Strimpel & Najim, 2016, 10–11.)

Lähtökohtana Hasbang-tuessa on korvata #-merkki reitityksessä Hashbang-merkillä (#!), jolloin *http://domain.fi/#yhteystiedot* olisi *http://domain.fi/#!yhteystiedot*. Jotta #-merkin korvaaminen onnistuu, täytyy hakukonebotille kertoa HTML-dokumentin meta:ssa, että kyseessä on näkymä, jolle näin halutaan tehtävän. Sen jälkeen Googlen hakubotit tunnistavat ja voivat indeksoida näkymien sisällön. Tämä tapahtuu siten, että botti muuntaa näkymän URL-osoitteen sallituksi URL versioksi, jolloin *http://domain.fi/#!yhteystiedot* saisi muodon *http://domain.com/?query&_escaped_fragment=yhteystiedot*. Tässä vaiheessa palvelin, jolla SPA-sovellus on, ymmärtää tarjota botille snapshotin HTML näkymästä, joka esittää sivua *http://domain.fi/#!yhteystiedot*. (AJAX Crawling (Deprecated), 2009.) Hashbangiin on mahdollista perehtyä lisää Googlen Webmaster oppaissa. Google on kuitenkin julistanut tavan vanhentuneeksi vuonna 2015, mutta koska monet sivustot käyttävät tätä tapaa

edelleen Google kertoo pitävänsä sitä yllä vielä toistaiseksi. (Deprecating our ajax crawling scheme, 2015.)

HTML5 History -rajapintaa käyttävä menetelmä on uudempi ja parempi tapa hoitaa sivuston reititys. HTML5 History -rajapinnan käyttäminen reitityksessä on yleistynyt, sillä yhä useampi selain sekä SPA-kirjastot ovat alkaneet tukea sitä. Tässä toteutustavassa huomattavana etuna on siistimpi URL-osoite, jossa ristikkomerkkiä ei enää käytetä. (Strimpel & Najim ,2016, 10.)

HTML5 History -rajapinta tarjoaa muutamia metodeita selainhistorian manipulointiin. Esimerkiksi ”window.history.back()”-metodilla liikutaan taakse ja ”window.history.forward()”-metodilla eteen selaimen historiassa. Mielenkiintoisin selainpuolen reititykseen käyttävä metodi on kuitenkin ”window.history.pushState()”. (Tsonev, 2016)

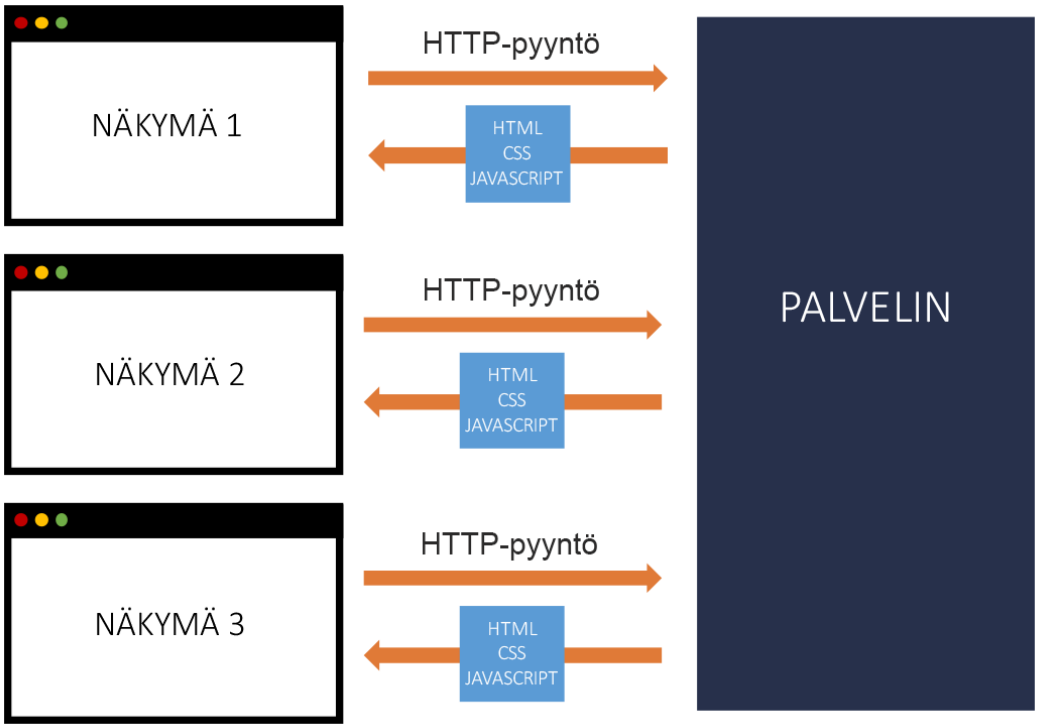
Selaushistorian tallennuksessa käytetään HTML5 History -rajapinnan pushState- ja replaceState-metodeita, joidenka avulla voidaan rakentaa selaimen historiaan ns. sisäänkäyntejä tiettyihin näkymiin. Metodin pushState ansiosta on mahdollista muuttaa selaimen URL-osoitetta ilman sivun uudelleenpäivitystä. URL-osoitteiden muuttumisen jälkeen voidaan ladata uusi siihen sidoksissa oleva sisältö. Mutta entä sitten, kun sivu päivitetään? Koska sivun osoite on luotu selaimessa JavaScriptillä, sivua päivitettäessä palvelin ottaa selaimen sijaan ohjat ja koska palvelin ei löydä JavaScriptillä muodostettuun URL-osoitteeseen vastaavaa dokumenttia se palauttaa käyttäjälle 404-virheilmoituksen. (Tsonev, 2016.) Tämä voidaan kuitenkin ratkaista esimerkiksi Webpackin tarjoamalla HistoryApiFallback -ratkaisulla (Webpack, 2017).

Hyvin toteutettu reititys eri näkymien välillä HTML5 History -rajapinnan avulla parantaa myös sovelluksen hakukoneystävällisyyttä (Deprecating our ajax crawling scheme, 2015).

3.2 SPA vs. perinteiset verkkosivut ja sovellukset

Ensimmäisistä HTML-sivuista lähtien palvelinpuolen renderointi on ollut keino, jolla saadaan sivu näkymään käyttäjälle selaimessa (Vega 2017). Eli toisin sanoen perinteiset verkkosivut ja sovellukset ovat lähes jatkuvasti yhteydessä palvelimeen. Käyttäjän

toimet aiheuttavat usein tarpeen palvelimelle lähetettävään HTTP-pyyntöön, jonka aikana data kulkee eri tasojen läpi pyyntöä koskevia tietoja keräten ja jonka jälkeen palvelin renderoi näitä tietoja käyttäen selaimelle palautettavan HTML-sivun. Tämä sivu lähetetään selaimelle, joka korvaa vanhan sivun kokonaan uudella. Selain siis näyttää serverillä valmiiksi rakennetun sivun käyttäjälle. (Ks. kuvio 4.) (Emmit & Scott 2016, 5-6.)

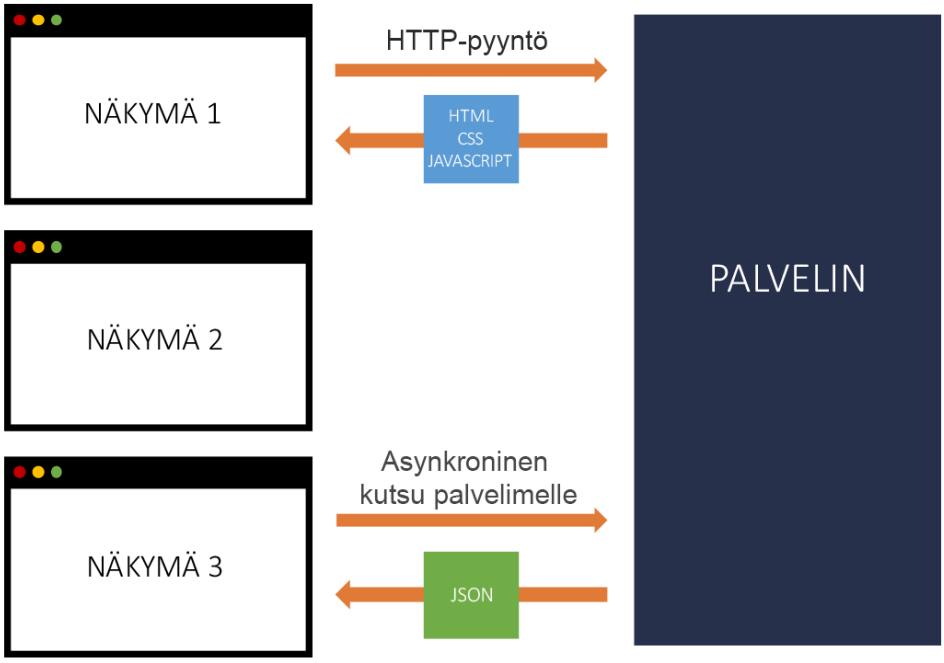


Kuvio 4. Näkymien muodostuminen palvelinrenderointia käytettäessä

Palvelinpuolen renderointi aikoinaan toimi hyvin, sillä sivut olivat silloin yksinkertaisempia ja kevyempiä, mutta se ei enää pidä paikkaansa, kun useat nykyaikaiset verkkosivut ovat lähempänä sovelluksia, jotka suorittavat monenlaisia tapahtumasarjoja (Vega, 2017). Edellä mainittu hidastaa palvelinpuolen renderointia huomattavasti ja nykyään serveripuolen renderoinnilla on käyttäjäystävällisyyteen negatiivinen vaikutus, koska sovelluksen käyttö keskeytyy aina näkymää vaihdettaessa. Sovelluksen toiminta on myös koko käytön ajan riippuvainen toimivasta internet yhteydestä. (Fink & Flatow, 2014, 6–8.) Hakukoneoptimointia ajateltaessa palvelinpuolen renderointi on kuitenkin toimiva ratkaisu, koska sivusto palautetaan selaimelle jo valmiissa muodos-

saan, jolloin hakubotit ymmärtävät sen, eikä mahdollisuutta liian aikaiseen indeksointiin ole, sillä sivua ei tässä tapauksessa rakenneta selaimessa vaan jo palvelimella (Strimpel & Najim, 2016, 6; Vega, 2017).

