Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Aaron Hakala

SPA-arkkitehtuurin saavutettavuusongelmat

Metropolia Ammattikorkeakoulu

Medianomi

Viestinnän tutkinto-ohjelma

Opinnäytetyö

16.11.2018

Tekijä(t) Otsikko Sivumäärä Aika	Aaron Hakala SPA-arkkitehtuurin saavutettavuusongelmat 23 sivua + 1 liitettä 16.11.2018
Tutkinto	Medianomi
Tutkinto-ohjelma	Viestinnän tutkinto-ohjelma
Suuntautumisvaihtoehto	Digitaalinen viestintä
Ohjaaja(t)	Lehtori Markus Norrena

Opinnäytetyön aiheena on selvittää SPA-sovellusten saavutettavuusongelmat ja etsiä niihin ratkaisuja. Aihe syntyi kirjoittajan omasta tarpeesta työelämässä saavutettavuusdirektiivin tuomien muutoksien myötä. Saavutettavuusdirektiivin asettamat vaatimukset julkisen hallinnon sovelluksille on pakottanut kehittäjiä perehtymään saavutettavuuteen ja etsimään ratkaisuja saavutettavuusongelmiin. Tavoitteena on löytää ongelmia SPA-sovelluksissa, jotka vaativat toimenpiteitä julkisen hallinnon sovelluksissa. Tavoitteena ei kuitenkaan ole löytää näihin ongelmiin absoluuttisia ratkaisuja, vaan tuoda ongelmat esille ja löytää niihin yksinkertaiset ratkaisut, joita voi hyödyntää myös muissa konteksteissa. Aihetta tarkastellaan ohjelmoijan näkökulmasta.

Työn teoriaosuudessa käsitellään perinteisiä verkkosovelluksia ja SPA-sovelluksia. Samalla tarkastellaan niiden keskeisiä eroja. Teoriaosuudessa käsitellään myös saavutettavuutta, saavutettavuusdirektiiviä ja ruudunlukijoiden käyttöä.

Tutkimusosuudessa etsitään SPA-sovelluksien ja perinteisten verkkosovellusten erojen takia syntyviä saavutettavuusongelmia, joita reflektoidaan WCAG 2.0 -standardin AA-tasoa vasten. SPA-sovellusten saavutettavuusongelmien tarkastelu rajataan käsittelemään ainoastaan ruudunlukijan käyttäjiä koskevia ongelmia. Löytyneitä ongelmia etsitään myös jo olemassa olevista palveluista. Lopussa saavutettavuusongelmiin kehitetään ratkaisuksi yksinkertainen esimerkkisovellus, jossa korjataan löytyneet ongelmat.

Lopuksi työssä todetaan, että saavutettavuusdirektiivin tuomat uudet vaatimukset pakottavat kehittäjiä miettimään saavutettavuutta julkisen hallinnon sovelluksien kehityksessä. Samalla parannetaan kehittäjien ymmärrystä saavutettavuudesta ja tämän uskotaan heijastuvan myös muihin sovelluksiin. Kehittäjien on kuitenkin tiedostettava SPA-sovellusten puutteet, jotta ongelmat voidaan korjata. SPA-sovellusten uskotaan myös avaavan uusia mahdollisuuksia parantaa käyttökokemusta avustavan teknologian käyttäjille.

Avainsanat	Single-page application, saavutettavuus, JavaScript, saavutettavuusdirektiivi, navigointi, WCAG 2.0

Author(s) Title Number of Pages Date	Aaron Hakala Accessibility Issues in Single-page Applications 23 pages + 1 appendices 16 November 2018
Degree	Bachelor of Culture and Arts
Degree Programme	Media
Specialisation option	Digital Media
Instructor(s)	Markus Norrena, Senior Lecturer

The aim of this Bachelor's thesis is to discover accessibility problems in Single-page applications and to find solutions to those problems. The subject stems from the authors need to build accessible Single-page applications in the everyday work life. The EU directive on the accessibility of websites and mobile applications has forced programmers to research accessibility and find solutions to accessibility problems. The goal is to find problems that require action in projects falling under the directive. The goal is not to provide an absolute solution to these problems but to point them out and provide tools and a simple proof-of-concept solution that can be used in different contexts. The subject is viewed from a programmer's perspective.

The theoretical section describes traditional web applications and single-page applications. Their main differences are also covered in the same section. Accessibility, the accessibility directive and the usage of screen-readers are also discussed later on in the section.

The research section focuses on looking for accessibility problems in single-page applications when compared to the traditional web applications. These problems will be examined using the WCAG 2.0 standards AA -level. The examining of SPA accessibility problems will be limited only to the problems affecting screen-reader users. The section will also look into a few popular services to find if the issue has been resolved. In the end a simple solution will be developed that will fix the accessibility issues in the single-page applications.

Finally, the work states that the new requirements of the Accessibility Directive force developers to think more about accessibility while they are developing public administration applications. Due to this, developers' gain a deeper understanding of accessibility and it is believed to reflect onto other applications as well. However, developers need to recognize the shortcomings of single-page applications to be able to remedy the problems. Single-page applications are also expected to open up new opportunities to help improve user experience for assistive technology.

Keywords	Single-page application, accessibility, JavaScript, web accessibility directive, navigation, WCAG 2.0

Sisällys

1 Johdanto

Olen valinnut opinnäytetyössäni aiheeksi SPA-sovellukset ja niiden uniikit saavutettavuusongelmat. SPA-sovellukset (engl. Single-page application) ovat sovelluksia, joissa navigoitaessa ei sivuja koskaan ladata uudestaan. Kaikki näytetään yhden HTML-sivun sisällä. Esimerkiksi sopii Gmail-sähköpostisovellus, jossa verkkosivu ei uudelleen latau, vaikka sivulla näytetään ja ladataan eri sisältöjä. SPA-sovellukset mahdollistavat nopeamman interaktion sivun kanssa ja samalla käyttäjille paremman käyttökokemuksen. Sovellukset eivät joudu uudelleen lataamaan ja rakentamaan sisältöä jokaisella navigaatiolla, vaan JavaScriptin avulla pystyvät hyödyntämään jo ladattuja ja rakennettuja elementtejä. SPA-sovellusten kehittäminen on helpottunut verkon infrastruktuurin kehittyessä, ja samalla on niiden suosio myös kasvanut räjähdysmäisesti. Monet suuret palvelut, kuten Youtube, Facebook, Instagram ja Gmail, on uudelleen rakennettu käyttäen SPA-arkkitehtuuria. Modernit JavaScript-sovelluskehykset kuten React.js, Angular.js 2 ja Vue.js ovat kaikki suosittuja kehyksiä SPA-sovellusten kehityksessä. Frontend-kehittäjänä työskennellessäni olen kuitenkin huomannut muutamia saavutettavuusongelmia SPA-sovelluksissa.

22.12.2016 voimaan tullut saavutettavuusdirektiivi asetti uusia vaatimuksia saavutettavuudesta julkisen hallinnon verkkopalveluille (Valtiovarainministeriö 2018). Saavutettavuusdirektiivillä halutaan edistää kaikkien mahdollisuutta toimia tasavertaisesti digitaalisessa yhteiskunnassa huolimatta yksilön omista rajoitteista. Esteettömyys koskee fyysistä maailmaa, kun taas saavutettavuus liittyy digitaalisiin palveluihin. Ohjelmistosuunnittelijan työssä olen joutunut kehittämään julkisen hallinnon verkkopalveluita, joista monissa on SPA-arkkitehtuuri käytössä. Opinnäytetyöni kautta haluan tuoda saavutettavuusongelmat esille, jotta voin löytää ratkaisun niihin omissa sekä julkisen hallinnon projekteissa. Opinnäytetyön lopussa toteutetaan pienimuotoinen esimerkkisovellus.

Tarkastelen opinnäytetyössäni SPA-sovellusten eroja perinteisiin verkkosovelluksiin ja näistä eroista syntyviä saavutettavuusongelmia. SPA-sovelluksia on mahdollista tehdä monilla eri JavaScript-sovelluskehyksillä, mutta tässä työssä sovelluksia tarkastellaan React.js:n kautta, ja esimerkkisovellus on myös tehty Reactilla. Saavutettavuuden tarkastelu rajataan käsittelemään ainoastaan sokeihin kohdistuviin saavutettavuusongelmiin. En käsittele työssä saavutettavuusongelmia, jotka kohdistuvat muihin ryhmiin. Saa-

vutettavuutta peilataan WCAG 2.0 -standardin AA-tasoa vasten, joka on määritetty saavutettavuusdirektiivissä minimitasoksi. WCAG 2.0 -standardissa internetsivuille on annettu ohjelmointiin keskittyviä huomioita, jotka ohjelmoijan tulisi ottaa huomioon sivua kehittäessään. Tästä syystä tutkimus on hyvin ohjelmointipainotteinen ja kohderyhmänä ovat ohjelmoijat. Opinnäytetyöni tutkimus on laadullista, koska aineisto, lähteet ja analyysitavat ovat laadullisia (Saaranen-Kauppinen & Puusniekka 2006).

