

## **Saavutettava SPA-sovellus**

Joonas Pilli

Opinnäytetyö  
Marraskuu 2019  
Liiketaloudenala  
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t) Pilli, Joonas	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Marraskuu 2019
	Sivumäärä 76	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi <b>Saavutettava SPA-sovellus</b>		
Tutkinto-ohjelma Tietojenkäsittelyn tutkinto-ohjelma		
Työn ohjaaja(t) Tommi Tuikka		
Toimeksiantaja(t)		
<p>Tiivistelmä</p> <p>Verkko on pohjimmiltaan suunniteltu kaikkien ihmisten käytettäväksi, mutta puutteellisesti toteutetut verkkopalvelut voivat jättää suuria ihmisryhmiä kykenemättömiksi hyödyntämään niitä. Saavutettavuudella tarkoitetaan verkkopalvelun helppoa lähestyttävyyttä koskien kaikkia ihmisiä. Digitalisaation myötä yhä useampi oleellinen palvelu on siirtynyt tai on siirtymässä ensisijaisesti verkkoon, joten saavutettavuuteen täytyy kiinnittää entistä enemmän huomiota. Modernien, SPA-arkkitehtuuria hyödyntävien web-sovellusten dynaamisuus asettaa useita haasteita saavutettavuudelle.</p> <p>Opinnäytetyön tavoitteena oli tutkia, kuinka moderneja SPA-sovelluksia voitaisiin toteuttaa World Wide Web Consortiumin (W3C) julkaisemien saavutettavuusohjeiden mukaisesti. Tavoitetta lähestyttiin valitsemalla tutkittavaksi kolme moderneille SPA-sovelluksille tyyppilistä käyttöliittymäelementtiä ja -toiminnallisuutta. Tutkimuskohteille pyrittiin tunnistamaan niille olennaisimmat, W3C:n julkaiseman Web Content Accessibility Guidelines 2.1 -dokumentin määrittelemät, saavutettavuuden onnistumiskriteerit, jotka otettiin käsiteltäväksi. Hyödyntäen web-sovelluskehys Angularia valituille tutkimuskohteille toteutettiin käsiteltäväksi valittujen onnistumiskriteerien vaatimusten mukaiset esimerkki-implementaatiot, joiden saavutettavuusseikat perusteltiin W3C:n dokumentaation kannalta.</p> <p>Tutkimustuloksena tuotettiin käytännön esimerkkejä saavutettavuuden edistämisestä modernissa web-sovelluskehityksessä sekä esimerkkiratkaisuja tukevaa teoriaa. Luodut esimerkkiratkaisut pyrittiin pitämään laajennettavina erilaisia käyttötarpeita varten ja niiden toimivuus pyrittiin varmistamaan useilla suosituilla ruudunlukuohjelma- ja verkkoselainkombinaatioilla. Tutkimustuloksia voidaan hyödyntää referenssinä toteutettaessa esimerkkien kaltaisia toiminnallisuksia moderneihin web-sovelluksiin.</p>		
<p>Avainsanat (asiasanat)</p> <p>Angular, saavutettavuus, saavutettavuusdirektiivi, verkkosisällön saavutettavuusohjeet, SPA-arkkitehtuuri, SPA-sovellus, WCAG, web-sovellus, verkkosovellus</p>		
Muut tiedot (Salassa pidettävät liitteet)		

Author(s) Pilli, Joonas	Type of publication Bachelor's thesis	Date November 2019 Language of publication: Finnish
	Number of pages 76	Permission for web publication: x
Title of publication <b>Accessible single-page application</b>		
Degree programme Bachelor's Degree in Business and Administration		
Supervisor(s) Tuikka, Tommi		
Assigned by		
Abstract  <p>The World Wide Web is fundamentally designed for all humans, but insufficiently implemented web services can leave large groups of people unable to use them. Accessibility refers to creating web services easily available for all people to use. With digitalization, an increasing amount of essential web services have migrated or are in process of migrating primarily to the Web, which is why accessibility has become increasingly more important. The dynamic nature of modern web applications utilizing the single-page architecture creates several challenges for accessibility.</p> <p>The goal of the thesis was to study how modern single-page applications could be implemented in accordance with the accessibility guidelines published by the World Wide Web Consortium (W3C). In order to achieve relevant data, three user interface elements and functionalities common to modern single-page applications were selected for study. An attempt was made to identify the most essential accessibility success criteria for the research subjects, as they are defined by the Web Content Accessibility Guidelines 2.1 document published by the W3C. Utilizing the Angular web application framework, exemplary implementations meeting the requirements of the selected success criteria were implemented for the selected research subjects, the accessibility aspects of which were substantiated by the W3C documentation.</p> <p>The research results were practical examples on how to enhance accessibility in modern web development and the theory to support them. There was a conscious attempt to keep the created solutions flexible. Their functionality was tested to work with a variety of popular screen reader and web browser combinations. The research results can be used as a reference for the implementation of similar functionalities to modern web applications.</p>		
Keywords/tags ( <a href="http://vesa.lib.helsinki.fi/">subjectshttp://vesa.lib.helsinki.fi/</a> ) Angular, accessibility, accessibility directive, Web Content Accessibility Guidelines (WCAG), single page architecture, single-page application, web application		
Miscellaneous (Confidential information)		