SPA-sovellusten toimintaperiaate eroaa paljolti perinteisistä sovelluksista. SPA-sovellusta ajettaessa jo ensimmäisen sivulatauksen jälkeen päästään selainpuolella käsiksi kaikkiin sovelluksen näkymiin, jolloin ainoa palvelimen ja selaimen välillä liikkuva datan siirto tapahtuu asynkronisilla kutsuilla, jotka palauttavat esimerkiksi JSON:ia. Nämä kutsut ovat nopeita ja kevyitä. Näkymien renderoinnin siirtyminen selain päähän tekee sivusta nopeamman ja sekä vapauttaa palvelinresursseja. (Emmit & Scott 2016, 6–13.)



Kuvio 5. Näkymien muodostuminen SPA-tekniikoin

SPA-sovellukset siis renderoidaan selaimessa JavaScriptin avulla. Käytännössä tämä toimii siten, että sivun ensimmäisellä latauskerralla serverin tehtävänä on ainoastaan ladata verkkosivun kehykset. Kaikki muu hoidetaan selainpuolella JavaScript kirjaston ja kustomoidun JavaScript -koodin avulla. Eli jos tässä tapauksessa ajetaan pelkkä HTML -koodi, mitään ei näy, koska mukaan tarvitaan myös sisällön tuova JavaScript-tiedosto, joka rakentaa HTML-sivun selaimessa. (Vega, 2017.)

Kuvitellaan tilanne, jossa käyttäjä on SPA-sovelluksen etusivunäkymässä, joka on pitkälti samanlainen kuin toinen saman sivuston näkymä, jonne käyttäjä päätyy navigointia klikkaamalla. Kun käyttäjä vaihtaa sivua selain ei tee uutta pyyntöä serverille koko sivusta vaan koostaa sen JavaScript kirjaston avulla selaimessa ja tarvittaessa hakee datan esimerkiksi JSON-muodossa palvelimelta tai muusta lähteestä, jonka jälkeen haettu data renderoidaan paikoilleen selaimessa sovellusnäkymään. Koko sivua ei siis päivitetä. Asynkroniset kutsut ja selainpuolen renderointi mahdollistavat paljon nopeamman tavan esittää sivusto käyttäjälle, kuin palvelinpuolen renderointi, sillä nyt koko sivun sijaan joudutaan renderoimaan vain pieni osa sivua. (Vega, 2017.)

4 SPA-sovellus ja hakukoneoptimointi

Strimpel ja Najim (2016) kertovat kirjassaan Building Isomorphic JavaScript Apps, että SPA-sovellukset eivät ole oletusarvoltaan hakukoneystävällisiä. Tämä tarkoittaa sitä, että on hyvin paljon mahdollista, etteivät käyttäjät löydä sovelluksen sivuja hakukoneiden kautta, ellei erityisiä toimenpiteitä SPA-sovellusten varalta ole tehty joko sovelluksen tai hakukoneen toimintaan.

Yksi SPA-sovellusten haasteista onkin sen näkymien dynaaminen muodostumisen ja JavaScript painottuneisuuden negatiivinen vaikutus hakukoneiden toimintaan. Sivustot saatetaan indeksoida väärin ja pahimmassa tapauksessa sivusto saattaa jäädä kokonaan indeksoinnin ulkopuolelle. (Strimpel & Najim, 2016.)

Google on lisännyt sivuja käsittelevään algoritmiinsa tuen JavaScriptin suorittamiselle SPA-sovellusten kasvavan määrän takia. Tämän takia Google pystyy ymmärtämään JavaScriptillä luotua sisältöä, joka mahdollistaa paremman hakukoneoptimoitavuuden SPA-sovelluksille. (Understanding web pages better, 2014.)

Googlen 23.5.2014 julkaisemassa ”Understandin web pages better”-artikkelissa käydään läpi, kuinka Google kykenee renderoimaan sisällöltään rikkaampia sivuja, jotka käyttävät toiminnoissaan pitkälti JavaScriptia. Google täsmentää vuoden 2015 lopussa, että heidän hakukoneensa näkevät sivustojen sisällön pitkälti samoin kuin modernit selaimet, eli käytännössä Google pystyy renderoimaan SPA-sovelluksen selaimen lailla. (Understanding web pages better, 2014.)

Vuoden 2015 lokakuussa julkaistussa Googlen ”Deprecating our ajax crawling scheme” -julkaisussa ilmoitetaan vanhentuneeksi 2009 julkaistu AJAX-crawling scheme eli tekniikka, jota aikaisemmin oli kehoitettu käyttämään SPA-sovellusten yhteydessä (Deprecating our ajax crawling scheme, 2015). Ajax crawling scheme-tekniikka perustui siihen, että sivusto käytti avukseen käyttöliittymätöntä selainta, jonka tehtävä oli toimittaa hakukoneelle tilannevedos (snapshot) sovelluksesta HTML-sivuna (AJAX Crawling (Deprecated), 2009).

Googlen 2014-vuoden julkaisussa, jossa todetaan Google ymmärtävän JavaScriptia, mainitaan kuitenkin, että on mahdollista, että joskus sivuston renderointi epäonnistuu. Google mainitsi myös muutaman tavan, kuinka ehkäistä ongelmia ennalta. Googlen on päästävä käsiksi JavaScript ja CSS tiedostoihin. Jos tämä on estetty esimerkiksi robots.txt-tiedostossa, Googlen hakubotit eivät voi tulkita sivua samoin kuin käyttäjä ja vaarana on, että osa tiedoista tai sivuista jää indeksoimatta. Kannattaa myös varmistaa, että verkkosivujen palvelin kykenee käsittelemään Googlen pyynnön päästä käsiksi sivuihin. On myös mahdollista, että sivuston JavaScript on joskus liian monimutkaista Googlen tulkittavaksi, jolloin sivuston täydellinen renderointi ei onnistu. Google kehottaa, että on hyvä idea varmistaa, että sivuston perustoiminnallisuudet toimivat jokaisessa käyttötapauksessa ja etenkin silloin, jos sivuston JavaScriptia ei voidakaan tulkita täysin. Tämä kohentaa sekä käyttäjäkokemusta että hakukoneiden mahdollisuuksia onnistuneesti indeksoida sivu. (Understanding web pages better, 2014.) Googlen Fetch as Google -työkalulla on mahdollista tutkia, kuinka Google näkee nettisivujen sisällön. Sen avulla voidaan simuloida Google botin pääsy sivuille ja kuinka Google saa sivun renderoitua (Other content considerations, 2017).

Vaikka selainpuolen renderointi on kohentanut monien sivujen käyttömukavuutta ja nopeutta huomattavasti, Google näyttäisi tällä hetkellä olevan ainoa hakukone, joka tulkitsee SPA-sovelluksia sujuvasti. (Connors, 2016; Strimpel & Najim 2016.)

5 Sivustokartta

Sivustokartta on tiedosto, jonne voidaan listata sivuston yksittäisiä sivuja kertomaan Googlelle ja muille hakukoneille sivuston rakenteesta. Hakukoneiden botit kuten esimerkiksi Google bot lukee sivustokartan ja osaa tämän jälkeen tulkita sivuston paremmin. Sivustokartta voi tarjota lisäksi arvokasta metadataa sivustokartassa listattuihin sivuihin liittyen. Metadatalla tarkoitetaan tietoa sivuun liittyen, kuten esimerkiksi milloin sivu on viimeksi päivitetty, kuinka usein sivua muutetaan, sivun tärkeys verrattuna sivuston muihin sivuihin. (Learn about sitemaps, 2017.)

Google tukee muutamia eri sivustokarttatiedostomuotoja mutta suosituin on XML-kartta. Oli kyseessä mikä tiedosto tahansa sivustokartta voi olla maksimissaan kooltaan 50MB ja sisältää 50,000 URL-osoitetta. Jos on tarve kookkaammalle sivustokartalle, on mahdollista luoda sivustokartta index-tiedosto, joka yhdistää useita yksittäisiä sivustokarttoja. Tämä index-tiedosto annetaan Googlelle, joka pääsee sen kautta käsiksi kaikkiin sivustokarttoihin, joihin index-tiedosto osoittaa. (Build and submit a sitemap, 2017.)