Tämän opinnäytetyön tavoitteena on auttaa lukijaa ymmärtämään SPA-sovellusten saavutettavuusongelmat ja tarjota heille työkaluja niiden ratkaisemiseksi. Tavoitteena ei kuitenkaan ole absoluuttisten ratkaisuiden löytäminen, vaan auttaa kehittäjiä näkemään haasteet SPA-sovellusten kehityksessä ja löytämään heidän tapauksiinsa sopivat ratkaisut.

Toisessa luvussa käsitellään verkkosovelluksia ja internetin toimintaa. Luvussa kolme käsitellään SPA-sovellusten toimintaa ja kuinka nämä eroavat perinteisistä verkkosovelluksista. Neljännessä luvussa kerrotaan, mikä on saavutettavuus, ja käsitellään myös saavutettavuusdirektiivin tuomia muutoksia julkisen hallinnon verkkosivuihin ja -palveluihin. Samassa luvussa käsitellään myös pintapuolisesti ruudunlukijoiden toimintaa ja käyttöä. Luvussa viisi tutkitaan SPA-sovelluksien uudesta arkkitehtuurista syntyviä saavutettavuusongelmia ja tuodaan esille asioita, joita sovelluskehittäjien tulisi ottaa huomioon kehityksessä. Luvussa kuusi toteutetaan esimerkkisovellus, jossa ongelma on ratkaistu. Viimeinen luku sisältää työn yhteenvedon ja ratkaisujen esittelyn.

2 Verkkosovellukset

Verkkosovellukset toimivat perustana monelle yritykselle digitaalisten palveluiden muodossa. Jotkin tarjoavat mahdollisuuden viestiä toiselle puolelle maailmaa, toiset taas auttavat pankkiasioiden hoitamisessa. Verkkosovellukset ovat avanneet uusia mahdollisuuksia yrittäjille ja yksityishenkilöille.

Verkkosovellukset rakentuvat erilaisista verkkoteknologioista. On palvelimia, asiakkaita, tietokantoja ja yhteyksiä niiden välillä. Palvelin on tietokone, joka vastaa verkkosivun säilytyksestä. Asiakas on toinen tietokone, joka pyytää palvelimelta verkkosivua, jonka voisi näyttää. Palvelin lähettää asiakkaalle verkkosivun, ja asiakas rakentaa tästä näkymän käyttäjälle. (IBM Knowledge Center n.d.) Palvelinpuolelle on omia ohjelmointikieliä ja asiakaspuolelle omia, mutta jotkin ohjelmointikielet toimivat kummallakin puolella.

Asiakassovelluksen käyttöliittymä rakentuu verkon peruspalikoista HTML- ja CSS-ohjelmointikielistä. HTML tarjoaa ohjelmoijalle mahdollisuuden lisätä tekstiä, kuvia, taulukoita, nappeja ja paljon muuta. (Flanagan 2011.) HTML vastaa sivun rakenteesta ja CSS taas tämän rakenteen ulkoasusta (W3C 2016). Näiden lisäksi verkkosivuilla käytetään paljon myös JavaScriptiä. JavaScriptiä käytettiin ennen vain pienenä lisämausteena sivuilla tarjoamalla käyttäjälle jotain ylimääräisiä efektejä tai pientä toiminnallisuutta, mutta nykyään monet sivut rakentuvat kokonaan JavaScriptin ympärille.

Asiakkaan ja palvelimen välisen datansiirron mahdollistamiseksi kehitettiin HTTP-protokolla (Hypertext Transfer Protocol). HTTP-protokollaa käytetään verkkosivujen ja verkkosivun tiedostojen pyytämiseen ja siirtoon. Kun asiakas haluaa verkkosivun, hän lähettää pyynnön palvelimelle. Tätä asiakkaan lähettämää pyyntöä kutsutaan HTTP-pyynnöksi. Perinteisesti pyyntöjä lähetetään aina, kun käyttäjä navigoi verkkosivujen välillä. Asiakassovellus lähettää HTTP-pyynnön (engl. request), ja palvelin vastaa pyyntöön vastauksella (engl. response). Tämän jälkeen asiakas näyttää palvelimelta saadun vastauksen sivulla. Pyyntöjä voidaan lähettää myös, kun siirrytään sovelluksen sisällä eri näkymiin tai kun tarvitaan lisää tietoa, esimerkiksi käyttäjän profiilitietoja. HTTP-protokollan lisäksi on myös paljon muita tiedonsiirtoprotokollia, mutta HTTP on osa verkon perustaa. (Haverbeke 2011.)

Verkkosivut olivat alkuun staattisia sivuja. Jokaiselle eri näkymälle kirjoitettiin oma HTML-dokumentti, joka sisälsi oman rakenteen ja ulkoasun. Staattiset sivut voivat sisältää multimediaa, kuten videoita tai kuvia, mutta sama sisältö näkyy jokaiselle käyttäjälle. Ainoastaan lähdekoodia muuttamalla pystyy sivun sisältöä vaihtamaan. Tämän jälkeen kehitettiin dynaamiset sivut. Dynaamiset sivut luovat sivun sisällön tilanteesta riippuen. Dynaamisissa sivuissa sisältö voi muuttua monen tekijän takia. Esimerkiksi sosiaalisissa medioissa sivun sisältö on jokaiselle käyttäjälle uniikki. Sosiaalisen median sovelluksen sisältöön voi muun muassa vaikuttaa kellonaika, sisältöä katseleva käyttäjä, käyttäjän aikaisempi käyttäytyminen sovelluksessa ja moni muu asia. Dynaamiset sivut luotiin alkuun aina palvelimen puolella ja asiakas sai valmiin sivun näytettäväksi. (Mangesh 2016.)

Dynaamisten sivujen yleistyttyä JavaScriptin rooli on tullut tärkeämmäksi verkko ympäristössä. JavaScript ei ole enää vain lisähöystettä, vaan yksi verkkosivujen tärkeimmistä ohjelmointikielistä HTML:n ja CSS:n rinnalla. JavaScript mahdollistaa nykyaikaiset interaktiiviset verkkosovellukset, koska se tarjoaa ohjelmoijalle mahdollisuuden kontrolloida

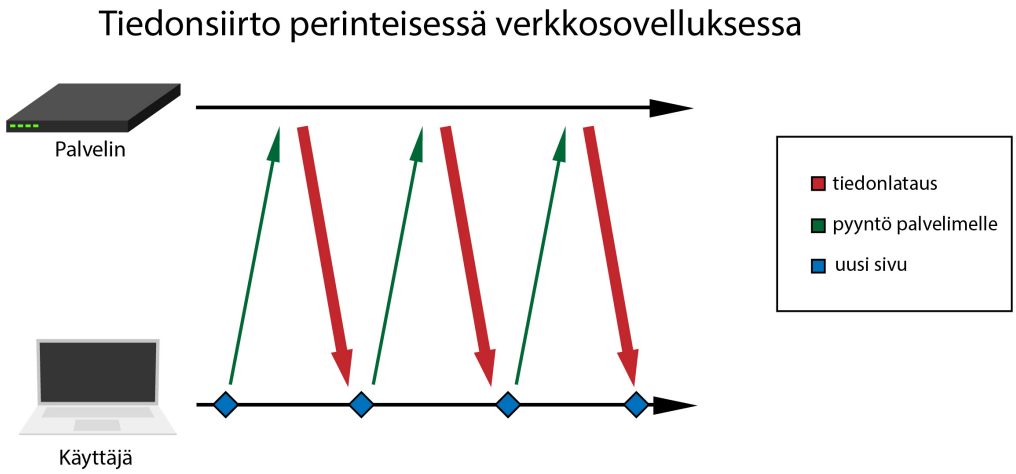
sivun sisältöä. Nykyään monet sivut tarvitsevat JavaScriptiä toimiakseen ja JavaScript toimii myös SPA-sovellusten tärkeimpänä osana (Neoteric 2016).

JavaScriptin lisäksi yksi SPA-sovellukset mahdollistanut teknologia on XMLHttpRequest (lyhyesti XHR). XHR-pyynnöt ovat JavaScriptillä lähetettyjä HTTP-pyyntöjä. Perinteisesti asiakassovellus lähetti HTTP-pyyntöjä vain käyttäjän navigoidessa sivujen välillä tai sovelluksen uudelleen latauksilla. XHR mahdollisti kuitenkin pyyntöjen lähettämisen milloin vain ilman, että sovelluksen sisältöä tarvitsisi ladata uudestaan. (Haverbeke 2011.) Tiedonsiirto tapahtuu taustalla asiakkaan käyttäessä sovellusta, eikä käyttäjä välttämättä huomaa pyyntöjä, koska ne eivät keskeytä sovelluksen käyttöä. SPA-sovellukset on rakennettu näiden taustalla toimivien HTTP-pyyntöjen ympärille. Alkuun XHR -pyynnöt oli suunniteltu lähettämään XML-dataa, mutta nykyään sitä käytetään myös JSON:in ja tekstin siirtoon (Haverbeke 2011).