## Sisältö

<b>1</b>	<b>Johdanto .....</b>	<b>7</b>
<b>2</b>	<b>Tutkimusasetelma .....</b>	<b>8</b>
2.1	Tutkimuksen taustat, tavoitteet ja rajoitteet.....	8
2.2	Tutkimuskysymykset .....	9
2.3	Tutkimusmenetelmät .....	10
<b>3</b>	<b>SPA-sovellukset ja saavutettavuus .....</b>	<b>11</b>
<b>4</b>	<b>Yleiset saavutettavuusongelmat .....</b>	<b>12</b>
<b>5</b>	<b>Käsiteltävät saavutettavuusongelmat .....</b>	<b>14</b>
5.1	SPA-arkkitehtuuri .....	15
5.2	Modaalinen dialogi.....	18
5.3	Ääretön vieritys .....	19
5.4	Yhteenveto .....	20
<b>6</b>	<b>Saavutettavuusongelmien korjaaminen ja välttäminen .....</b>	<b>21</b>
6.1	SPA-arkkitehtuuri .....	23
6.1.1	Fokuksen hallinta.....	23
6.1.2	HTML-dokumentin otsikon päivittäminen .....	33
6.1.3	Sivun vaihdosta ilmoittaminen.....	37
6.1.4	Yhteenveto .....	44
6.2	Modaalinen dialogi.....	45
6.2.1	Dialog ja alertdialog -roolit.....	45
6.2.2	Fokuksen hallinta.....	50
6.2.3	Näppäimistökomennot.....	55
6.2.4	Yhteenveto .....	57
6.3	Ääretön vieritys .....	58
6.3.1	Feed-rooli.....	59
6.3.2	Article-rooli .....	62
6.3.3	Näppäimistökomennot.....	64
6.3.4	Yhteenveto .....	69

<b>7</b>	<b>Pohdinta.....</b>	<b>71</b>
7.1	Mahdolliset jatkotutkimusaiheet .....	73
	<b>Lähteet .....</b>	<b>74</b>

## Kuviot

Kuvio 1. Esimerkkisovelluksen reittien pääsisällön otsikoimisessa hyödynnettävät tietotyypit .....	27
Kuvio 2. Näyte esimerkkisovelluksen pääreititysmoduulista .....	28
Kuvio 3. Näyte reittien pääsisällön otsikon päivittämisestä RoutePurposeService-palvelussa.....	29
Kuvio 4. Esimerkkisovelluksen onPathChange-apufunktio .....	30
Kuvio 5. Esimerkkisovelluksen MainContentComponent-komponentti .....	31
Kuvio 6. MainContentComponent-komponentin käyttöönotto esimerkkisovelluksen juurikomponentin HTML-rakenteessa .....	33
Kuvio 7. Esimerkkisovelluksen HTML-dokumentin otsikon hallitsemisessa hyödynnettäviä tietotyyppisiä .....	34
Kuvio 8. Havainnollistus esimerkkisovelluksen HTML-dokumentin otsikon rakenteesta .....	35
Kuvio 9. Näyte HTML-dokumentin otsikon päivittämisestä esimerkkisovelluksen RoutePurposeService-palvelussa.....	36
Kuvio 10. Havainnollistus esimerkkisovelluksen HTML-dokumentin otsikon muodostamisesta otsikkotasolla .....	37
Kuvio 11. Esimerkkisovelluksen statusviestien käsittelyssä hyödynnettävät tietotyypit .....	38
Kuvio 12. StatusAnnouncementService-palvelu.....	38
Kuvio 13. Esimerkkisovelluksen StatusAnnouncerComponent-komponentin HTML-rakenne .....	39
Kuvio 14. CSS-tyyliohjeet elementin piilottamiseksi visuaalisesti.....	41
Kuvio 15. Esimerkkisovelluksen StatusAnnouncerComponent-komponentin luokka.....	42