5.1 XML-sivustokartan tiedostomuotona

Google tukee sivustokarttojen standardiprotokollaa. Tämän lisäksi se tukee XML-lisäosia videolle, kuville ja uusille resursseille. Näitä lisäosia voidaan käyttää, kun halutaan indeksoida video-, kuva- tai muita vaikeasti indeksoitavissa olevia tiedostoja. Alla Googlen antaa esimerkin sivuillaan XML-sivustokartasta, joka sisältää yhden URL-osoitteella määritellyn sijainnin. (Ks. Kuvio 6) (Build and submit a sitemap, 2017.)

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/foo.html</loc>
  </url>
</urlset>
```

Kuvio 6. Esimerkki yhden osoitteen XML-sivustokartasta

Googlen antaa sivuillaan myös esimerkin monimutkaisemmasta sivustokarttarakenteesta, joka sisältää URL-osoitteen lisäksi kuva- ja videotiedostojen informaatiot (Ks. Kuvio 7) (Build and submit a sitemap, 2017).

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
  xmlns:image="http://www.google.com/schemas/sitemap-image/1.1"
  xmlns:video="http://www.google.com/schemas/sitemap-video/1.1">
  <url>
    <loc>http://www.example.com/foo.html</loc>
    <image:image>
       <image:loc>http://example.com/image.jpg</image:loc>
       <image:caption>Dogs playing poker</image:caption>
    </image:image>
    <video:video>
      <video:content_loc>
        http://www.example.com/video123.flv
      </video:content_loc>
      <video:player_loc allow_embed="yes" autoplay="ap=1">
        http://www.example.com/videoplayer.swf?video=123
      </video:player_loc>
      <video:thumbnail_loc>
        http://www.example.com/thumbs/123.jpg
      </video:thumbnail_loc>
      <video:title>Grilling steaks for summer</video:title>
      <video:description>
        Cook the perfect steak every time.
      </video:description>
    </video:video>
  </url>
</urlset>
```

Kuvio 7. Esimerkki monimutkaisemmasta sivustokarttarakenteesta.

5.2 Yleiset ohjeet sivustokartan luomiseen

1. Käytä olemassa olevia täysin päteviä URL-osoitteita. Google tutkii sivuston URL-osoitteen juuri sellaisina, kun ne on listattu sivustokartassa. Esimerkiksi, jos sivustosi on osoitteessa *http://www.esimerkki.fi/* ei pidä ilmoittaa URL osoitteeksi *http://esimerkki.fi/* (ilman www osaa) tai *./seuraavasivu.html* (relatiivinen URL) (Build and submit a sitemap, 2017.)

2. Osoita URL-osoitteen eri kielikäännökset Googlelle indeksointia varten listaamalla canonical URL-osoitteet eri kielelle sivustokarttaan käyttäen hreflang annotaatiota (Build and submit a sitemap, 2017).

3. Sivustokartan tulee olla UTF-8 enkoodattu (Build and submit a sitemap, 2017).

4. Sivustokartta tulee jakaa useammaksi sivustokartaksi, jos sen URL-osoitteiden määrä ylittää 50,000 tai sivustokartan koko 50MB rajan (Build and submit a sitemap, 2017).

5. Käytä yksittäisten sivustokarttojen sijaan sivustokarttojen index-tiedostoa, jonne on listattu kaikki sivuston sivustokartat (Build and submit a sitemap, 2017).

6. Kerro Googlelle, jos sivustollesi on pääsy sekä www-domainversiosta ja ei-www-domainversiosta (Build and submit a sitemap, 2017).

5.3 Sivustokartan lisääminen Googleen ja sivustokarttojen hallinta

Sivustokartan lisäämiseen on kaksi keinoa. Sen voi tehdä manuaalisesti käyttäen Googlen Search Consolen sivustokarttatyökalua tai lisäämällä sivuston robots.txt määrityksen sivustokartan sijainnista. Robots.txt tiedostoon sivustokartan lisääminen onnistuu seuraavan rivin lisäämisellä: *Sitemap: http://example.com/sitemap_location.xml*. (Build and submit a sitemap, 2017.)

Jos käytössä on useita sivustokarttoja, kannattaa niihin ohjata index-tiedoston kautta. Index-tiedoston ansiosta nämä sivustokartat voidaan lisätä Googlelle yhdellä kerralla. XML index-tiedosto on hyvin samankaltainen kuin perus XML-sivustokartta. Index-tiedosto käyttää alla mainittuja tageja. (Simplify multiple sitemap management, 2017.)

- **Sitemapindex** – Parent tagi, joka ympäröi tiedoston.
- **Sitemap** – Parent tagi jokaiselle sivustokartassa listatulle sivustokartalle. (sitemapindex:n lapsi)
- **Loc** – Sivustokartan sijainti (sitemap tagin lapsi)
- **Lastmod** – viimeisin päivä, jolloin sivustokarttaa on muokattu (ei pakollinen)

Googlen sivuilla on esimerkki sivustokarttojen index-tiedostosta. (Ks. Kuvio 8)

```
<?xml version="1.0" encoding="UTF-8"?>
   <sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <sitemap>
      <loc>http://www.example.com/sitemap1.xml.gz</loc>
      <lastmod>2004-10-01T18:23:17+00:00</lastmod>
   </sitemap>
   <sitemap>
      <loc>http://www.example.com/sitemap2.xml.gz</loc>
      <lastmod>2005-01-01</lastmod>
   </sitemap>
   </sitemapindex>
```

Kuvio 8 Esimerkki index-sivustokartasta

6 Hakukoneystävällisen ReactJS-sovelluksen luominen

Tässä luvussa keskitytään siihen, kuinka osana opinnäytetyötä rakennettiin ReactJS sovellus ja kuinka sen näkymät saatiin näkyviin Googlelle. Luvussa kuvataan ensin testisovelluksessa käytettyjä tekniikoita, kuten millainen kirjasto ReactJS on ja mitkä sen pääominaisuudet ovat. Tämän jälkeen kuvataan testisovelluksen rakenne ja siinä käytetyt tekniikat. Viimeisimpänä kuvataan, kuinka sovelluksen indeksoituminen Googleen onnistui ja kerrotaan muista testissä havaituista asioista.

6.1 Node.js ja npm

JavaScript syntyi toimimaan selaimessa, mutta Node.js mahdollistaa JavaScript-ohjelmien suorittamisen paikallisesti sekä palvelimella. Node.js:n avulla kehittäjä voi tehdä normaalia palvelinpuolenohjelmointia JavaScriptilla. (Gakenheimer, 2015.)

Yhdessä npm:n, eli Node.js:n pakettimanagerin, kanssa Node.js:sta on tullut korvaamaton JavaScript-sovellusten paikalliselle kehittämiselle, jolloin kehittäjä voi luoda komentosarjoja käynnissä oleviin tehtäviin, kuten kopiointiin, tiedostojen siirtämiseen tai paikallisen kehityspalvelimen käynnistämiseen. Npm:n avulla voidaan myös automaattisesti ladata riippuvuuksia. (Gakenheimer, 2015.)

6.2 ReactJS

ReactJS on Javascript -kirjasto, jota voidaan käyttää selaimessa, palvelimessa ja mobiililaitteissa ja joka on kehitetty Facebookin insinöörien toimesta Facebookin käyttöön korvaamaan aiemmin käytössä ollutta perinteistä asiakaspuolen Model View Controller -mallia hallitsemaan sivuston pienien erillisten osien tiloja. Lisäksi ReactJS:lla pyrittiin parantamaan DOM:in renderointinopeutta ja -tehokkuutta. (Gackenheimer, 2015.)

SPA-sovellukset hakevat jatkuvasti uutta tietoa ja muuntavat DOM:n osia käyttäjän vuorovaikutuksesta. DOM eli Document Object Model on, jokaisen HTML-sivun ydin. Se on rajapinta HTML- ja XML-dokumenteille ja esittää sivun siten, että ohjelmistoilla on mahdollista muokata dokumentin rakennetta, tyylejä ja sisältöä. DOMin avulla ohjelmointikielet voivat siis luoda, muokata ja poistaa jo olemassa olevaa sivun sisältöä. (What is DOM?, 2017.)

Sovellusten tiedonhakuun tarvittavien rajapintojen kasvaessa ja muuttuessa monimutkaisemmiksi, on yhä vaikeampaa sekä hitaampaa tarkastella sovelluksen eri näkymiä ja tehdä muutoksia suoraan DOM:iin. Useat JavaScript -kehykset käyttävät yhä Data binding -tekniikkaa, joka sitoo datan lähteen ja esittäjän yhteen synkronoiden ne. Tällä lähestymistavalla on kuitenkin negatiivisia vaikutuksia ylläpidettävyyteen, skaalautuvuuteen ja suorituskykyyn. (De Sousa Antonio 2015.)

ReactJS puolestaan käyttää näkymämuutoksiaan varten omaa virtuaalista DOMiaan. Tämä kevyt DOM sijaitsee ReactJS:n muistissa ja sen käyttäminen on nopeampaa ja tehokkaampaan kuin oikean DOMin jatkuva muuttaminen. Kun sovelluksen tila muuttuu esimerkiksi käyttäjän vuorovaikutuksen tai tiedonkeruun tuloksena, muutokset sijoitetaan ensin virtuaaliseen DOMiin, jonka jälkeen ReactJS vertaa siihen sovelluksen tämänhetkistä tilaa ja toteuttaa mahdollisimman vähän oikean DOM:n muutoksia saavuttaakseen halutun lopputuloksen. Tämä tekee ReactJS:sta hyvin nopean, sillä ylimääräisiä DOM muutoksia ei toteuteta. (De Sousa Antonio 2015.)