Ohjelmoijien työtä helpottamaan on rakennettu erilaisia valmiita sovelluskehyksiä (engl. framework). Sovelluskehykset sisältävät valmiita sovelluksenosia (funktioita ja moduuleja), jotka nopeuttavat sovelluskehitystä. Hyödyntämällä sovelluskehyksiä ohjelmoijien ei tarvitse itse rakentaa kaikkia sovelluksen osia. (Baker 2009.) Sovelluskehykset voivat tarjota esimerkiksi navigaation, rakennuslogiikan tai tapoja hallita sisältöä. Sovelluskehyksiä on todella monia, mutta suosituimmat JavaScript-kehykset tällä hetkellä ovat React.js, Angular.js 2 ja Vue.js (Smith 2018). Lähes kaikki SPA-sovellukset käyttävät jotain sovelluskehystä, mutta sovelluskehyksen käyttö ei ole välttämätöntä. Tässä työssä tarkastelen SPA-sovelluksia ja niiden saavutettavuusongelmia kuitenkin Reactin kautta React-ohjelmointi taustani takia.

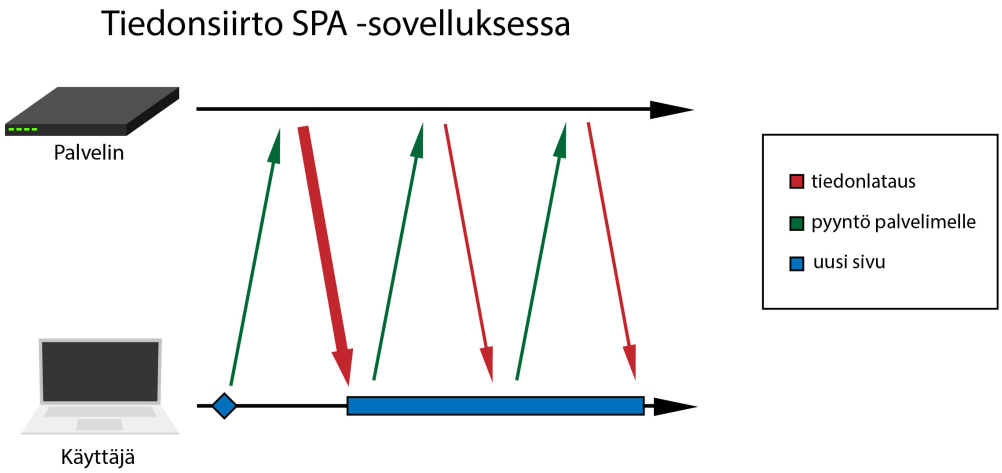
3 Mikä on SPA-sovellus?

Internetin kehittyminen ja uudet verkko ohjelmointirajapinnat (engl. Web API) ovat olleet avaintekijänä SPA-sovellusten suosion kasvussa. Uudet ohjelmointirajapinnat mahdollistivat selaimen toiminnallisuuden hyödyntämisen, vaikka SPA-sovellukset eivät toimikaan perinteisellä tavalla. SPA-sovellus eli yhden sivun -sovellus (engl. Single-page application) on nimensä mukaan yhteen HTML-sivuun rakennettu sovellus. Sovelluksen logiikka ladataan ensimmäisellä pyynnöllä palvelimelta, ja tämän jälkeen JavaScript rakentaa uudet näkymät aina saman, ensimmäisellä kerralla ladatun HTML-dokumentin sisään. (Sherman 2018.) Sovellus hyödyntää XHR-pyyntöjä ja tarvittaessa pyytää palvelimelta lisätietoja uuden näkymän rakentamiseen.



Kuvio 1. Perinteisen verkkosovelluksen kommunikaatio palvelimen kanssa

Perinteisesti sivujen rakentamis- ja toimintalogiikka on ollut aina palvelimen puolella. Asiakkaan pyynnöllä palvelin on rakentanut sivun ja lähettänyt valmiin sivun asiakkaalle. Näin asiakas ei ole ollut tietoinen käyttöliittymän rakennuslogiikasta ja on joutunut aina pyytämään palvelimelta seuraavaa sivua eikä ole voinut hyödyntää sivuilla toistuvia elementtejä (kuvio 1.). SPA-sovelluksissa tämä logiikka ladataan ensimmäisellä HTTP-pyynnöllä asiakkaan puolelle (Neoteric 2016). Tästä seuraa tavallista suurempi ensimmäinen lataus. Tämän latauksen jälkeen JavaScript rakentaa sivulle ensimmäisen näkymän. Koska JavaScript on vastuussa sivun rakentamisesta, on sillä myös tiedossa näkymässä käytetyt elementit, ja se voi käyttää niitä tarvittaessa uudestaan seuraavissa näkymissä. Näin asiakassovelluksen ei tarvitse pyytää jo ladattua sisältöä uudestaan. Kommunikaatio palvelimen ja asiakkaan välillä vähenee, ja tästä seuraa sovelluksen nopea vasteaika, koska kommunikaatio hitaan internetyhteyden yli on minimoitu.



Kuvio 2. SPA-sovelluksen kommunikaatio palvelimen kanssa

Sovelluslogiikan latauksen jälkeen SPA-sovellus hyödyntää XHR-pyyntöjä, jotta se voi tarvittaessa keskustella palvelimen kanssa taustalla. XHR-pyyntöjen avulla sovellus ei joudu uudelleen lataamaan sivua, vaikka tarvitsisi lisää tietoa seuraavaan näkymään. (Itsnat 2015.) Usein ensimmäisellä latauksella sovellus ei kuitenkaan lataa kaikkien näkymien sisältöä, koska sitä voi olla todella paljon. Esimerkiksi sosiaalisen median käyttäjäprofiilinäkymiä voi olla miljoonia. Näkymiin tarvittava sisältö ladataan käyttäjän navigoidessa niihin. JavaScript lähettää tarvittavaa sisältöä varten XHR-pyynnön palvelimelle, joka hakee tiedot taustalla (kuvio 2.). Tämän jälkeen JavaScript hyödyntää palvelimelta saatua vastausta uuden näkymän rakennuksessa. Perinteisellä navigaatiolla käyttäjän painaessa navigaatiolinkkiä selain aloittaa uuden sivun latauksen ja samalla poistaa vanhan sivun. Tällöin selaimen toiminta voi hidastua tai sivu saattaa muuttua kokonaan valkoiseksi uuden sivun latautuessa. Tämä hidastuminen ja hetken valkoinen välähdys sivujen välillä huonontaa käyttäjän käyttökokemusta.

Perinteisissä verkkosovelluksissa sovelluksen käyttö aina keskeytyy käyttäjän navigoidessa sivulta toiselle, koska uusi sivu korvaa vanhan ja selaimen on uudelleen rakennettava käyttöliittymä (kuvio 1.). Samalla selain joutuu lataamaan jo olemassa olevaa tietoa ja tiedonsiirto navigoidessa on aina suuri. Hyvällä yhteydellä tiedonsiirrosta syntyvän tauon pituus on alle sekunnin, mutta huonolla yhteydellä lataustauko voi olla jopa kymmeniä sekunteja. Käyttäjät todennäköisesti hylkäävät sivun vaikka käyttöliittymä olisi hyvä, jos sivujen välissä liikkuminen tuntuu hankalalta. Vuonna 2009 tehdyn tutkimuksen mukaan 47 % käyttäjistä olettaa sivun latautuvan maksimissaan kahdessa sekunnissa

ja 40 % käyttäjistä poistuu sivulta, jos latauksessa kestää yli kolme sekuntia (Akamai 2009).

SPA-sovelluksissa tämä ongelma ratkeaa, koska uusi näkymä rakennetaan JavaScriptillä saman jo ladatun HTML-sivun sisään. JavaScript pystyy myös hyödyntämään jo ladattuja elementtejä, eikä niitä tarvitse ladata uudelleen. JavaScript pystyy reagoimaan nopeasti käyttäjän pyyntöihin, ja näin sivu tuntuu nopealta. SPA-sovelluksilla saa siis aikaan paremman käyttökokemuksen poikkeamalla perinteisestä verkkosivujen ja sovellusten toiminnasta.