Kuvio 16. Näyte sivun vaihdoksen ilmoittamisesta esimerkksiovelluksen RoutePurposeService-palvelussa.....	44
Kuvio 17. Esimerkkisovelluksen dialogien käsittelyssä hyödynnettävät tietotyypit .....	46
Kuvio 18. Näyte esimerkksiovelluksen DialogService-palvelusta .....	48
Kuvio 19. Esimerkkisovelluksen DialogComponent-komponentin HTML-rakenne .....	49
Kuvio 20. Näyte esimerkksiovelluksen DialogComponent-komponentin luokasta .....	51
Kuvio 21. Näyte esimerkksiovelluksen RestrictFocusDirective-direktiivistä .....	53
Kuvio 22. Fokuksen palauttamiseen liittyvät lisäykset DialogService-palveluun ..	55
Kuvio 23. Esimerkkisovelluksen näppäimistökomentojen määrittämisessä hyödynnettävät tietotyypit.....	56
Kuvio 24. Näppäimistökomentolisäykset esimerkksiovelluksen DialogComponent-komponenttiin.....	57
Kuvio 25. Esimerkkisovelluksen ääretön vieritys -implementaatiossa hyödynnettävät tietotyypit.....	60
Kuvio 26. Näyte esimerkksiovelluksen FeedComponent-komponentin luokasta	61
Kuvio 27. Näyte esimerkksiovelluksen FeedComponent-komponentin HTML-rakenteesta .....	62
Kuvio 28. Näyte esimerkksiovelluksen FeedItemComponent-komponentin luokasta.....	63
Kuvio 29. Tietotyyppilisäykset esimerkksiovelluksen näppäimistökomentojen toteuttamiseksi .....	65
Kuvio 30. Näppäimistökomentolisäykset esimerkksiovelluksen FeedComponent-komponentin luokkaan .....	66
Kuvio 31. Näyte esimerkksiovelluksen FeedComponent-komponentin päivitetystä HTML-rakenteesta .....	67
Kuvio 32. Lisäykset esimerkksiovelluksen FeedItemComponent-komponentin luokkaan.....	67
Kuvio 33. Näppäimistökomentojen dokumentaation lisäys esimerkksiovelluksen FeedComponent-komponentin HTML-rakenteeseen .....	68
Kuvio 34. Esimerkkisovelluksen VHiddenDirective-direktiivi .....	69

## Taulukot

Taulukko 1. Tarkasteltaviksi valitut verkkosivustot .....	12
Taulukko 2. Tutkimuksessa käsiteltävät käyttöliittymäelementit ja - toiminnallisuudet .....	14
Taulukko 3. Saavutettavuusongelmien havaitsemisessa hyödynnettävät työkalut .....	15
Taulukko 4. SPA-arkkitehtuurin käsiteltävät onnistumiskriteerit .....	18
Taulukko 5. Modaalin käsiteltävät onnistumiskriteerit .....	19
Taulukko 6. Äärettömän vierityksen käsiteltävät onnistumiskriteerit .....	20
Taulukko 7. Yhteenveto käsiteltävistä onnistumiskriteereistä .....	21
Taulukko 8. Esimerkki-implementaatioiden testaamiseen käytettävät ohjelmistot .....	22
Taulukko 9. Yhteenveto SPA-arkkitehtuurille valittujen onnistumiskriteerien käsittelystä .....	45
Taulukko 10. Yhteenveto modaalisille dialogeille valittujen onnistumiskriteerien käsittelystä .....	58
Taulukko 11. Yhteenveto äärettömälle vieritykselle valittujen onnistumiskriteerien käsittelystä .....	71

## Käsitteet

<b>Ajax</b>	Useasta web-tekniikasta koostuva toimintamalli, joka mahdollistaa dynaamisen asiakas-palvelin-vuorovaikutuksen web-sovelluksissa, ilman verkkosivun uudelleen lataamista.
<b>Angular</b>	Googlen kehittämä sovelluskehys web-sovelluksien kehittämiseen.
<b>Avustava teknologia</b>	Laitteisto ja/tai ohjelmisto, joka toimii asiakassovelluksena tai yhdessä asiakassovelluksen kanssa, tarjoten vammaisille tai rajoitteisille käyttäjille heidän tarpeitansa vastaavia toiminnallisuuksia.
<b>Cascading Style Sheets (CSS)</b>	Tyyliohjekieli, jolla voidaan määritellä HTML-dokumenttien visuaalista ulkoasua.
<b>Hypertext Markup Language (HTML)</b>	Merkintäkieli, jota käytetään erityisesti verkkosivudokumenttien kuvaamiseen.
<b>JavaScript</b>	Ohjelmointikieli, jota käytetään etenkin verkkopalveluiden toiminnallisuuden ehostamiseen.
<b>Ruudunlukija</b>	Avustava teknologia, jota useimmiten sokeat henkilöt käyttävät lukeakseen tekstimuotoista informaatiota.
<b>Saavutettavuus</b>	Saavutettavuudella tarkoitetaan verkkopalveluiden helppoa lähestyttävyyttä.