ReactJS-sovellukset koostuvat komponenteista, jotka ovat itsenäisiä sovelluksen rakennuspalikoita. Sovelluksen komponentit on hyvä pitää yksinkertaisina, jotta niitä

on mahdollista yhdistellä helposti ja rakentaa niiden avulla isoja kokonaisuuksia. Komponenteilla muodostetaan siis sovelluksen näkymät. Nämä tyypillisesti luodaan alla olevan esimerkin tapaan. (Gackenheimer, 2015.)

```
class MyClass extends React.Component {

    render() {
        return <div>hello world</div>;
    }
}
```

JSX on HTML:n kaltainen syntaksi, jota React käyttää. JSX periytyy ECMAScriptista. Syntaksi preprosessoidaan käyttäen esimerkiksi Babelia, joka muuntaa JSX:n standardin mukaiseksi JavaScriptiksi, jonka JavaScript -tulkki kykene suorittamaan. (Gackenheimer, 2015.) JSX:n avulla voidaan kirjoittaa HTML:n kaltaisia rakenteita samaan tiedostoon, jossa on JavaScript -koodia. Babelin tehtävä on muuttaa JSX JavaScriptiksi. (Gackenheimer, 2015.)

JSX:ää ei ole pakko käyttää ReactiJS:ssa, mutta sen käyttö on suositeltavaa, sillä se tekee sovelluksen rakentamisesta sulavampaa. JXS kääntää tagit ReactJS:n elementeiksi, kuten alla olevassa esimerkissä. (Gackenheimer, 2015.)

JSX-versio

```
React.render(

  <div>
    <h1>Header</h1>
  </div>
);
```

JSX-versio käännetään tähän muotoon

```
React.render(
  React.createElement('div', null,
    React.createElement('h1', null, 'Header')
  );
);
```

Koska JavaScriptia kehitetään jatkuvasti, se muuttuu vuosien varrella. Viime aikoina yhteisö sopi parannuksista kielelle ja tästä versiosta muodostui ES6. Jotkut uusimmista selaimista ovat jo toteuttaneet tällaisia ominaisuuksia ja kykenevät suorittamaan ES6 versiota JavaScriptista. Vanhempia selaimia varten kieli kannattaa kuitenkin kääntää esimerkiksi Babelin avulla tavalliseksi JavaScriptiksi. (Gakenheimer, 2015.)

ReactJS-kirjastoa varten on olemassa useita lisäosia, jotka auttavat ja tekevät kehityksestä mielekkäämpää. Nämä lisäosat eivät sisälly ReactJS:n omaan kirjastoon vaan niitä varten on omat kirjastot, jotka voidaan asentaa projektikohtaisesti. (Gackenheimer, 2015.)

Yksi ReactJS:n eduista onkin se, että se ei ole raskas, sillä pääkehyksen lisäksi kaikki, muu ladataan ReactJS-sovellukseen lisäosakirjastojen avulla. Näitä kirjastoja on olemassa todella monia ja niitä kehitetään jatkuvasti lisää. (Gackenheimer, 2015.)

6.3 Testisovellus ja sen rakenne

Sovelluksen avulla on tarkoitus hahmottaa, kuinka Google näkee ReactJS-pohjaisen sovelluksen sisällön ja indeksoi sen tietokantaansa. Sovelluksen näkymät muodostuvat kolmesta eri näkymäkomponentista, joista yksi on komponentti, jolla on neljä eri sisältömahdollisuutta (kissa-, koira-, tiikeri- ja karhusivu) sen perusteella, mitkä tiedot sinne haetaan asynkronisesti sovelluksen ulkopuolella sijaitsevasta JSON-tiedostosta. (Ks. Liite 1 & 2) Jokaisella näistä sovelluksen tiloista on oma URL-osoitteensa ja React Helmet:n avulla vaihdetut näkymäkohtaiset tiedot sivun head-osiossa. React Helmet on komponentti, jonka avulla dokumentin head-osion sisältöä voidaan muuttaa. Testisovelluksessa mahdollistetaan title -tagien ja metatagien muutokset React Helmet:n avulla.

Yhteensä näkymiä, joilla on oma URL-osoite, on siis kuusi kappaletta:

Etusivu: https://react-seo-sauranen.herokuapp.com/
Eläinvalikko: https://react-seo-sauranen.herokuapp.com/elaimet
Kissasivu: https://react-seo-sauranen.herokuapp.com/elaimet/kissa

Koirasivu: https://react-seo-sauranen.herokuapp.com/elaimet/koira

Tiikerisivu: https://react-seo-sauranen.herokuapp.com/elaimet/tiikeri

Karhusivu: https://react-seo-sauranen.herokuapp.com/elaimet/karhu

Sovelluksen pohjana käytetään www.reactboilerplate.com sivuston sovelluspohjaa, josta muokattiin opinnäytetyötä varten sopiva versio. Esimerkiksi sovelluspohjassa käytetty Redux ja sen lisäosat poistettiin, sillä testisivusto haluttiin pitää mahdollisimman yksinkertaisena.

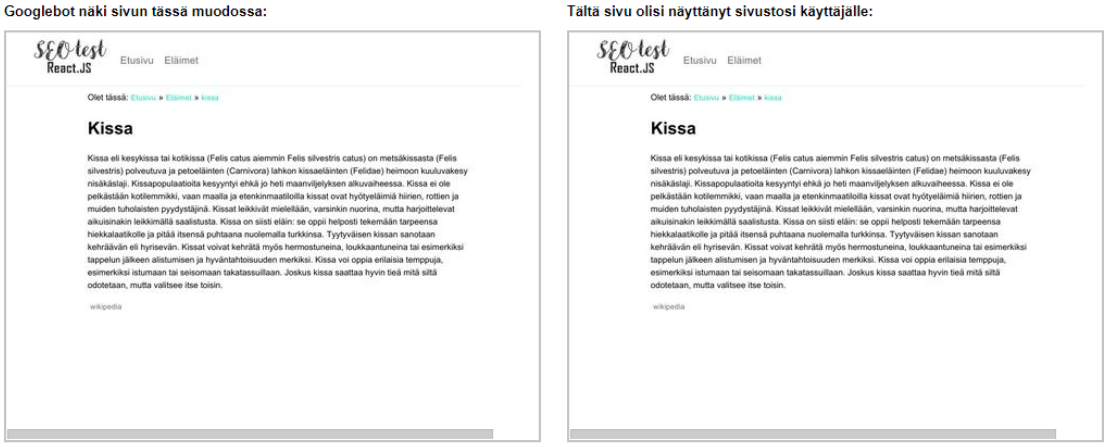
Sovelluspohjan container/component-arkkitehtuuri yksinkertaistettiin pelkäksi component-arkkitehtuuriksi. Käytännössä tämä tarkoittaa sitä, että toisin kuin sovelluspohjassa, testisovelluksessa ei ole container-komponentteja, jotka hoitavat datan hakemisen ja tämän jälkeen näkymästä huolehtivan alikomponentin renderoinnin. Näkymäkohtaisen datan hakeminen ja näkymien luonti tapahtuvat siis samalla komponentilla.

Valmis sovellus sijoitetiin Heroku:n pilviympäristöön. Ympäristön Ilmaisen version haasteeksi koitui se, että palvelimen vastausaika oli hyvin hidas, sillä jollei sivustolla käynyt, sen palvelin vaipui uneen ja käynnistyi aina sivuilla käytäessä uudelleen. Tämän vuoksi Google ei saanut ensin indeksoitua kaikkia sivuja. Ongelma ratkaistiin Kaffeine -sivuston avulla, joka pingaa Heroku-sovelluksen 30 minuutin välein, joten se ei koskaan mene nukkumaan.

6.4 Testisovelluksen indeksoituminen Googleen

Fetch as Google on työkalu, jonka avuilla voidaan tarkistaa, miten Googlebot näkee sinne syötetyn URL-osoitteen sivun sisällön. Fetch as Google -työkalulle annetaan sivun URL-osoite, jonka jälkeen Google näyttää vierekkäin, miten Googlebot ja oikea käyttäjä näkevät sivun.

Alla kuva Fetch as Google -vertailun tuloksesta. Vasemmalla Googlebotin näkemä sivu ja oikealla käyttäjän näkemä sivu. (Ks. Kuvio 9)



Kuvio 9. Kuva Fetch as Google vertailun tuloksesta

Edesauttamaan sivuston indeksoitumista on hyvä lisätä Googlelle sivuston sivustokartta. Googlen Search Consolessa on omat työkalunsa sivustokartan lisäämisen ja sen käsittelyn seuraamiseen. Googlen Search Console ilmoittaa, jos jotain sivustokartan käsittelyssä on mennyt pieleen ja mistä se mahdollisesti voisi johtua.

Testin aikana saatiin selville, että ReactJS-pohjainen sovellus on mahdollista indeksoida Googleen. (Ks. Liite 5) Sovellusta rakentaessa täytyy ottaa huomioon tietyt asiat, mitä Google pitää indeksoitumisen kannalta tärkeänä. Kuten jo aikaisemmissa luvuissa mainittiin, Google indeksoi sivuston sivut URL-osoitteiden perusteella. ReactJS-sovelluksen kaikilla näkymillä, jotka halutaan indeksoida sivuina, tulee olla oma URL-osoitteensa.