4 Saavutettavuus

Saavutettavuus tarkoittaa kohteen tai verkkopalvelun helppoa lähestyttävyyttä kaikille ihmisille (Celia 2018). Esteettömyys koskee fyysistä maailmaa, kun taas saavutettavuus liittyy digitaalisiin palveluihin. Saavutettavuus koskee erityisesti sokeita ja heikkonäköisiä, liikuntavammaisia ja kognitiivisista ongelmista kärsiviä, mutta saavutettavuuden hyödyt eivät rajoitu ainoastaan näihin ryhmiin vaan niistä hyötyvät kaikki, koska ne parantavat myös yleistä käytettävyyttä. Saavutettavuus on otettava huomioon sovelluksen kehitysvaiheessa, jotta saavutettavuusongelmat pystytään välttämään. (Valtiovarainministeriö 2018.)

Palveluiden digitalisoituessa saavutettavuuden rooli on tullut yhä tärkeämmäksi. Vuonna 2016 tuli voimaan Euroopan parlamentin ja neuvoston saavutettavuusdirektiivi. Direktiivi koskee julkisen sektorin elinten verkkosivustojen ja mobiilisovellusten saavutettavuutta, ja sillä halutaan turvata kaikkien mahdollisuuden toimia tasavertaisesti digitaalisessa yhteiskunnassa. Saavutettavuusdirektiivissä säädetään saavutettavuuden minimitaso julkisen hallinnon verkkopalveluille. (Valtiovarainministeriö 2018.)

> Saavutettavuusvaatimukset koskevat seuraavia: valtion viranomaiset ja liikelaitokset (esim. ministeriöt ja virastot), kunnalliset viranomaiset mukaan lukien liikelaitokset ja koulut, osa eduskunnan virastoista ja Tasavallan presidentin kanslia, julkisoikeudelliset yhdistykset siltä osin kun ne hoitavat julkista hallintotehtävää, itsenäiset julkisoikeudelliset laitokset (esim. Kela), yliopistot, yliopistolain (558/2009) pykälässä tarkoitetut, ammattikorkeakoulut, ammattikorkeakoululaissa (932/2014) säädetyt, ortodoksinen kirkko ja ortodoksiset seurakunnat, lakisääteisiä tehtäviä hoitavat yhtiöt (esim. katsastuskonttorit, eläke- ja tapaturmavakuutusyhtiöt). (Valtiovarainministeriö 2018.)

Saavutettavuusvaatimukset koskevat näiden lisäksi myös muun muassa vesi- ja energiahuollon, liikenteen ja postipalveluiden alalla toimivien julkisten yritysten digitaalisia palveluita sekä joitain vakuutusyhtiöitä ja kolmannen sektorin organisaatioita. Saavutettavuusvaatimukset astuvat voimaan portaittain 23.9.2019, jolloin 23.9.2018 ja sen jälkeen julkaistujen verkkosivustojen pitää olla vaatimusten mukaisia. Ennen 23.9.2018 julkaistujen verkkosivustojen pitää olla saavutettavuusvaatimusten mukaisia 23.9.2020 ja mobiilisovellusten 23.6.2021. (Valtiovarainministeriö 2018.)

4.1 WCAG 2.0-Standardi

Saavutettavuusdirektiivi pohjautuu WCAG 2.0-standardiin. WCAG 2.0 on tekninen ohjeistus saavutettavuuden toteuttamiseen verkkopalveluissa. WCAG sisältää huomioita, joita ohjelmoijan tulisi ottaa huomioon sovellusta kehittäessään. Standardi ei kuitenkaan tarjoa valmiita ratkaisuja vaan yleiset ohjeet, joita voi noudattaa teknologiasta riippumatta. WCAG 2.0-standardissa on kolme eri saavutettavuus tasoa: A, AA ja AAA, joista AAA-taso on tiukin ja A-taso löyhin. Saavutettavuusdirektiivin vaatimukset pohjautuvat WCAG 2.0 -standardin AA-tasoon. (W3C 2008.)

WCAG 2.0 on jaettu neljään alakategoriaan: havaittava (engl. perceivable), käytettävä (engl. operable), ymmärrettävä (engl. understandable), lujatekoinen (engl. robust). Kategoriassa ”havaittava” käsitellään tietojen esittämistä tavalla, jolla käyttäjät voivat sen havaita. Esimerkiksi sokean on mahdotonta ymmärtää kuvan sisältöä, jos se ei sisällä kuvatekstiä. Kategoriassa annetaan ohjeeksi vaihtoehtoisen tekstin tarjoamista sisällölle, joka ei ole tekstiä, kuvatekstien tarjoamista videoihin ja muuhun multimediaan ja sisällön näkemisen ja kuulemisen helpottamista. Tämän lisäksi ohjelmoijia ja suunnittelijoita kehotetaan luomaan sisältö niin, että sen voi esittää eri tavoilla, ilman että sisältö menettää merkityksensä. (W3C 2008.)

Kategoriassa ”käytettävä” sen sijaan käsitellään käyttöliittymän käyttöön liittyviä asioita. Ohjeessa sanotaan, että kaiken sivun sisällön pitää olla käytettävissä näppäimistöltä ja käyttäjillä pitää olla tarpeeksi aikaa lukea ja käyttää sisältöä. Ohjeessa kehotettaan myös olemaan käyttämättä sisältöä käyttöliittymässä, joka saattaisi aiheuttaa kohtauksia. Kehittäjien tulisi myös tarjota käyttäjille mahdollisuus etsiä sisältö ja helpottaa heidän navigointiaan. Näiden lisäksi pitäisi myös pitää mielessä, että kaikki käyttäjät eivät välttämättä voi käyttää näppäimistöä, ja suunnittelussa pitäisi helpottaa myös heidän sovelluksen käyttöään.

Ymmärrettävän kategorian sisään kuuluu käyttöliittymän ja sivun sisällön ymmärrettävyys. Kategoria sisältää alakohtia, jotka käsittelevät tekstin luettavuutta ja ymmärrettävyyttä, sisällön näkymistä ja käyttäytymistä oletettavalla tavalla sekä käyttäjien auttamista ongelmien välttämisessä kuin myös ongelmien ratkaisussa. Viimeisessä kategoriassa ”lujatekoinen” kehotetaan kehittäjiä maksimoimaan sovelluksen yhteensopivuus eri käyttäjäagenttien kanssa ja tarjotaan siihen huomioitavia asioita.

4.2 Ruudunlukijat

Saavutettavuuden mahdollistamiseksi sokeille ja heikkonäköisille on kehitetty ruudunlukijat, jotta heillä voisi olla tapa käyttää tietokonetta. Sokeat ja heikkonäköiset käyttävät usein näppäimistöä kontrolloidakseen ruudunlukijaa, mutta ruudunlukijaa voi käyttää myös muilla apulaitteilla. Ruudunlukija lukee näytöllä olevaa sisältöä käyttäjän liikkuessa sivulla näppäimistöä käyttäen. (American Foundation for the Blind 2018.) Verkkosivuilla linkkien välillä liikkumiseen käytetään usein tabulaattoria. Ruudunlukija lukee sisällön kohdasta, jonka päällä käyttäjä on eli jossa käyttäjällä on fokus. Fokusta voi siirtää myös JavaScriptin avulla, ja näin ohjelmoija voi kontrolloida ruudunlukijan lukemaa sisältöä.

HTML -kieleen on rakennettu myös monia ruudunlukijoiden käyttäjiä helpottavia lisäominaisuuksia, joista monet kuuluvat WAI-ARIAan. WAI-ARIA attribuuteilla kehittäjät voivat kontrolloida ruudunlukijan toimintaa vielä tarkemmin, esimerkiksi rajoittamalla luettavaa sisältöä tai käskemällä ruudunlukijaa lukemaan, jos sisältö muuttuu. WAI-ARIA auttaa erityisesti dynaamisen sisällön ja monimutkaisten käyttöliittymien kanssa, jotka on kehitetty JavaScriptiä, HTML:ää ja XHR-pyyntöjä käyttäen. WAI-ARIAa suositellaan kuitenkin hyödyntävän, vaikka sivu ei olisikaan dynaaminen. WAI-ARIAsta eivät hyödy ainoastaan ruudunlukijoiden käyttäjät, vaan ne parantavat myös sivun yleistä saavutettavuutta. (Web Accessibility Initiative 2018a.)

Ruudunlukijoita on monia, mutta yleisimmät ovat JAWS, NVDA ja VoiceOver. WebAIMin järjestämän ruudunlukijakyselyn mukaan noin 46,6 % kaikista ruudunlukijoiden käyttäjistä käyttävät JAWSia, 31,9 % NVDA:ta ja 11,7 % VoiceOveria (WebAIM 2018). Ruudunlukijoissa on jotain eroja, eivätkä ne käyttäydy kaikissa tilanteissa samalla tavalla. Olen huomannut, että kehittäjän kannattaa kokeilla itse, toimiiko sovellukseen implementoitu ratkaisu kaikilla ruudunlukijoilla. Toteutan esimerkkisovelluksen tässä opinnäytetyössä hyödyntäen VoiceOveria, koska JAWS ja NVDA ovat saatavilla ainoastaan Windows käyttöjärjestelmällä.