<b>SPA-arkkitehtuuri</b>	Verkkosivun toimintatapa, jossa sivujen/näkymien välillä siirtyminen tapahtuu HTML-dokumentin dynaamisella uudelleenkirjoittamisella.
<b>SPA-sovellus</b>	SPA-arkkitehtuuria hyödyntävä web-sovellus
<b>SPA-teknologia</b>	Sovelluskehys tai ohjelmointikirjasto, jota voidaan käyttää SPA-arkkitehtuuria hyödyntävän web-sovelluksen toteuttamiseen.
<b>TypeScript</b>	JavaScriptistä johdatettu ohjelmointikieli, joka yleensä muunnetaan JavaScriptiksi ennen suoritusta.
<b>Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA)</b>	W3C:n julkaisema tekninen spesifikaatio, joka määrittelee tapoja edistää verkkosisällön ja web-sovelluksien saavutettavuutta.
<b>Web-sovellus</b>	Verkkoselaimella käytettävä sovellus
<b>Web Content Accessibility Guidelines (WCAG)</b>	W3C:n julkaisema ohjeistusdokumentti verkkosisällön saavutettavuuden edistämiseen.
<b>World Wide Web Consortium (W3C)</b>	Useasta jäsenorganisaatiosta koostuva kansainvälinen organisaatio, joka kehittää ja ylläpitää web-standardeja.

# 1 Johdanto

Verkko on pohjimmiltaan suunniteltu kaikkien ihmisten käytettäväksi, välittämättä henkilön tavasta tai kyvystä käyttää sitä. Verkkopalvelut voivat helpottaa ihmisten kommunikaatiota ja vuorovaikutusta huomattavasti, mutta puutteellisesti toteutetut verkkopalvelut saattavat jättää osan ihmisistä kykenemättömiksi hyödyntämään niitä. (Henry & McGee n.d.) Saavutettavuudella tarkoitetaan verkkopalvelun helppoa lähestyttävyyttä kaikille ihmisille (Yleistä tietoa saavutettavuudesta n.d.).

Nyky aikaisten verkkosivujen, kuten SPA-sovelluksien, dynaaminen luonne asettaa haasteita muun muassa näkörajoitteisille henkilöille, jotka voivat selata verkkoa avustavien teknologioiden avulla. Nykyisellään avustavat teknologiat, kuten ruudunluokohjelmat, eivät välttämättä osaa toivotulla tavalla reagoida verkkosivun rakenteessa tapahtuviin muutoksiin, mikäli verkkosivu on toteutettu puutteellisesti. Koska avustavien teknologioiden hyödyllisyys on usein hyvin riippuvaista itse verkkosivun toteutuksesta, tulisi verkkosivujen kehittäjien tiedostaa dynaamisuuden tuomat vaikutukset verkkosivujen saavutettavuuteen. (Accessibility of AJAX Applications 2014.)

Saavutettavuudesta on hyötyä useille isoille käyttäjäryhmille, joilla on jonkunlaisia rajoitteita vuorovaikuttaa verkkopalveluiden kanssa perinteisin tavoin, mutta saavutettavasta toteutuksesta voivat kuitenkin hyötyä kaikki ihmiset, sekä myös yritykset. Saavutettavuuden huomioiminen edistää muun muassa verkkosivun ymmärrettävyyttä ja loogista rakennetta, jotka puolestaan voivat edistää verkkosivun sijoitusta hakukoneissa. (Miksi saavutettavuus on tärkeää? n.d.) Saavutettavuudelle voi myös olla lainsäädännöllisiä velvoitteita. Vuonna 2016 Euroopan parlamentti hyväksyi saavutettavuusdirektiivin, jonka tarkoituksena on, että kaikkien EU:n jäsenmaiden valtioiden ja kuntien verkkopalvelut olisivat kaikkien kansalaisten käytettävissä. Saavutettavuusdirektiivi koskee tällöin myös Suomea, joka ottaa direktiivin lainsäädännön portaittain käyttöön 23.6.2021 mennessä. (Saavutettavuus n.d.).

Opinnäytetyön tavoitteena on tutkia, kuinka moderneja SPA-sovelluksia voitaisiin toteuttaa W3C:n julkaisemien saavutettavuusohjeiden mukaisesti. Tavoitetta lähestytään valitsemalla tutkittavaksi kolme, moderneille SPA-sovelluksille yleistä, käyttöliittymäelementtiä tai -toiminnallisuutta, joiden saavutettava toteutus oletettavasti vaatii erityistä huomiota. Valituille tutkimuskohteille pyritään havaitsemaan niille olennaisimmat, World Wide Web Consortiumin (W3C) *Web Content Accessibility Guidelines (WCAG) 2.1* -dokumentin määrittämät, onnistumiskriteerit, jotka käsitellään kehittämällä kohteille valittujen onnistumiskriteerien vaatimuksien mukaiset esimerkkiimplementaatiot sovelluskehys Angularilla. Esimerkkiratkaisujen kehittämisessä hyödynnetään useita W3C:n julkaisemia saavutettavuusohjeistuksia ja ratkaisujen saavutettavuusasiat perustellaan näiden ohjeistuksien kannalta. Saavutettavuusongelmat eivät kuitenkaan pohjimmaltaan ole sovelluskehyskohtaisia tai niiden aiheuttamia, joten vaikka ratkaisussa pyritään tehokkaasti hyödyntämään Angularin erityispiirteitä, tulisi ratkaisujen abstrakti logiikka olla pitkälti hyödynnettävissä myös muissa modernin web-kehityksen työkaluissa.