HTML5 History -rajapintaa hyväksikäyttävät URL-osoitteet ovat selkeämpiä käyttäjälle sekä hakukoneelle, joten näillä toteutetut sivut tulevat sijoittumaan paremmin haussa. On kuitenkin mahdollista indeksoida myös #-merkkiä käyttävät URL-osoitteet Googlen Hashbang-ratkaisulla, mutta se ei ole kannatettavaa, sillä tapa on vanhentunut. HTML5 History -rajapinnan reitityksille on tehtävä ratkaisu sivun päivittämisen varalle, ettei palvelin ohjaa URL-osoitteesta 404-sivulle, kun selaimessa luotua URL-osoitetta ei löydy palvelimelta.

Lisäksi tärkeää sivun indeksoitumisen kannalta on, että näkymillä on omat näkymäkohtaiset arvonsa dokumentin head-osiossa, kuten aiemmissa luvuissa ollaan mainittu. Tämä testisovelluksessa hoidetaan käyttämällä React helmet -kirjastoa, joka muuntaa dokumentin head-osiota JavaScriptin avulla. (Ks. Kuvio 10)

```
export default class HomePage extends React.PureComponent { // eslint-disable-line react/prefer-stateless-
  render() {
    return (
      <div className="container">
        <Helmet>
          <meta charSet="utf-8" />
          <title>Opinnäytetyötuotos</title>
          <meta name="description" content="Työ on tuotettu osana tietojenkäsittelyn koulutusohjelmaa." />
        </Helmet>
        <div className="breadcrumb">
          Olet tässä:
          <Link to={'/'}>Etusivu</Link>
        </div>
        <h1>Opinnäytetyötuotos</h1>
        <p>Työ on tuotettu osana Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmaa. Kysees
      </div>
    );
  }
}
```

Kuvio 10. Helmet -tagin esiintyminen koodissa

Kokeen aikana huomattiin, että vaikka dokumentin head-osioon muutettiin JavaScriptillä oikeat tiedot (kaikille näkymille omat title -tagit ja metatagit yms.) ne indeksoituivat Googleen oikein. Sivustokartan kautta indeksoituneiden sivujen kohdalla kävi niin, että ne näkyivät aluksi haussa otsikolla, joka on merkitty HTML-kehykseen ennen kuin se muutetaan JavaScriptillä React helmet –lisäosalla oikeaksi. Google korjasi väärin indeksoituneiden sivujen otsikot itsestään parin päivän kuluttua. Kaikkiin näiden näkymien head-osioihin sisältö päivitettiin asynkronisesti ja osaan se haettiin sovelluksen ulkopuolelta eri palvelimella sijaitsevalta JSON-tiedostosta. (Ks. Kuvio 11 & 12) Kuviossa 11 ilmentyvän koodipätkän koko komponentti on liitteessä 1. Liite 2 sisältää kuvan JSON-tiedostosta.

```
<Helmet>
  <meta charSet="utf-8"/>
  <title>{animal.name}</title>
  <meta name="description" content={animal.kuvaus} />
</Helmet>
```

Kuvio 11. Titlen ja meta kuvauksen muuttaminen asynkronisilla arvoilla

Kuvio 12. Sivun esiintyminen Googlen hakutulossivulla

Apuna indeksoitumiseen käytettiin sivustokarttaa sekä Fetch as Googlen kautta lähetettyjä yksittäisiä pyyntöjä indeksoida tietyn URL-osoitteen takainen näkymä. Fetch as Googlen kautta lähetetyt yksittäiset pyynnöt indeksoituivat lähes heti pyynnön lähettämisen jälkeen. Sivuista Fetch as Googlen avulla indeksoitiin seuraavat sivut:

- https://react-seo-sauranen.herokuapp.com/elaimet
- https://react-seo-sauranen.herokuapp.com/elaimet/kissa
- https://react-seo-sauranen.herokuapp.com/elaimet/karhu

Osa sivuista indeksoitiin Fetch as Google -työkalulla ja loput sivustokartan avulla. Sivustokartan avulla indeksoitiin seuraavat sivut:

- https://react-seo-sauranen.herokuapp.com/elaimet/koira
- https://react-seo-sauranen.herokuapp.com/elaimet/tiikeri

Sivustokartan käsittelyssä aikaa kului pari viikkoa ennen kuin se sai muuta kautta indeksoimattomat sivut käsiteltyä. Sivustot indeksoituivat tätä kautta ensin väärin, ilman JavaScriptilla tehtyjä muutoksia, mutta muutaman päivän odottelun jälkeen Google korjasi asian. Usean sivun indeksoiminen vain sivustokartan avulla tulee olemaan oletettavasti hidas prosessi. On kuitenkin mahdollista, että tämä johtui siitä, että Heroku:n ilmaisversion vastausaika oli edelleen liian pitkä, mutta tästä ei tutki-

muksessa tullut varmuutta. Syynä indeksoinnin hitauteen voi olla myös sivuston vähäinen suosio, jolloin Google katsoo sivun indeksoimisen turhaksi. Muilla hakukoneilla kuin Googlella indeksoitumista ei edes testattu, sillä tutkimuksen aikana on tullut selväksi, että ne eivät kykene suorittamaan JavaScriptia.

7 Isomorfinen/universaali web-sovellus

Jos tavoitteena on löytää sivusto muidenkin hakukoneiden kuin Googlen avulla, ainoa ratkaisu on renderoida sovelluksen sisältö palvelimella. Tähän on keksitty ratkaisu, joka ei ole yhtä hidas, kun perinteinen palvelinpuolen renderointi.

Isomorfinen, tai toisin sanottuna universaali, JavaScript-sovellus on sovellus, joka jakaa saman JavaScript -koodin selaimelle sekä palvelimelle. ”Isomorfinen”-sanan synonyymi samanmuotoinen selittää jo itsessään hieman sovelluksen toimintamallia, sillä sovellus pysyy samana sen ajoympäristöstä riippumatta. Isomorfinen sovellus voidaan siis renderoida palvelimella tai selaimella sovelluksen koodin pysyessä samana. Isomorfisessa sovelluksessa hakukoneoptimoitavuuden sekä käyttökokemuksen kannalta kannattaa käyttää hakukoneystävällisiä URL-osoitteita HTML5 History -rajapinnan avulla. (Strimpel & Najim, 2016, 4.)

Selainpuolen reitityksen tai muun toiminnallisuuden mahdollisesti pettäessä, esimerkiksi vaikka HTML5 History -rajapintaa tukemattomien selainten tai JavaScriptia hallitsemattomien hakukoneiden takia, isomorfinen sovellus perääntyy palvelinpuolelle ja kykenee palvelimella etsimään reitittimen kautta oikean näkymän ja renderoimaan sivuista valmiin HTML-sivun, jonka se palauttaa selaimelle. (Strimpel & Najim, 2016, 4.) Isomorfinen sovellus on siis palvelinpuolen renderoinnin vuoksi kaikkien hakukoneiden indeksoitavissa, muttei silti luovu hyvästä käytettävyydestään ja nopeasta toimivuudestaan selainpuolen renderoinnin ansiosta (Strimpel & Najim, 2016, 12).

Jotta sama koodi olisi mahdollista renderoida sekä selain- että palvelinpuolella, tarvitaan palvelin, joka ymmärtää JavaScriptia. Node.js tarjoaa V8 JavaScript-ytimellä varustetun palvelinympäristön, mutta useat yritykset ja organisaatiot eivät välttämättä ole tilanteessa, jossa heillä olisi valmiuksia käyttää ja ylläpitää Node.js-palvelinta. Kuitenkin palvelimille, joilla on käytössään Java, Ruby, Python tai PHP, on olemassa kaksi

ratkaisua. Näissä ratkaisuissa haittapuolena esiintyy arkkitehtuurin monimutkaisuus, joka tekee sovelluksesta raskaamman. (Strimpel & Najim 2016, 16.)

1. Node.js-prosessin ajaminen normaalinpalvelimen rinnalla paikallisena tai etäisenä renderointipalveluna.
2. Upotetun JavaScript runtime:n käyttö. Esimerkkinä Nashorn JavaScript engine, joka tulee Java 8:n osana.

Opinnäytetyön osuus isomorfisista sovelluksista rajataan kuitenkin sovellusmalliin, jonka palvelimella toimii Node.js-ympäristö.

8 Isomorfisen/universaalin sovelluksen kehittäminen

Opinnäytetyön toisena tuotoksena kehitettiin isomorfinen/universaali sovellus. Sovellus käyttää pohjanaan React Cool Starter nimistä sovelluspohjaa, jonka on rakentanut Github käyttäjä Welly Shen. Kyseinen sovelluspohja valittiin siksi, että sitä päivitettiin säännöllisesti ja se sisälsi isomorfisen sovelluksen tekemiseen tarvittavat lisäosat.

Sovellus toimii käytännössä siten, että selain lähettää ns. pääpyynnön palvelimelle. Kun palvelin saa pyynnön se käsittelee sen Express.js:n avulla, joka katsoo, sopiiko URL-kenttään syötetty osoite minkään reitittimessä määriteltyjen URL-osoitteiden kanssa yhteen. (Ks. Liite 3) Mikäli sivu löytyy, palvelin käy reitittimen kautta siihen linkitetyn datan läpi ja renderoi sivun, joka palautetaan selaimelle. Tämä palautettu sivu on palvelinrenderoinnin vuoksi kaikkien hakukoneiden tulkittavissa ja indeksoitavissa. (Santiago, 2017.)