5 SPA-sovellusten saavutettavuus

Saavutettavuusongelmien etsiminen aloitetaan vertailemalla perinteistä verkkosovellusta SPA-sovellukseen. Näin rajataan ongelmat yleisistä saavutettavuusongelmista ainoastaan SPA-sovelluksen arkkitehtuurin takia syntyviin ongelmiin. Eroja tarkastellessa pidetään mielessä myös ruudunlukijan käyttäjät, jotta voidaan rajata pois muut mahdolliset ongelmat saavutettavuudessa. Saavutettavuusongelmien löytämisessä ja varmentamisessa käytetään apuna VoiceOver-ruudunlukijaa. Mahdollisia ongelmakohtia peilataan WCAG 2.0-standardin AA-tasoa vasten, jotta ongelmat pysyvät huomattavina ja tärkeinä. Tavoitteena on löytää ongelmia, jotka aiheuttavat toimenpiteitä saavutettavuuden mahdollistamisessa julkisen hallinnon verkkopalveluissa ja sovelluksissa.

SPA-sovelluksissa sivun rakentamis- ja toimintalogiikka ladataan ensimmäisellä pyynnöllä palvelimelta. Tämän jälkeen sovellus kontrolloi itse sivun sisältöä rakentaen sen uudelleen käyttäjän navigoidessa sivulla. Navigaatiolinkkien toiminta on muutettu perinteisestä uuden sivun latauksesta JavaScript-funktion kutsuksi. Perinteisesti navigaatiolinkin painaminen lähettää uuden HTTP-pyynnön palvelimelle, selain vastaanottaa uuden HTML-sivun ja korvaa vanhan sivun tällä sivulla. Perinteinen linkki pakottaisi sivun uudelleen latautumaan, kun taas JavaScript funktiolla voidaan kontrolloida tarkemmin sivun toimintaa. Tämä perinteinen toiminta on estetty JavaScript-funktiolla “event.preventDefault()”. Funktio käskee selainta unohtamaan tavallisen tavan toimia, ja kehittäjät voivat itse määrittää toiminnan (MDN web docs 2018b).

Perinteisesti linkin painaminen aiheuttaa monia asioita. Sovellus lähettää HTTP -pyynnön palvelimelle, ja selain alkaa rakentamaan uutta sivua. Tämän jälkeen selain vastaanottaa uuden HTML -sivun palvelimelta ja käsittelee vastaanotetun sivun ja rakentaa siitä näkymän käyttäjälle. Samalla selain lähettää uudet HTTP -pyynnöt jokaiselle sivun kuvalle, scripti- ja css-tiedostolle ja lisää nämä sivulle saatuaan vastaukset. (Ostrovsky 2010.) Vanha sivu lisätään myös selaimen sivuhistoriaan. Uuden sivun latauksen myötä ruudunlukija huomaa uuden sivun latautuneen ja lukee tämän sivun otsikon. Selain siirtää samalla myös fokuksen uuden sivun alkuun, jotta ruudunlukijan käyttäjä voi aloittaa sivun läpikäynnin.

SPA-sovelluksissa linkin painaminen toimii toisella tavalla. Ensimmäisellä latauksella on jo ladattu alasivujen rakenteet ja ulkoasut, joten niitä ei tarvitse uudelleen ladata. Suosittua React Router -kirjastoa käytettäessä navigaatio linkin painaminen käskee Reactia

rakentamaan seuraavan näkymän (React training 2018). Tämän jälkeen React mahdollisesti lähettää taustalla XHR-pyynnön, jolla se pyytää palvelimelta lisää tietoa sivulle, jos lisää tietoa tarvitaan. React käyttää jo ladattuja komponentteja uuden näkymän rakentamisessa ja täydentää näitä elementtejä lisätiedolla, jonka se saa palvelimelta XHR-pyynnöillä. Vanha näkymä lisätään sivu historiaan käyttäen selaimeen rakennettua sivuhistoria-rajapintaa, jotta käyttäjä voi palata takaisin aikaisemmalle sivulle selaimen takaisin-nappia painamalla. Fokus ei automaattisesti siirry sivun alkuun, koska kyseessä ei ole sivun uudelleen lataus.

Osa selaimen toiminnasta jää SPA-sovelluksissa kehittäjien toteutettavaksi. Kehittäjien pitää ymmärtää SPA-sovelluksen toiminta logiikka ja kuinka se on muuttunut perinteisestä sovelluksesta, jotta he voisivat ymmärtää uudet ominaisuudet, jotka heidän tulee toteuttaa. Mielestäni liian usein hypätään käyttämään uusia teknologiota tuotannossa ilman että kunnolla ymmärretään niitä ja niiden tuomia vastuita. Tämä on kuitenkin ymmärrettävää, koska asiakkaanpuolen kehitysympäristö on jatkuvan muutoksen alla. Samalla uudet teknologiat vaativat aikaa kypsyä, ennen kuin ovat valmiita tuotantoon.

Perinteisesti selain on hoitanut fokuksen siirron ja samalla ruudunlukija on myös tiedostanut uuden sivun latautuneen. Ruudunlukijat eivät kuitenkaan huomaa SPA-sovellusten näkymän vaihdosta, koska HTML-sivu on pysynyt samana ja vain sisältö on muuttunut (DeLuca 2017). Fokus jää paikoilleen sisällön muutoksen aiheuttaneeseen linkkiin, vaikka se olisi hävinnyt sivulta ja ruudunlukija jatkaa saman linkin sisällön lukemista. Ruudunlukijat eivät myöskään lue sivun otsikkoa, vaikka se olisi JavaScriptin avulla muutettu. Selain ja ruudunlukija eivät ole tietoisia onko sivulla muuttunut sisältö vain joku JavaScript efekti vai onko sivu vaihtunut. Mielestäni tästä voi seurata ruudunlukijoiden käyttäjille epäilys linkin toimivuudesta tai tietämättömyys, koska he eivät tiedä onko uusi sisältö latautunut tai mistä se löytyy.

WCAG 2.0 -standardin AA-tasossa ei suoraan puhuta selaimen navigaatiosta, vaan standardi on kirjoitettu kattamaan monia teknologiota. Standardi rakentuu neljästä eri kategoriasta havaittava, käytettävä, ymmärrettävä ja lujatekoinen. SPA-sovellusten linkkien käyttäytymisen haasteet osuvat mielestäni parhaiten käytettävä ja ymmärrettävä kategorioihin. Kategoriassa ”käytettävä” käsitellään käyttöliittymän käyttöä ja se sisältää ongelmaan liittyviä alakohtia, jotka käsittelevät näppäimistöllä liikkumista ja navigointia sivulla. Kategoriassa ”ymmärrettävä” taas käsitellään sisällön ymmärrettävyyttä ja yhtenä alakohtana on sisällön toimivuus oletettavalla tavalla. ”Ymmärrettävä” -kategoria

sisältää kuitenkin enemmän ohjeita sisällön johdonmukaisuudesta. Sen tarkoituksena on varmistaa, että käyttäjät voivat helposti muodostaa mielessään kuvan sivun rakenteesta, joten se ei liity tähän sivun navigaatio-ongelmaan.

Ongelma osuu hyvin standardin toiseen kategoriaan ”käytettävä”, koska se käsittelee liikkumista käyttöliittymässä. Ensimmäisessä kohdassa puhutaan näppäimistön käytöstä liikkumiseen, mutta SPA-sovellusten ei kuitenkaan ole ongelmana näppäimistön käyttö, vaan fokuksen siirtyminen ja käyttäjän informointi. Kategorian neljännessä kohdassa puhutaan näkymissä liikkumisesta, sekä näkymien välillä liikkumisesta. Siellä käsitellään muun muassa sivujen otsikointia, fokusta sekä ymmärrettävää navigointia. Mielestäni kohta 2.4.3 osuu parhaiten SPA-sovellusten ongelmiin, koska siinä käsitellään fokuksen loogisen järjestyksen tärkeyttä ja fokuksen liikkumista sivuilla. (Web Accessibility Initiative 2018b.) ”Käyttäjien navigoidessa sisällön läpi, he kohtaavat tietoja järjestyksessä, joka on sopusoinnussa sisällön merkityksen kanssa” (Web Accessibility Initiative 2018c).