## 2 Tutkimusasetelma

### 2.1 Tutkimuksen taustat, tavoitteet ja rajoitteet

Tulin tietoisiksi saavutettavuudesta työskennellessäni yrityksessä, jossa kehitettiin ja ylläpidettiin erään kansainvälisesti toimivan yrityksen verkkopalveluita. Kyseisellä asiakasyrityksellä oli, heidän toimialastansa ja toimialueestansa johtuen, lainsäädännöllisiä velvoitteita heidän verkkopalveluiden saavutettavuudelle, joka käytännössä tarkoitti, että heidän verkkopalveluiden tulisi täyttää *Web Content Accessibility Guidelines (WCAG)* -dokumentin AA-noudattamistason vaatimukset. Verkkopalveluiden saavutettavuuteen kiinnitettiin siis erityistä huomiota. Työssäni kohtasin saavutettavuuden kannalta ongelmallisia tilanteita usein, etenkin toteutettaessa dynaamista HTML:ää vaativia toiminnallisuuksia, kuten eräänlaista SPA-sovellusta. W3C:n julkaisemat, saavutettavuuteen liittyvät ohjeistukset ovat laajoja ja niitä on useita, jolloin niistä voi olla haastavaa löytää juuri toteutettavalle toiminnallisuudelle olennaiset

asiat. Internetistä löytää epävirallisia, spesifimpiä ohjeistuksia saavutettavuuden huomioimiselle SPA-sovelluksissa, mutta kokemukseni mukaan kyseisiä ohjeistuksia harvoin perustellaan virallisten (W3C:n) saavutettavuusohjeistuksien kannalta, jolloin niiden pätevyys jää epäselväksi. Epävirallisissa ohjeistuksissa esitettävät ratkaisut eivät myöskään usein toimi luotettavasti eri avustava teknologia- ja verkkoselainkombinaatioin välillä.

Opinnäytetyön tavoitteena on tutkia, kuinka moderneja SPA-sovelluksia voitaisiin toteuttaa W3C:n julkaisemien saavutettavuusohjeiden mukaisesti. Käytännössä tämä tapahtuu tutkimalla käyttäjien yleisesti kohtaamia, modernien SPA-sovelluksien käyttöliittymässä ilmeneviä, saavutettavuusongelmatilanteita. Tutkittavaksi valitaan yleisesti esiintyviä käyttöliittymäelementtejä ja -toiminnallisuuksia, joiden saavutettava toteutus oletettavasti vaatii erityistä huomiota. Valituille tutkimuskohteille pyritään havaitsemaan niille olennaisimmat, W3C:n *Web Content Accessibility Guidelines 2.1* -dokumentin määrittämät, onnistumiskriteerit, joiden perusteella tutkimuskohteille kehitetään onnistumiskriteerien vaatimuksien mukaiset esimerkkiratkaisut sovelluskehys Angularilla. Luodut esimerkkiratkaisut perustellaan W3C:n ohjeistuksien kannalta ja niiden raportoinnissa pyritään tuomaan esiin saavutettavuuden kannalta olennaisimmat asiat. Esimerkkiratkaisujen toimivuus pyritään myös varmistamaan usealla eri ruudunlukuohjelma- ja verkkoselainkombinaatiolla. Tutkimustuloksena on tarkoitus tuottaa virallisten ohjeistuksien kannalta perusteltuja, subjektiivisesti tehokkaita, käytännön esimerkkejä ja niitä tukevaa teoriaa saavutettavuuden huomioimisesta modernissa web-sovelluskehityksessä.

## 2.2 Tutkimuskysymykset

- Mitä yleisiä saavutettavuusongelmia esiintyy moderneissa SPA-sovelluksissa?
- Mikä on valittujen saavutettavuusongelmien merkitys käyttäjän näkökulmasta?
- Miten valitut saavutettavuusongelmat voitaisiin tehokkaasti korjata tai välttää?

## **Tehokkuus**

Tutkimuksessa tuotettavien ratkaisujen tehokkuudella tarkoitetaan, että esimerkkiimplementaatioissa pyritään uudelleenkäytettävyyteen ja noudattamaan teknologia-kohtaisia hyviä käytäntöjä. Tuotettavien ratkaisujen tulisi myös täyttää käsiteltäväksi valittavien onnistumiskriteerien vaatimukset.