Kuvitellaan, että käyttäjä on verkkosivustolle saapuessaan päätynyt etusivulle. Serveri siis käsittelee edellä mainitulla tavalla etusivun ja selain näyttää sen käyttäjälle. Kun käyttäjä haluaa vaihtaa etusivulta esimerkiksi yhteystietosivulle, palvelin ei enää renderoikkaan sivua käyttäjälle vaan sivu muodostetaan SPA-tekniikoilla suoraan selaimessa. Palvelin huolehtii kuitenkin, että näkymät renderoidaan myös palvelimella SPA prosessin taustalla, jotta hakukoneilla olisi aina käsillä täydellinen HTML-versio sivusta.

Ongelmaksi Isomorfisessa ratkaisussa koituivat kuitenkin SPA-sovellusrakenteen asynkroniset pyynnöt. Jollei niitä varten olisi rakennettu erillistä hallintatapaa Redux:lla, niitä ei olisi saatu renderoitua ajoissa, sillä express.js:lla kirjoitettu palvelimen toiminnallisuus ajoi itsensä ennen kuin asynkronisten pyyntöjen paluudata oltiin keretty asettaa paikoilleen. Asynkroniset pyynnöt olisivat siis jääneet JavaScriptia tulkitsemattoman hakukoneen ulottumattomiin.

Opinnäytetyön aikana kehitetyssä isomorfisessa sovelluksessa hyödynnetään Redux Storea asynkronisten pyyntöjen suorittamiseksi. Reduxin virallisilla sivuilla vuonna 2017 kerrotaan, että Redux on säiliö JavaScript sovellusten stateille, joka kehitetiin yksinkertaistamaan JavaScript-sovelluksia. Statet säilyttävät sisällään tarpeellisia arvoja, joita sovellus tarvitsee. Tämän esimerkin tapauksessa statessa säilytetään asynkronisesti haettua dataa, sekä tietoa datan hakuprosessista. Reduxin päätoiminnot koostuvat seuraavista asioista:

> **Actions:** Lähettää datan sovelluksesta Redux Storeen ja toimivat ainoastaan tiedon lähteenä. Actions kuvaa siis, että jotain tapahtui, mutta ei tarkenna kuinka sovelluksen state muuttuu tapahtuman seurauksena. (Actions, 2017.)
>
> **Reducers:** Reducers-toiminnot sisältävät funktioita, jotka ottavat vastaan vanhan staten ja tekevät siihen Actioneiden kautta välitetyt muutokset ja palauttavat uudet statet sovellukselle. (Reducers, 2017.)
>
> **Store:** Redux:ssa kaikki sovelluksen statet on säiliöity yhteen JavaScript-olioon, jota säilytetään Redux Storessa. (Store, 2017.)

Reduxin omilla sivuilla kerrotaan, että asynkronisten pyyntöjen ja sovelluksen ulkopuolelta haettavan datan hakeminen onnistuu rakentamalla middleware-palvelu Redux thunk -lisäosan avulla. Middleware sijaitsee Actions-toimintojen ja sen vaiheen, milloin tavoitetaan Reducer-toiminnot, välissä. Redux thunk -lisäosan avulla voidaan lähettää Reducer-toiminnoille Actions-toiminnoilla asynkroninen kutsu,

jonka middleware tunnistaa ja välittää sen eteenpäin Reducerin-toiminnoille. Samaan aikaan middleware on ottanut yhteyttä ulkoiseen tiedonlähteeseen, esimerkiksi sovelluksen ulkopuolella toisella palvelimella sijaitsevaan JSON-tiedostoon, ja kun ulkoinen tiedonlähde palauttaa vastauksensa, middleware lähettää saadun datan Reducerille, joka sijoittaa sen paikoilleen. (Nickell, 2017.)

Redux Storen käyttö isomorfisessa sovellusrakenteessa aloitetaan rakentamalla normaalit Redux Actions -toiminnot ja Store. Tiedon noutamiseen käytetään Axios-lisäosaa datan hakijana, joka palauttaa promise:n sisäiseen funktioon. Actions-toiminnot rekisteröidään reitityksessä. (Ks. Liite 4) Reititystä kutsutaan palvelimella, jossa lähetetään myös Actions-toiminnot. (Ks. Liite 3) Shen kertoo sovelluspohjan dokumentoinnissa, että selaimen puolella Actions-toiminnot pitää herättää ReactJS:n componentDidMount()-funktion sisällä. Tämä takaa, että komponenttiin päästään käsiksi myös selaimen puolelta, jolloin samat Actions-toiminnot saadaan suoritettua myös selaimessa. (Shen, 2017.)

Isomorfinen sovellus yhdistää siis serverillä renderoidun sovelluksen ja SPA-sovelluksen kyvyt. Tällä keinolla saavutetaan sovellukselle hyvä käyttökokemus ja hakukoneystävällinen palvelinrenderointi. (Strimpel & Najim, 2016.)

9 Tutkimustulokset ja johtopäätökset

Tässä osiossa vastataan työn alussa kerrottuihin tutkimuskysymyksiin ja kootaan opinnäytetyöstä lyhyet tiivistelmät jokaisen kysymyksen alle. Nämä tiivistelmät kertovat mihin lopputulokseen tutkimuksessa ollaan päädytty ja millaisia johtopäätöksiä siitä voidaan muodostaa.

Millainen sivu on teknisesti hyvin hakukoneoptimoitavissa ja miksi hakukoneoptimointi kannattaa?

Tekniseltä kannalta katsottuna hyvin hakukoneoptimoidulla sivulla on otettu seuraavat asiat huomioon. Näkymillä on oltava oma näkymäkohtainen head-osionsa, joissa hakukoneille annettavan informaation kannalta tärkeitä tekijöitä ovat mm. title -tag, Meta kuvaus ja meta avainsana -atribuutit, Meta robots -atribuutti sekä Canonical-

atribuutti. Muita tärkeitä tekijöitä on laadukas käyttäjäystävällinen sisältö, tekstisisällön järkevä otsikointi otsikkotageilla sekä sisäiset linkitykset että ulkoiset linkitykset. On myös hyvä tarkistaa, kykeneekö Googlebot ymmärtämään näkymien sisällön Fetch as Google -työkalun avulla. Jos hakukonenäkyvyyttä toivotaan muiltakin hakukoneilta kuin Googlelta, tulee hakutuloksen osalta tärkeän sisällön renderointia selaimessa JavaScriptin avulla välttää.

Hakukoneoptimoinnin aiheuttamat hyödyt ovat seuraavat. Hyvin hakukoneoptimoitu sivu tavoittaa suuren yleisön ja pysyy hakukoneiden hakutuloksissa pitkään. Hakukoneoptimointi ei kustanna paljoa. Google tarjoaa hakukoneoptimoiduista sivuista dataa ja analyysejä, joiden avulla markkinoiden kilpailussa pysyminen on helpompaa. Hyvin hakukoneoptimoitu sivu on usein käyttäjäystävällinen, sillä hakukoneoptimointi ja käytätettävyys kulkevat käsi kädessä.

Miten SPA-sovellukset ja niiden hakukoneoptimointi eroavat perinteisistä sivuista?

Single Page Application eli SPA-sovellus toimii selaimessa eikä käytön aikana tarvitse sivun uudelleen latausta palvelimelta, sillä se lataa tarvittavat resurssit, kuten HTML, CSS ja Scriptit vain kerran sovelluksen käytön aikana. Näiden resurssien avulla rakennetaan kaikki sovelluksen näkymät ja toiminnot JavaScriptin avulla ilman uutta palvelimelle osoitettua pyyntöä ja sivuston uudelleenlatausta.

Palvelimella sijaitsevia tietoja käsitelläkseen SPA-sovellus ottaa palvelimeen tarvittaessa yhteyden asynkronisilla kutsuilla. Kun sisältö palvelimelta kutsutaan sovellukseen asynkronisesti, kutsun vastausta ei jäädä odottamaan. Sovellusnäkymä siis renderoidaan esitettäväksi selaimeen odottamatta kutsun vastauksena saapuvaa dataa. Kun kutsun vastausdata saapuu, se renderoidaan sivunäkymään paikoilleen jälkikäteen.

SPA-sovellukset eivät ole lähtökohtaisesti hakukoneystävällisiä ja ne hakukoneet, jotka eivät tulkitse JavaScriptia, eivät kykene indeksoimaan SPA-sovelluksia. Google on kuitenkin lisännyt hakuunsa tuen JavaScriptin suorittamiseen ja kykeni indeksoimaan opinnäytetyön ReactJS-testisovelluksen, kunhan sovelluksen rakenteessa huomioitiin indeksointiin vaikuttavat tekijät, jotka mainittiin ylemmissä luvuissa.

Perinteiset verkkosivut ja -sovellukset puolestaan ovat lähes jatkuvasti yhteydessä palvelimeen. Käyttäjän toimet aiheuttavat usein tarpeen palvelimelle lähetettävään HTTP-pyyntöön, jonka aikana data kulkee eri tasojen läpi pyyntöä koskevia tietoja keräten ja jonka jälkeen palvelin renderoi näistä tiedoista selaimelle palautettavan HTML-sivun. Hakukoneilla ei ole ongelmia tulkita perinteisiä sivuja.