Kohdan 2.4.3 alla on myös tekniikoita, joilla sen kriteerit saavutetaan. Mielestäni tekniikka SCR26, ”Dynaamisen sisällön lisääminen Document Object Modeliin heti lisäyksen laukaisseen elementin perään”, kohdistuu parhaiten SPA-sovellusten ongelmiin. Kohdassa käsitellään dynaamista sisällön lisäystä HTML-sivun sisälle, ja kuinka tämä sisältö pitäisi heti olla lisäyksen aiheuttaneen elementin perässä (Web Accessibility Initiative 2018d). SPA-sovelluksissa navigaatio linkin painaminen aiheuttaa juuri tällaisen dynaamisen sisällön lisäämisen sivulle. Sovellus rakentaa uuden näkymän ja korvaa olemassa olevan näkymän muuttuneella näkymällä. SPA-sovelluksissa tosin navigaatio linkin painaminen aiheuttaa usein koko sisällön poiston ja uuden sisällön lisäyksen, eikä tästä syystä ole mahdollista lisätä sisältöä vain muutoksen aiheuttaneen elementin perään. Ratkaisu tähän voisi olla, että fokus siirretään muuttuneen alueen eteen, jotta käyttäjä vois suoraan jatkaa uuden sivun tarkastelua. Toisaalta tällä tavoin poikettaisiin perinteisestä tavasta liikkua sivujen läpi, jossa ruudunlukijankäyttäjät ovat tottuneet hyppäämään navigaation yli.

Kohdassa puhutaan myös siitä, kuinka fokuksen tulisi säilyä lisäyksen laukaisseessa elementissä. SPA-sovelluksissa tämä muutoksen laukaissut elementti saatetaan myös poistaa sivulta muutoksen aikana, eikä fokus silloin pysy siinä. SCR26 onkin mielestäni ajateltu käsittelemään vain pieniä lisäyksiä sivuille, kuten alas putoavia navigaatiota eikä siinä ole otettu huomioon SPA-sovellusten suuria poisto ja lisäys operaatioita. Tämä voi

johtua siitä, että WCAG 2.0 -standardi on julkaistu vuonna 2008, kun taas SPA-sovellusten suosio kasvoi vasta HTML 5:n julkaisun jälkeen. Huomiona SCR26:ssa on kuitenkin, että tekniikka on suunniteltu synkronisille tapahtumille. Asynkronisiin tapahtumiin, kuten esimerkiksi XHR -pyyntöihin, pitää olla lisäksi vielä tapa ilmoittaa avustavalle teknologialle, että sisältö on lisätty (Web Accessibility Initiative 2018d).

WCAG 2.0 -standardin AA-tason saavuttamiseksi ja saavutettavuusdirektiivin vaatimuksien saavuttamiseksi on tämä saavutettavuusongelma ratkaistava, koska ”käytettävä” -kategorian kohta 2.4.3 kuuluu löyhempään A -tasoon. Seuraavaksi tutkin jo olemassa olevia SPA-sovelluksia ja katson, löytyykö niistä tätä ongelmaa.

Valitsin tarkasteltavaksi kaksi suurta verkkopalvelua Facebook.com ja Suomi.fi, jotka kummatkin ovat SPA-sovelluksia. SPA-sovelluksen tunnistin niiden ominaispiirteistä, sivua ei ladata uudestaan navigaatiolla. Tosin sovelluksissa oli myös alasivuja, jotka eivät kuuluneet SPA-sovellukseen, vaan olivat erillisiä sovelluksia ja näille sivuille navigoidessa sivu latautuu uudelleen. Palveluita tarkastellessa seuraan ruudunlukijan käyttäytymistä navigoidessa SPA-sovelluksen näkymien välillä. Yritän myös huomioida sivulle rakennettuja muita ominaisuuksia saavutettavuuden mahdollistamiseksi.

5.1 Suomi.fin tarkastelu

”Suomi.fi-verkkopalvelusta löytyy tietoa ja palveluja eri tilanteisiin kansalaisille, yrityksille ja yrityksen perustamista harkitseville” (Suomi.fi 2018). Päätin ottaa Suomi.fi verkkosivun mukaan tähän tarkasteluun, koska se on tärkeä julkisen hallinnon palvelu, jonka tulee olla saavutettavuusdirektiivin vaatimusten mukainen. Suomi.fi-sivulla saavutettavuus on otettu hyvin huomioon ja ongelma näytti siellä olevan ratkaistu. Tarkastelin navigaatiota päänavigaatiosta löytyvien sivujen välillä. Valtuudet- ja tunnistautumis-sivut eivät kuuluneet SPA-sovellukseen, joten ne jätin ulos tarkastelusta.

Suomi.fi-sivulla fokus palaa navigaatioilla sivun alkuun ja NVDA-ruudunlukija lukee sivulle määritetyn otsikon. Näin sovelluksessa matkitaan perinteistä selaimen toimintaa ja ruudunlukijan käyttäjät eivät välttämättä huomaa sovelluksessa eroa perinteisiin sovelluksiin. Perinteiseen tapaan fokus siirtyy linkistä sivun alkuun, eikä muuttuneeseen sisältöön. Joissain alasivuilla kuten ”Ohjeet ja tuki” -sivulla huomasin, että sivun otsikko ei muuttunut ja ruudunlukija luki vielä vanhan sivun otsikon. Kehittäjät ovat olleet tietoisia SPA-sovelluksen ongelmista ja ottaneet ne huomioon sovellusta kehittäessä.

5.2 Facebook.comin tarkastelu

Facebook.com on suosittu sosiaalinen media, joka perustettiin vuonna 2004. Facebookin mukaan heidän palveluitaan käytti syyskuussa 2018 2,27 miljardia ihmistä (Facebook 2018). Kaikki näistä eivät kuitenkaan ole Facebook.comin käyttäjiä. Facebook.com-sivuston halusin ottaa mukaan tähän tarkasteluun, koska se on suosittu sosiaalinen media ja koska React.jsän taustalla on Facebook. Facebook.com sisältää joitain SPA-sovelluksia, mutta tässä työssä tarkastellaan kirjautumisen jälkeistä etusivua ja etusivulta liikkumista profiilisivulle ja ”tapahtumat” -sivulle.

Facebook on rakentanut sovellukseensa apuvälineitä, joilla on tarkoitus auttaa ruudunlukijoiden käyttäjiä navigoimisessa ja sovelluksen käyttämisessä. Sivulle tultaessa ensimmäisellä tabulaattorin painamisella aukeaa navigointiapuri (engl. navigation assistant), josta käyttäjä voi siirtyä sivulla näkyvään sisältöön tai muihin ala sivuihin. Navigointiapurin sisältä löytyy myös ohjeet käyttöliittymän pikanäppäinten käyttöön.

Käyttäen navigointiapurin työkaluja navigointiin huomasin, että pääsivulta siirtyminen profiilisivulle ei siirrä fokusta sivun alkuun. Navigointilinkki katoaa ja fokus siirtyy sivun loppuun. Seuraavalla tabulaattorin painamisella fokus siirtyy sivun ulkopuolelle. Samaan aikaan ruudunlukija ei ilmoita uuden sivun latautumisesta ja käyttäjä on epävarma, onko navigointi tapahtunut vai lataako sivu sisältöä vielä. Facebook on voinut rakentaa taustalle jonkun muun metodin, jolla käyttäjät voivat navigoida paremmin, mutta mielestäni perinteinen navigointi pitäisi myös mahdollistaa. Facebook.comia ensimmäistä kertaa käyttävät eivät kuitenkaan tunne sovelluksen omia työkaluja.

6 Ratkaisut saavutettavuusongelmiin

Ensimmäinen tapa ongelmien ratkaisemiseksi olisi, että SPA-sovelluksissa voisi matkia selaimen tavallista toimintaa, kuten suomi.fi palvelussa tehtiin. Ruudunlukijoiden käyttäjät eivät välttämättä huomaisi sovelluksen eroa perinteiseen verkkosovellukseen. Perinteisesti sivun vaihdoksella ruudunlukija on lukenut aina uuden HTML-sivun otsikon ja fokus on siirtynyt sivun alkuun. Jos SPA-sovelluksissa matkittaisiin tätä korvattua selaimen toimintaa voisi käyttökokemus ruudunlukijan kanssa olla samanlainen kuin aiemmin perinteisissä sovelluksissa.

Toisena mahdollisuutena on myös tavallisesta poikkeava navigointi mahdollisuus. Reactilla sovellus päivittää vain näkymässä muuttuneet osat ja jättää päivittämättä näkymissä toistuvat elementit. Tämä mahdollistaisi fokuksen siirron suoraan sivulla muuttuneeseen sisältöön, eikä käyttäjän tarvitsisi liikkua sivuilla toistuvan sisällön yli päästäkseen muuttuneeseen osaan. Tämä ratkaisu olisi enemmän WCAG 2.0 -standardin SCR26-tekniikan mukainen, koska muuttunut sisältö löytyy heti seuraavaksi tabulaattoria painettaessa. Mielestäni tämä voi kuitenkin aiheuttaa ongelmia, koska näin poiketaan ruudunlukijan perinteisestä toiminnasta. Suurissa sisältö muutoksissa ruudunlukijoiden käyttäjien voi olla vaikea hahmottaa sivun sisältöä tai navigaation sijaintia, jos osa sisällöstä hypätään yli sivun vaihdoksella, mutta samalla SPA-sovellukset avaavat tilaisuuden kyseenalaistaa perinteisen tavan navigoida sivuilla. Esimerkkisovelluksen toteutustavaksi valitsin kuitenkin perinteisen selaimen navigaation matkimisen, koska se on käyttäjille jo ennestään tuttu.