## **2.3 Tutkimusmenetelmät**

### **Tutkimusote**

Kananen (2008, 17) määrittelee kvantitatiivisen tiedon olevan lukuja, luotettavaa ja tieteellistä, kun taas kvalitatiivinen tieto on sanoja ja lauseita. Kananen (2008, 27) myös kertoo kvalitatiivisen tutkimuksen tiedon luonteen olevan subjektiivista ja sen päättelyn tapahtuvan induktiivisesti. Tämän tutkimuksen tavoitteena on luoda käytännön toiminnasta johdatettavaa, subjektiivista teoriaa saavutettavuuden tehokkaasta toteuttamisesta modernissa SPA-sovelluskehityksessä, eikä tarkoituksena ole tuottaa mitattavaa tietoa. Täten tässä tutkimuksessa on päädytty soveltamaan kvalitatiivista tutkimusotetta.

### **Tiedonkeruumenetelmät**

Tiedonkeruussa tutkimuksessa sovelletaan tapaustutkimuksen periaatteita. Tutkimuksen tapauksina toimii kolme, järjestelmällisesti valittua, saavutettavuuden kannalta ongelmallista käyttöliittymäelementtiä ja -toiminnallisuutta. Valittujen tutkimuskohteiden käsittelystä pyritään tuottamaan yksityiskohtaista ja intensiivistä tietoa.

### **Analyysimenetelmät**

Kananen (2008, 57) määrittelee, että laadullisessa tutkimuksessa tiedonkeruu- ja analyysi kulkevat rinnakkain, joka tukee tämän tutkimuksen tavoitteita hyvin. Saavutettavuusongelmille tuotettavat esimerkkiratkaisut, niiden perustelut ja niitä tukeva teoria tulevat yhdessä muodostamaan vastaukset opinnäytetyön tutkimuskysymyksiin.

### 3 SPA-sovellukset ja saavutettavuus

SPA-arkkitehtuuria hyödyntävät web-sovellukset eli yhden sivun sovellukset tai SPA-sovellukset, mahdollistavat dynaamiset verkkosivut, joiden käyttökokemus mukailee perinteisiä työpöytäsovelluksia (Flanagan 2011, 491). SPA-sovelluksissa hyödynnetään Ajax-mallia luomaan verkkosivu, jossa näkymät ja sisältö päivittyvät asynkronisesti ilman verkkosivun uudelleen lataamista, joka käytännössä tapahtuu muokkamalla sivun HTML-rakennetta JavaScriptin avulla. Tällainen dynaamisuus voi kuitenkin puutteellisesti toteutettuna huomattavasti vaikeuttaa tai jopa kokonaan estää joidakin ihmisryhmiä, kuten näkörajoitteisia henkilöitä, selaamasta verkkosivua. (Accessibility of AJAX Applications 2014.) Digitalisaation myötä yhä useampi ihmisille oleellinen palvelu on siirtynyt tai on siirtymässä ensisijaisesti verkkoon, joten verkkosivujen saavutettavuuteen on tarvetta kiinnittää entistä enemmän huomiota (Digitaalisten palvelujen ensisijaisuus n.d.).

W3C:n *Web Content Accessibility Guidelines (WCAG) 2.1* -dokumentti on Euroopan lainsäädännössä sovellettava ohjeistus verkkopalveluiden saavutettavuuden edistämiseksi, johon myös tämän tutkimuksen saavutettavuusmääritelmät perustetaan (L 327/84 2018). Kyseiset ohjeet määrittelevät joukon teknologiariippumattomia, testattavia lausuntoja, ns. onnistumiskriteerejä, joiden vaatimuksia noudattamalla verkkosisällöstä tehdään helpommin käytettävää henkilöille, joilla on erilaisia vammoja tai rajoitteita, jotka vaikuttavat heidän verkkoselaamiseensa. Onnistumiskriteerit on jaettu kolmeen noudattamistasoon, A-, AA- ja AAA-taso, joista A-taso on matalin ja AAA-taso korkein. Kriteerien noudattamistasoon vaikuttaa muun muassa kriteerin olennaisuus, sen toteutettavuus kaikissa sisältötyypeissä sekä sen toteuttamiseen vaadittavat taidot. (Web Content Accessibility Guidelines (WCAG) 2.1 2018; Understanding Conformance n.d.) Tässä tutkimuksessa hyödynnetään myös W3C:n muita, aihealueelle olennaisia julkaisuja, kuten *Accessible Rich Internet Applications (WAI-ARIA) 1.1* -dokumenttia.

## 4 Yleiset saavutettavuusongelmat

Tässä tutkimuksessa pyritään käsittelemään käyttäjien yleisemmin kohtaamia SPA-sovelluksien saavutettavuusongelmia, jotta tutkimustulokset palvelisivat modernin SPA-sovelluskehityksen yleisempiä, saavutettavuuden kannalta ongelmallisia tilanteita. Ongelmallisten tilanteiden havaitseminen aloitetaan tutkimalla ja testaamalla käyttöliittymäelementtejä ja -toiminnallisuuksia SPA-teknologioita merkittävästi hyödyntäviltä verkkosivustoilta, joilla vierailee käyttäjiä paljon ja usein.