Kuinka parantaa hakokoneiden indeksointia ReactJS-pohjaisissa sovelluksissa?

Opinnäytetyössä todettiin, että Google pystyy indeksoimaan oikein toteutetun ReactJS sovelluksen. Jos tarve on kutenkin saada näkyvyyttä myös muilla hakukoneilla, on käännyttävä isomorfisen JavaScriptin puoleen. Tästä on kerrottu enemmän otsikon isomorfinen/universaali web-sovellus alla.

10 Pohdinta

Opinnäytetyön tuloksena tuotettiin kaksi ReactJS-sovellusta; tavallinen sekä isomorfinen ReactJS-sovellus. Tavallinen ReactJS-sovellus sijoitettiin Heroku:n pilvipalvelimelle ja sen indeksoitumista Googleen havainnoitiin. Isomorfisen sovelluksen osalta kehittämistyön lisäksi ei tehty muuta, mutta sen voi ladata ja asentaa omalle koneelleen lokaalisti osoitteesta https://github.com/hennasauranen/opinnaytetyo-2017.

Opinnäytetyön aikana havaittiin, että normaali ReactJS-sovellus indeksoitui Googleen täysin. Apuna indeksoitumiseen käytettiin Fetch as Google -työkalulla lähetettyjä pyyntöjä ja Googleen lisättyä sivustokarttaa. Molemmat onnistuivat indeksoimaan sivuja.

Fetch as Google -työkalulla lähetetyt yksittäiset pyynnöt indeksoituivat heti, mutta sivustokartan käsittelyssä aikaa kului noin pari viikkoa ja osa sivuista indeksoitui ensin väärin, mutta Google sai lopulta itsestään korjattua asian. Jatkossa tätä asiaa voisi jäädä tukimaan enemmänkin, sillä nyt jäi avoimeksi, oliko syy tähän missä.

Tutkimus suoritettiin melko pienellä testisovelluksella, joten tutkimusta voisi jatkaa testaamalla isomman näkymäkokonaisuuden indeksoitumista. SPA-sovelluskehyksiä ja kirjastoja on ReactJS:n lisäksi monia muita. Lisätutkimusta voisi siis tehdä myös

muiden SPA-sovellusten indeksoitumisesta ja kuinka Google kykenee niillä rakennetut sivut indeksoimaan.

Tutkimuksen tulosten perusteella oletetaan, että Googlen tulisi pystyä indeksoimaan muitakin JavaScript-pohjaisia sovelluksia kuin ReactJS-sovelluksia, kunhan sovellukset on vain rakenteellisesti tehty Google huomioon ottaen. Jotta JavaScript-sovellus saadaan indeksoitua Googleen, on sovellus rakennettava siten, että siinä ollaan otettu huomioon indeksointiin vaikuttavat tekijät, jotka ovat lähes samat myös serveripuolen renderointiin luottavilla sovelluksilla. Nämä tekijät mainitaan opinnäytetyössä useaan otteeseen.

Tutkimuksen aikana ei selvinnyt voidaanko Googlen kykyyn luottaa indeksoitaessa suuria JavaScript sivukokonaisuuksia, joilla on paljon näkymiä. ReactJS-pohjainen testisovellus kuitenkin todistaa, että pienen sivukokonaisuuden indeksoitumisessa ei ollut suurempia ongelmia.

Muut hakukoneet kuin Google eivät kykene yhtä tehokkaasti suorittamaan JavaScriptia. Ainoa mahdollisuus indeksoitua näiden hakukoneiden tietokantoihin on rakentaa isomorfinen SPA-sovellus, joka takaa hakukoneoptimoitavuutensa lisäksi myös parhaan käytettävyyden SPA-tekniikoin, sekä JavaScriptin pettäessä perääntymisen palvelinpuolen renderointiin.

Lähteet

Actions, 2017. Redux.js.org. Viitatu 1.11.2017.
https://redux.js.org/docs/basics/Actions.html.

AJAX Crawling (Deprecated) 2009,Google. Viitattu 13.9.2017.
https://developers.google.com/webmasters/ajax-crawling/docs/specification#bidirectional-between--url-to-_escaped_fragment_-url.

AsyncFlow, 2017. Redux.js.org. Viitattu 1.11.2017.
https://redux.js.org/docs/advanced/AsyncFlow.html.

Barysevich, A 2016. 9 Biggest differences Yandex & Google seo. Viitattu 31.10.2017.
https://www.searchenginejournal.com/9-biggest-differences-yandex-vs-google-seo/168628/.

Build and submit a sitemap, 2017. Google. Viitattu 31.10.2017.
https://support.google.com/webmasters/answer/183668?hl=en&ref_topic=4581190
.

Cauthorn, D, 2016. Single Page Applications (SPA) and the SEO problem. Viitattu 17.10.2017.
https://adkgroup.com/insights/single-page-applications-spa-and-seo-problem.

Deprecating our ajax crawling scheme, 2015. Google. Viitattu 14.9.2017.
https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html.

De Sousa Antonio, C, 2015. *Pro React*. [Kustannuspaikka tuntematon]: Apress.

Emmit A , Scott Jr. 2016. SPA Design and Architecture. New York: Manning.

Ewerlöf, A, 2017. You might not need to transpile your JavaScript. Viitattu 6.11.2017.
https://medium.freecodecamp.org/you-might-not-need-to-transpile-your-javascript-4d5e0a438ca.

Edwards C, 2014. Is the meta keyword tag still used by Google, Bing and Yahoo? Viitattu 20.9.2017. https://chrisedwards.me/seo/keyword-meta-tag-google/.

Fink, G & Flatow, I. 2014. Pro Single Page Application Development. New York: Apress.

Fishkin, R, 2017. Opinnäytetyön raportointi. Viitattu 5.11.2017.
https://moz.com/beginners-guide-to-seo/how-search-engines-operate.

Gackenheimer, C, 2015. Introduction to React. Apress.

How Google Search Works, 2017. Google. Viitattu 20.10.2017. https://support.google.com/webmasters/answer/70897?hl=en&ref_topic=4558960.

Learn about sitemaps, 2017. Google. Viitattu 1.11.2017. https://support.google.com/webmasters/answer/156184?hl=en.

Meta tags that Google understands, 2017. Google. Viitattu 30.9.2017. https://support.google.com/webmasters/answer/79812?hl=en.

Mäkäräinen, P, 2017. Web-sovelluskehitys SPA-arkkitehtuurilla. Viitattu 23.9.2017. https://www.theseus.fi/bitstream/handle/10024/127908/makarainen_pyry.pdf?sequence=1&isAllowed=y.

Nickell, C, 2016. Async Redux Explaned. Viitattu 2.11.2017. https://medium.com/@beBrllnt/async-redux-explained-112c002043d5.

Other content considerations, 2017, Google, viitattu 1.10.2017. https://support.google.com/webmasters/answer/96569?hl=en&ref_topic=4617741.

A proposal for making AJAX crawlable, 2009. A proposal for making AJAX crawlable. Viitattu 14.9.2017. https://webmasters.googleblog.com/2009/10/proposal-for-making-ajax-crawlable.html.

Reducers, 2017. React.js.org. Viitattu 1.11.2017. https://redux.js.org/docs/basics/Reducers.html.

Redux, 2017. Redux.js.org. Viitattu 2.11.2017. https://redux.js.org/.

Santiago, A, 2016. Introducing Universal Web Applications Viitattu 30.10.2017. http://www.acuriousanimal.com/2016/08/10/universal-applications.html.

Shen, W, 2017. React Cool Starter. Viitattu 12.11.2017. https://github.com/wellyshen/react-cool-starter.

Shenoy, A & Prabhu, A, 2016. Introducing SEO. Apress.

Simplify multiple sitemap management, 2017. Google. Viitattu 31.10.2017. https://support.google.com/webmasters/answer/75712.

Strimpel, J & Najim, M, 2016. Building isomorphic JavaScript apps: From concept to implementation to real-world solutions. Sebastopol, CA: O'Reilly.

Store, 2017. React.js.org Viitattu 1.11.2017. https://redux.js.org/docs/basics/Store.html.

Tsonev, K, 2016. Deep dive into client-side routing. Viitattu 1.10.2017.

http://krasimirtsonev.com/blog/article/deep-dive-into-client-side-routing-navigo-pushstate-hash.

Understanding web pages better, 2014. Google. Viitattu 14.9.2017. https://webmasters.googleblog.com/2014/05/understanding-web-pages-better.html.

Use canonical URLs, 2017, Google. Viitattu 20.10.2017 https://support.google.com/webmasters/answer/139066.

Vega, J, 2017. Client-side vs. server-side rendering: why it's not all black and white. Viitattu 15.9.2017. https://medium.freecodecamp.org/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d.

Webpack, 2017. Viitattu 10.10.2017. https://webpack.github.io/docs/webpack-dev-server.html.