6.1 Esimerkkisovellus

Esimerkkisovelluksen toteutan Reactilla ja hyödyntäen React Router -lisäosaa. En kuitenkaan syvenny tarkasti Reactin käyttöön, enkä muihin taustalla toimiviin teknologiohin, mutta selitän jotain oleellisia asioita. Sovellus rakennetaan ja testataan käyttäen VoiceOveria ja Google Chrome -selainta, koska VoiceOver oli suosituin Mac-käyttöjärjestelmällä toimiva ruudunlukija. Sovelluksen rakennan Codesandbox.io palveluun, jotta sovelluksen jako ja lähdekoodin lukeminen olisi mahdollisimman helppoa.

Toteutuksen tavoitteena on ratkaista SPA-sovelluksen navigaation saavutettavuusongelmat tarjoamalla yksinkertainen ratkaisu, jota voi hyödyntää myös muissa projekteissa. React.js:n luonteen mukaan tämä toteutetaan tekemällä komponentti, joka on itsenäinen ja uudelleenkäytettävä. Näin sen voi ottaa käyttöön myös muissa ympäristöissä. Toteutus ratkaisee ongelman matkimalla perinteistä selaimen toimintaa. Sovellus hyödyntää HTML:n WAI-ARIA -tageja ja JavaScriptiin sisäänrakennettuja ominaisuuksia. Sovellus siis ilmoittaa käyttäjälle uuden sivun latautumisesta ja siirtää fokuksen sivun alkuun. Sovellus saattaa poiketa saavutettavuus standardeista muilla osa-alueilla, mutta sovellus keskittyy ratkaisemaan SPA-sovelluksissa olevan uniikin navigointiongelman.

React.js on JavaScriptillä kehitetty ohjelmointikehys, joka auttaa kehittäjiä luomaan interaktiivisia käyttöliittymiä. React julkaistiin vuonna 2013 ja on alun perin lähtöisin Facebookista, mutta nyt sitä kehittävät Facebookin lisäksi monet kehittäjät ja yritykset. React

on kehitetty käyttöliittymien ja näkymien rakentamista varten, kun taas muut teknologiat vastaavat palvelin puolesta. (React 2018a.) Reactia kuitenkaan harvoin käytetään yksin näkymien kehityksessä, vaan sen kanssa käytetään erilaisia lisäosia, jotka kaikki auttavat ohjelmoijia kehittämään laadukkaita käyttöliittymiä. Esimerkkisovelluksessa käytetään muun muuassa React Router -lisäosaa, joka tarjoaa kehittäjälle työkaluja yksinkertaisen navigaation toteuttamiseen.

Koodi 1. JSX-komponentti

```
<HelloWorld name=”Aaron” />
```

Reactin avulla kehittäjät voivat luoda uudelleen käytettäviä komponentteja. JSX:n ansiosta nämä komponentit muistuttavat HTML -tageja (koodi 1.). JSX käytetään yleisesti vain Reactin kanssa ja se auttaa React-elementtien luonnissa. React-komponentteja voidaan luoda jatkamalla React.Component-mallia (koodi 2.). Sen avulla saadaan hyödynnettyä Reactiin jo rakennettuja ominaisuuksia. Uuden komponentin sisällä pystyy myös käyttämään komponentille määritettyjä attribuutteja. Esimerkissä koodi 2. tulostetaan attribuutti ’name’.

Koodi 2. React-komponentin luonti

```
import React from ‘react’
class HelloWorld extends React.Component {
  render() {
    return (
      <div>Hello {this.props.name}</div>
    )
  }
}
```

6.2 Sovelluksen toteutus

Toteutuksessa hyödynnän aria-live -tagia ilmoittaakseni ruudunlukijalle sivun vaihdosta. Aria-live -tagi lukee elementin sisällön, jos se muuttuu (MDN web docs 2018a). React router toimii applikaation reitittimenä. React routeria varten pitää sovelluksessa käyttää Router-komponenttia. Router-komponentin sisällä määritetään kaikki sovelluksen alasivut käyttäen Route-komponenttia. (Koodi 3.) Komponentit saa react-router-dom -moduulista.

Koodi 3. React routerin perusasetukset

```
<Router>
  <Route exact path=”/” render={() => (
    <HomePage />
  )} />
  <Route path=”/otherpage” render={() => (
    <OtherPage />
  )} />
</Router>
```

Sovellukseen kehitettään logiikka, joka tunnistaa sivun vaihdoksen. Sivun vaihdoksen voi tunnistaa osoitepolun muuttumisesta, React routerin reitin muuttumisesta tai uuden sisällön prosessoinnista. Päätin toteuttaa sen tässä esimerkissä seuraamalla React routerin reitinvaihtoa, koska se oli helppo implementoida. Osoite polun muuttumisen seuraaminenkin olisi ollut hyvä vaihtoehto, koska se usein muuttuu sivunvaihdoksella.

Ilmoituksen toteutan tekemällä komponentin, joka sivunvaihdoksella vaihtaa aria-live -elementistä sisällön ilmoitettavaan tekstiin. Luettava teksti on usein myös sivun otsikko, joten muutan sivun otsikon samalla siihen. Komponentti myös laukaisee fokuksen siirron.

Fokuksen siirto toteutetaan käyttäen reactin ref-ominaisuutta. Refin avulla JavaScript voi siirtää fokuksen tiettyyn komponenttiin käyttöliittymässä (React 2018b). Ref sisältää referenssin sivulla näytettyyn elementtiin. Fokusoitavan komponentin laitan sivun alkuun, koska haluan matkia selaimen tavallista toimintaa.

Koodi 4. AlertScreenReader-komponentti

```
class AlertScreenReader extends React.Component {
  componentDidMount() {
    this.props.focus.current.focus();
    document.title = pageNames[this.props.location].title;
  }
  render() {
    return (
      <div aria-live="assertive">
        {pageNames[this.props.location].announce}
      </div>
    );
  }
}
```

Kun AlertScreenReader-komponentti lisätään näkymään, se hakee pageNames-objektista sivulle uniikin ilmoituksen. Ilmoitus haetaan komponentin attribuutilla ”location”, joka on määritetty komponenttia käytettäessä. Divin attribuutti ’aria-live=”assertive”’ käskee ruudunlukijaa lukemaan komponentin sisällön heti sen vaihtuessa. Funktio ”componentDidMount” laukeaa myös komponentin lisäyksellä. Funktion sisällä JavaScript siirtää fokuksen komponentin ”focus” attribuutilla ilmoitettuun elementtiin. (Koodi 4.)

Koodi 5. ”FocusReset” -komponentti

```
const FocusReset = React.forwardRef((props, ref) => (
  <div ref={ref} tabIndex="-1" aria-hidden />
));
```

FocusReset-komponentti ottaa vastaan fokuksen käyttäjän navigoidessa uudelle sivulle, AlertScreenReader -komponentti siirtää fokuksen siihen. FocusReset-komponenttiin yhdistetään ref-attribuutti ja tabIndex määritetään arvolla ”-1”, jotta siihen voi JavaScriptin avulla siirtää fokuksen. Attribuutilla ”aria-hidden” käsketään ruudunlukijaa olla lukematta komponentin sisältöä, koska se on tyhjä. (Koodi 5.) Tämän jälkeen kaikki on kasattava yhteen.

Liittessä 1 on kaikki kasattuna yhteen. FocusReset -komponenttiin tarjotaan sisältönä Reactillä luotu ref-muuttuja ja sama muuttuja lisätään myös AlertScreenReader-komponenttiin. AlertScreenReader lisätään jokaisen reitin sisään ja FocusReset pääsivun alkuun. Esimerkkisovellus ilmoittaa ruudunlukijan käyttäjille navigaatioiden tapahtuneen ja hoitaa fokuksen siirron. En avaa tässä opinnäytetyössä tarkemmin sovelluksen muita komponentteja, kuten näkymiä, mutta ne ovat nähtävissä tyyleineen esimerkkisovelluksessa osoitteessa https://codesandbox.io/s/ywpomrw8kv. Sovellus ei ole täydellinen, mutta tarjoaa yksinkertaisen ratkaisun ongelmaan.

7 Yhteenveto

SPA-verkkosovellukset mahdollistavat nopean ja sujuvan interaktion sovelluksen kanssa. Näin SPA-sovellukset tarjoavat paremman käyttökokemuksen monille käyttäjille, mutta osa selaimen logiikasta jää kehittäjien toteutettavaksi. Kehittäjien on tiedostettava SPA-sovellusten puutteet ja samalla uudet mahdollisuudet, jotta uusin saavutettavuusongelmiin voidaan kehittää ratkaisu. Muuten SPA-sovellusten uudesta tavasta navigoida kärsii ruudunlukijoiden käyttäjät.