Alexa Internet -verkkosivusto listaa Internetin suosituimpia sivustoja (Top sites n.d.). Listauksen kärkipäästä lähtien, tutkittavaksi valitaan viisi sivustoa, joiden rakenteesta pystyy selvästi havaitsemaan jonkin SPA-teknologian hyödyntämistä. Valituista viidestä sivustosta valitaan tutkimukseen käsiteltäväksi kolme erilaista, tutkimuksen tarkoitukselle sopivaa ja valittujen sivustojen kesken usein esiintyvää käyttöliittymäelementtiä tai -toiminnallisuutta. Käyttöliittymäelementtien ja -toiminnallisuuksien valinnassa on keskitytty verkkosivustojen etusivun työpöytäkäyttöön suunnattuun versioon, ellei verkkosivun peruskäyttö selvästi vaadi lisätoimenpiteitä, kuten sisäänkirjautumista tai siirtymistä sivuston toiselle sivulle. Jos verkkosivustosta ei ole saatavilla virallista suomen- tai englanninkielistä versiota, on se jätetty huomioimatta kontekstinymmärtämisvaikeuden vuoksi. Myös sisällöltään kyseenalaiset tai tutkimukseen sopimattomat verkkosivustot on jätetty huomioimatta. Listauksesta valitut verkkosivustot on listattu seuraavaan taulukkoon (ks. Taulukko 1).

Taulukko 1. Tarkasteltaviksi valitut verkkosivustot

Verkkosivusto	Alexa-sijoitus (10.8.2019)
Facebook.com	3
Twitter.com	11
Reddit.com	12
Instagram.com	14
Netflix.com	22

Millään valitulla verkkosivustolla ei vaikuta olevan veloitteita noudattaa **Euroopan** saavutettavuusdirektiivin määrittämää lainsäädäntöä, sillä ne eivät lukeudu kyseisen direktiivin kohdepiiriin (L 327/84 2018). Jokainen valituista verkkosivustoista on kuitenkin rakenteeltaan sopiva tutkimuksen tarkoitukselle ja usean sivuston rakenteesta voidaankin selkeästi havaita, että saavutettavuuteen on kiinnitetty huomiota.

Seuraava taulukko (ks. Taulukko 2) listaa käsiteltäviksi valitut käyttöliittymäelementit ja -toiminnallisuudet, jotka olivat havaittavissa tarkastelluissa verkkosivustoissa. Kaikki tutkittavaksi valitut käyttöliittymäelementit ja -toiminnallisuudet ovat enemmän tai vähemmän riippuvaisia dynaamisesta HTML:stä, jolloin ne voidaan katsoa potentiaalisiksi etenkin avustaviin teknologioihin liittyville saavutettavuusongelmille (Accessibility of AJAX Applications 2014).



Taulukko 2. Tutkimuksessa käsiteltävät käyttöliittymäelementit ja -toiminnallisuudet

Käyttöliittymäelementti tai -toiminnallisuus	Kuvaus	Valitsemisen syy
<b>SPA-arkkitehtuuri</b>	Toimintatapa, jossa sivujen tai näkymien välillä navigoiminen perustuu HTML-dokumentin uudelleenkirjoittamiseen.	SPA-sovelluksien määrittävä toiminnallisuus. Aiheuttaa verkkosivuun merkittäviä rakenteellisia muutoksia. Esiintyi kaikissa tutkimukseen valituissa verkkosivustoissa.
<b>Modaalinen dialogi</b>	Käyttöliittymään kuuluva ali-ikkunaelementti, joka vaatii käyttäjän vuorovai- kutusta ennen sovelluksen käytön jatkamista.	Aiheuttaa merkittäviä visuaalisia muutoksia sekä rakenteellisia muutoksia. Esiintyi lähes kaikissa tutkimukseen valituissa verkkosivustoissa.
<b>Ääretön vieritys (engl. infinite scrolling)</b>	Toiminnallisuus, jossa käyttäjän toiminnan seurauksena verkkosivuun laadetaan asynkronisesti lisää sisältöä, näennäisesti äärettömästi.	Aiheuttaa merkittäviä rakenteellisia muutoksia. Esiintyi kaikissa tutkimukseen valituissa verkkosivustoissa.

## 5 Käsiteltävät saavutettavuusongelmat

Valituille käyttöliittymäelementeille ja -toiminnallisuuksille mahdollisia saavutettavuusongelmatilanteita lähdetään kartoittamaan pyrkimällä havaitsemaan kohteille olennaisimmat onnistumiskriteerit *Web Content Accessibility Guidelines (WCAG) 2.1* -dokumentista sekä tarkastelemalla ja testaamalla kohteiden olemassa olevia imple-

mentaatioita niillä sivustoilla mistä ne on tutkimukseen valittu. Ongelmien havaitsemisessa hyödynnetään saavutettavuuden auditointityökaluja, ruudunlukuohjelmia sekä verkkoselaimia. Ruudunlukuohjelma- ja verkkoselainvalinnat perustuvat WebAIM-sivuston viimeisimpään ruudunlukuohjelmien käyttäjätutkimukseen sekä tutkijan käytettävissä oleviin resursseihin (Screen Reader User Survey #8 Results 2019). Seuraava taulukko (ks. Taulukko 3) listaa ongelmien havaitsemisessa hyödynnettävät työkalut.