What is DOM?, 2017. Viitattu 1.11.2017. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

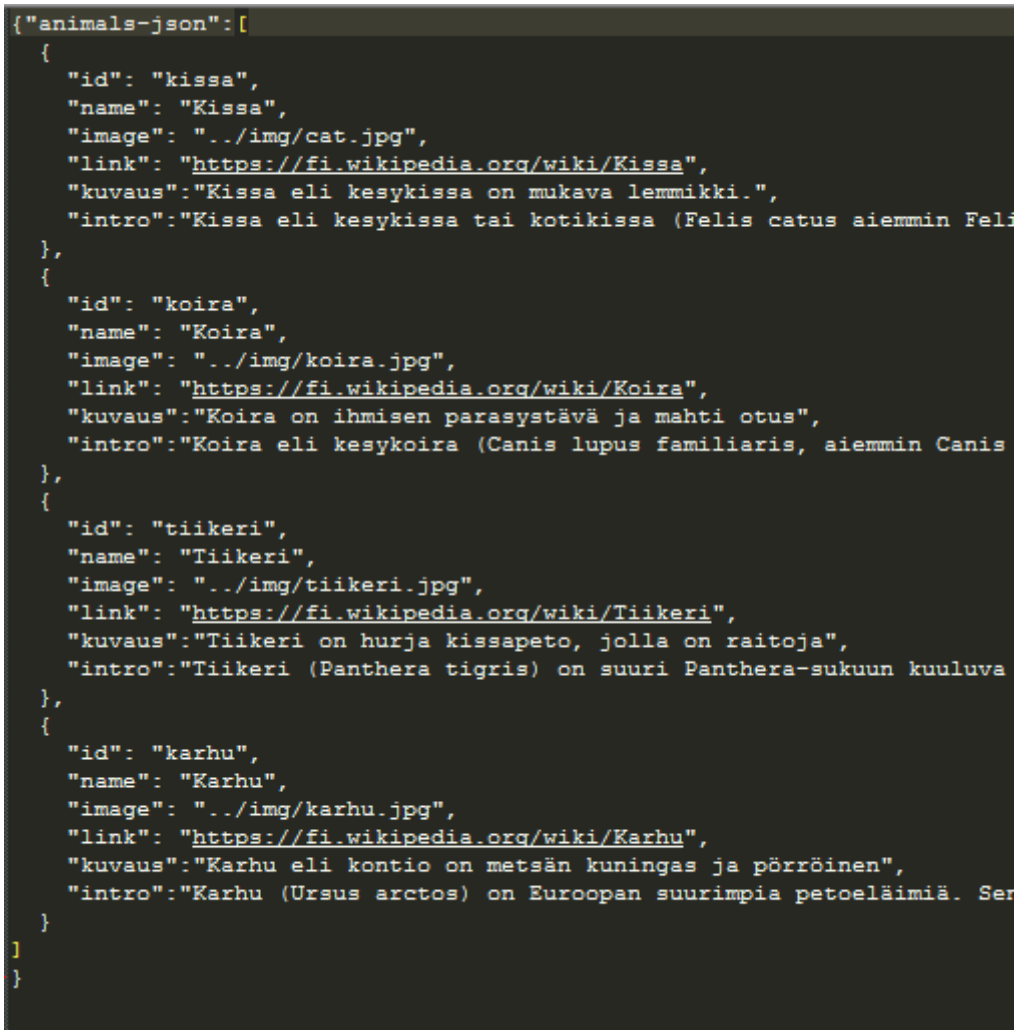
Liitteet

Liite 1. Komponentti, johon tiedot haetaan asynkronisesti.

```
export default class UserInfo extends React.PureComponent {
  constructor(props) {
    super();
    this.state = {
      animals: [],
      gotAnimals: false
    };
  }
  componentWillMount() {
    axios.get('https://api.myjson.com/bins/1aeuin').then((res) => {
      const result = res.data;
      this.setState({ animals: result, gotAnimals: true });
    });
  }
  filterAnimal(array) {
    const url = window.location.pathname;
    const urlsplit = url.split('/').slice(-1)[0];
    const id = urlsplit;
    console.log(id);
    let i = 0;
    const animalArray = array.length;
    for (i = 0; i < animalArray; i++) {
      if (array[i].id === id) {
        const returnAnimal = array[i];
        return returnAnimal;
      }
    }
  }
  render() {
    let animal = this.filterAnimal(this.state.animals);
    if (this.state.gotAnimals === false) {
      return (<div>'loading...'</div>)
    }
    else {
      if (!animal) return (<div><NotFoundPage/></div>)

      return (
        <div className="container">
          <Helmet>
            <meta charSet="utf-8"/>
            <title>{animal.name}</title>
            <meta name="description" content={animal.kuvaus}/>
          </Helmet>
          <div className="breadcrumb">
            Olet tässä:
            <Link to={'/'}>Etusivu</Link>
            &raquo;
            <Link to={'/elaimet'}>Eläimet</Link>
            &raquo;
            <Link to={`/elaimet/${animal.id}`}>{animal.id}</Link>
          </div>
          <div className="picture-container">
            <h1 className="name">{animal.name}</h1>
          </div>
          <p>{animal.intro}</p>
          <a href={animal.link}>wikipedia</a>
        </div>
      );
    }
  }
}
```

Liite 2. JSON-tiedosto, josta sovellus hakee asykronisesti sisältöä



```
{"animals-json":[
  {
    "id": "kissa",
    "name": "Kissa",
    "image": "../img/cat.jpg",
    "link": "https://fi.wikipedia.org/wiki/Kissa",
    "kuvaus":"Kissa eli kesykissa on mukava lemmikki.",
    "intro":"Kissa eli kesykissa tai kotikissa (Felis catus aiemmin Feli
  },
  {
    "id": "koira",
    "name": "Koira",
    "image": "../img/koira.jpg",
    "link": "https://fi.wikipedia.org/wiki/Koira",
    "kuvaus":"Koira on ihmisen parasystävä ja mahti otus",
    "intro":"Koira eli kesykoira (Canis lupus familiaris, aiemmin Canis
  },
  {
    "id": "tiikeri",
    "name": "Tiikeri",
    "image": "../img/tiikeri.jpg",
    "link": "https://fi.wikipedia.org/wiki/Tiikeri",
    "kuvaus":"Tiikeri on hurja kissapeto, jolla on raitoja",
    "intro":"Tiikeri (Panthera tigris) on suuri Panthera-sukuun kuuluva
  },
  {
    "id": "karhu",
    "name": "Karhu",
    "image": "../img/karhu.jpg",
    "link": "https://fi.wikipedia.org/wiki/Karhu",
    "kuvaus":"Karhu eli kontio on metsän kuningas ja pörröinen",
    "intro":"Karhu (Ursus arctos) on Euroopan suurimpia petoeläimiä. Ser
  }
]
}
```

Liite 3. Palvelin saa pyynnön ja käsittelee sen Express.js:lla

```
app.get('*', (req, res) => {
  if (__DEV__) webpackIsomorphicTools.refresh();

  const history = createHistory();
  const store = configureStore(history);
  const renderHtml = (store, htmlContent) => { // eslint-disable-line no-shadow
    const html = renderToStaticMarkup(<Html store={store} htmlContent={htmlContent} />);
    return `<!doctype html>${html}`;
  };
  // If __DISABLE_SSR__ = true, disable server side rendering
  if (__DISABLE_SSR__) {
    res.send(renderHtml(store));
    return;
  }
  // Load data on server-side
  const loadBranchData = () => {
    const promises = [];
    routes.some((route) => {
      const match = matchPath(req.url, route);
      // $FlowFixMe: the params of pre-load actions are dynamic
      if (match && route.loadData) promises.push(route.loadData(store.dispatch, match.params));
      return match;
    });
    return Promise.all(promises);
  };
  // Send response after all the action(s) are dispathed
  loadBranchData()
    .then(() => {
      // Setup React-Router server-side rendering
      const routerContext = {};
      const htmlContent = renderToString(
          <Provider store={store}>
            <StaticRouter location={req.url} context={routerContext}>
              <App />
            </StaticRouter>
          </Provider>,
        );
        // Check if the render result contains a redirect, if so we need to set
        // the specific status and redirect header and end the response
        if (routerContext.url) {
          res.status(301).setHeader('Location', routerContext.url);
          res.end();
          return;
        }
        // Checking is page is 404
        const status = routerContext.status === '404' ? 404 : 200;
        // Pass the route and initial state into html template
        res.status(status).send(renderHtml(store, htmlContent));
      })
      .catch((err) => {
        res.status(404).send('Not Found :(');
        console.error(`==> 😭  Rendering routes error: ${err}`);
      });
  });
```

Liite 4. Redux Actionit kutsutaan retityksessä

```
import type { Dispatch } from './types';
import { fetchUsersIfNeeded } from './containers/Home/action';
import { fetchUserIfNeeded } from './containers/UserInfo/action';
import HomePage from './containers/Home';
import List from './containers/List';
import UserInfoPage from './containers/UserInfo';
import NotFoundPage from './containers/NotFound';

export default [
  {
    path: '/',
    exact: true,
    component: HomePage, // Add your route here
    loadData: (dispatch: Dispatch) => Promise.all([
      dispatch(fetchUsersIfNeeded()), // Register your server-side call action(s) here
    ]),
  },
  {
    path: '/elaimet',
    exact: true,
    component: List, // Add your route here
    loadData: (dispatch: Dispatch) => Promise.all([
      dispatch(fetchUsersIfNeeded()), // Register your server-side call action(s) here
    ]),
  },
  {
    path: '/elaimet/:id',
    component: UserInfoPage,
    loadData: (dispatch: Dispatch, params: Object) => Promise.all([
      dispatch(fetchUserIfNeeded(params.id)),
    ]),
  },
  {
    path: '*',
    component: NotFoundPage,
  },
];
```

Liite 5. Testisovellus indeksoitui Googlen tietokantaan