Saavutettavuusajattelu ei ole kehittäjien keskuudessa ollut kovin trendikäs aihe ja saavutettavuuden puutteita löytyy monista sovelluksista. Saavutettavuus on asiakasprojekteissa usein hankala perustella asiakkaalle, ja monet saavutettavuuden osa-alueet ovat tavalliselle käyttäjälle näkymättömiä. Eettisen suunnittelun periaatteilla on kuitenkin tärkeä huomioida kaikki käyttäjät ja tarjota kaikille mahdollisuus käyttää palveluita tasavertaisesti. Samalla saavutettavuudesta hyötyy myös sovelluksen muu käytettävyys.

Mielestäni usein kehittäjien valinnat eivät kuitenkaan ole tietoisia, vaan pohjautuvat ymmärryksen puutteeseen. Kehittäjät eivät välttämättä ymmärrä ruudunlukijan toimintaa tai uutta teknologiaa tarpeeksi, jotta osaisivat ottaa kaikki saavutettavuuden osa-alueet huomioon. Saavutettavuusdirektiivi onneksi pakottaa julkisen hallinnon sovellusten kehittäjiä kartuttamaan osaamistaan saavutettavuudesta ja miettimään saavutettavuutta kehitystyössään. Uskon tämän kautta kehittäjien uuden osaamisen heijastuvan myös muihin sovelluksiin ja näin parantavan yleistä saavutettavuutta verkossa.

SPA-sovellusten saavutettavuushaasteista huolimatta ne tarjoavat kehittäjille myös mahdollisuuden kontrolloida ruudunlukijan luettua sisältöä, jota ei aiemmin sivunvaihdoksilla pystynyt tekemään. Tämän lisäksi SPA-sovellusten saavutettavuusongelmat on

mahdollista korjata hyödyntämällä uusia verkko-ohjelmointirajapintoja ja WAI-ARIA-tageja. Ongelmat ratkeavat helposti yksinkertaisella komponentilla, mutta uskon SPA-sovellusten avaavan myös uusia tapoja parantaa ruudunlukijoiden käyttäjien käyttökokemusta. Hyvällä suunnittelulla ruudunlukioidenkin käyttäjät hyötyvät SPA-sovellusten ominaisuuksista.

Lähteet

Akamai 2009. Akamai Reveals 2 Seconds As The New Threshold Of Acceptability For ECommerce Web Page Response Times [verkkojulkaisu]. <https://www.akamai.com/us/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp> (luettu 31.10.2018)

American Foundation for the Blind 2018. Screen Reading Technology and Refreshable Braille Displays [verkkosivu]. <http://www.afb.org/info/living-with-vision-loss/using-technology/assistive-technology-videos/screen-reading-technology/1235> (luettu 3.11.2018).

Baker Mike 2009. What is a Software Framework? And why should you like ‘em [verkkojulkaisu]? <https://www.cimetrix.com/blog/bid/22339/what-is-a-software-framework-and-why-should-you-like-em> (luettu 20.10.2018).

Celia 2018. Saavutettavuus [verkkosivu]. <https://www.celia.fi/saavutettavuus/> (luettu 7.11.2018).

DeLuca Robert 2017. Single Page Apps routers are broken [verkkojulkaisu]. <https://medium.com/@robdel12/single-page-apps-routers-are-broken-255daa310cf> (luettu 2.11.2018).

Facebook 2018. Facebook Reports Third Quarter 2018 Results [verkkojulkaisu]. <https://investor.fb.com/investor-news/press-release-details/2018/Facebook-Reports-Third-Quarter-2018-Results/default.aspx> (luettu 9.11.2018).

Flanagan David 2011. JavaScript - The definitive guide. 6. painos. 1.

Haverbeke Marijn 2011. Eloquent JavaScript – A Modern Introduction to Programming. No starch press. 189–195.

IBM Knowledge Center n.d. The client/server model [verkkosivu]. <https://www.ibm.com/support/knowledgecenter/en/SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_clnt_sevr_model.html> (luettu 8.11.2018).

Itsnat 2015. The Single Page Interface Manifesto [verkkojulkaisu]. <http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php> (luettu. 20.10.2018).

Mangesh Bulkar 2016. The Difference Between Static and Dynamic Websites? [verkkojulkaisu]. <https://www.linkedin.com/pulse/difference-between-static-dynamic-websites-mangesh-bulkar/> (luettu 30.10.2018).

MDN web docs 2018a. ARIA Live Regions [verkkosivu]. <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions> (luettu 20.10.2018).

MDN web docs 2018b. Event.preventDefault() [verkkosivu]. <https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault> (luettu 8.11.2018).

Neoteric 2016. Single-page application vs. multiple-page application [verkkojulkaisu]. <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58> (luettu 30.10.2018)

Ostrovsky Igor 2010. What really happens when you navigate to a URL [verkkojulkaisu]. <http://igoro.com/archive/what-really-happens-when-you-navigate-to-a-url/> (luettu 10.11.2018).

React 2018a. React [verkkosivu]. <https://reactjs.org/> (luettu 7.11.2018).

React 2018b. Refs and the DOM [verkkosivu]. <https://reactjs.org/docs/refs-and-the-dom.html> (luettu 5.11.2018).

React training 2018. React Router: Declarative Routing for React.js [verkkosivu]. <https://reacttraining.com/react-router/web/guides/basic-components> (luettu 7.11.2018).

Saaranen-Kauppinen & Puusniekka 2006. KvaliMOTV – Menetelmäopetuksen tietovaranto [verkkojulkaisu]. Tampere: Yhteiskuntatieteellinen tietoarkisto [ylläpitäjä ja tuottaja]. <http://www.fsd.uta.fi/menetelmaopetus/> (luettu 1.10.2018).

Sherman Paul 2018. How Single-page Applications Work [verkkojulkaisu]. <https://medium.com/@pshrmn/demystifying-single-page-applications-3068d0555d46> (luettu 24.10.2018).

Smith Jeff 2018. 9 Popular JavaScript Frameworks Used in 2018 [verkkojulkaisu]. <https://raygun.com/blog/popular-javascript-frameworks/> (luettu 2.11.2018).

Suomi.fi 2018. Yleistä verkkopalvelusta [verkkosivu]. <https://www.suomi.fi/ohjeet-ja-tuki/yleista-verkkopalvelusta> (luettu 1.11.2018).

Valtiovarainministeriö 2018. Saavutettavuus [verkkosivu]. <https://vm.fi/saavutettavuusdirektiivi> (luettu 1.10.2018).

W3C 2016. HTML & CSS [verkkosivu]. <https://www.w3.org/standards/webdesign/htmlcss> (luettu 1.11.2018).

W3C 2008. Web Content Accessibility Guidelines (WCAG) 2.0 [verkkojulkaisu]. < https://www.w3.org/TR/WCAG20/> (luettu 30.10.2018).

WebAIM 2018. Screen Reader User Survey #7 Results [verkkojulkaisu]. <https://webaim.org/projects/screenreadersurvey7/> (luettu 2.11.2018).

Web Accessibility Initiative 2018a. WAI-ARIA Overview [verkkosivu]. <https://www.w3.org/WAI/standards-guidelines/aria/> (luettu 16.10.2018).

Web Accessibility Initiative 2018b. How to Meet WCAG 2 (Quick Reference) [verkkosivu]. <https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=211%2C243#principle2> (luettu 2.10.2018).

Web Accessibility Initiative 2018c. Understanding Success Criterion 2.4.3: Focus Order [verkkosivu]. <https://www.w3.org/WAI/WCAG21/Understanding/focus-order.html> (luettu 2.11.2018).

Web Accessibility Initiative 2018d. Inserting dynamic content into the Document Object Model immediately following its trigger element [verkkosivu]. <https://www.w3.org/WAI/WCAG21/Techniques/client-side-script/SCR26> (luettu 2.11.2018).

Esimerkkisovellus

```
// importattava tarvittavat komponentit
class App extends React.Component {
  constructor(props) {
    super(props);
    this.FocusRef = React.createRef();
  }
  render() {
    return (
      <Router>
        <React.Fragment>
          <FocusReset ref={this.FocusRef} />
          <Link to=”/”>Home</Link>
          <Link to=”/page”>Other page</Link>
          <Route
            exact
            path=”/”
            render={() => (
              <React.Fragment>
                <AlertScreenReader focus={this.FocusRef} location=”home” />
                <HomePage />
              </React.Fragment>
            )}
          />
          <Route
            path=”/page”
            render={() => (
              <React.Fragment>
                <AlertScreenReader focus={this.FocusRef} location=”page” />
                <OtherPage />
              </React.Fragment>
            )}
          />
        </React.Fragment>
      </Router>
    );
  }
}
```