Taulukko 3. Saavutettavuusongelmien havaitsemisessa hyödynnettävät työkalut

Työkalu	Kuvaus
<b>Google Chrome</b>	Verkkoselain
<b>Google Lighthouse</b>	Työkalu verkkosivujen laadun edistämiseen ja saavutettavuuden evaluointiin
<b>Microsoft Edge</b>	Verkkoselain
<b>Microsoft Narrator</b>	Ruudunlukuohjelma
<b>Mozilla Firefox</b>	Verkkoselain
<b>NVDA</b>	Ruudunlukuohjelma
<b>WAVE Evaluation Tool</b>	Saavutettavuuden evaluointityökalu

Verkkosivujen implementaatioista ja saavutettavuusohjeiden onnistumiskriteereistä pyritään kertomaan olennaisia huomioita, joilla pyritään tuomaan esiin havaittujen ongelmien merkitystä käyttäjälle sekä saavutettavan toteutuksen olennaisuutta. Käsiteltäväksi valittavien onnistumiskriteerien olennaisuutta tuodaan enemmän ilmi esimerkkiratkaisujen käsittelemisen yhteydessä.

## 5.1 SPA-arkkitehtuuri

SPA-arkkitehtuuria eli HTML-dokumentin uudelleenkirjoittamiseen perustuvaa navigointitoiminnallisuutta, hyödynnettiin kaikilla valituilla verkkosivustoilla. WCAG 2.1 ei

suoranaisesti käsittelee SPA-arkkitehtuuria, mutta sille enemmän tai vähemmän olennaisia onnistumiskriteerejä kuitenkin löytyy (Web Content Accessibility Guidelines (WCAG) 2.1 2018).

SPA-arkkitehtuurissa ongelmaksi nousee etenkin fokuksen hallitseminen. Näppäimistöä tai ruudunlukuohjelmia hyödyntävät käyttäjät selaavat verkkosivuja siirtymällä HTML-elementtien välillä näppäimistöä käyttäen eli he siirtävät fokusta elementistä toiseen. Vain yksi elementti voi kerrallaan olla fokusoituna ja ruudunlukuohjelmaa hyödyntävät käyttäjät voivat saada fokusoidusta elementistä erilaisia tietoja, kuten minkälainen elementti on kyseessä, mitä siinä lukee ja mitä sillä voidaan tehdä. Koska SPA-arkkitehtuurissa sivujen välillä siirtyminen tapahtuu HTML-dokumentin uudelleenkirjoittamisella, sivun vaihtamisen yhteydessä fokusoitu elementti saatetaan poistaa dokumentin rakenteesta, jolloin fokus on siirrettävä muualle. Eri verkkoselaimet käsittelevät fokuksen siirtämistä tällaisissa tilanteissa eri tavoin. Esimerkiksi jotkut selaimet vaikuttavat siirtävän fokuksen lähimpään, muutoksen jälkeen säilytettävään, isäntäelementtiin ja jotkut siirtävät sen HTML-dokumentin visuaalista rakennetta käärivään body-elementtiin. Fokuksen hallitsemisella pyritään siihen, että fokuksen siirtäminen tapahtuisi loogisessa järjestyksessä loogisiin elementteihin.

### **Havaintoja verkkosivustoilta**

Mikään tarkastelluista sivustoista ei vaikuta tekevän fokuksen hallintaa SPA-arkkitehtuurin mukaisen sivun vaihtamisen yhteydessä, eikä sivun vaihtumisesta suoranaisesti viestitä itse verkkosivujen toimesta. Fokusoidun elementin poistuessa HTML-dokumentin rakenteesta, Google Chrome ja Mozilla Firefox -verkkoselaimet vaikuttavat siirtävän fokuksen lähimpään säilytettävään isäntäelementtiin, joka tarjoaa tarkastelluissa sivustoissa useimmiten loogisen jatkamiskohdan näppäimistö-pohjaiselle selailulle. Fokuksen hallitsemisen jättäminen verkkoselaimen vastuulle voi kuitenkin johtaa epäloogiseen fokuksen siirtämisjärjestykseen, jolloin esimerkiksi ruudunlukuohjelmien käyttäjillä voi olla vaikeuksia ymmärtää uuden sisällön konteksti (Understanding Success Criterion 2.4.3: Focus Order n.d.). Lisäksi ainakin Microsoft Edge -verkkoselain ei vaikuta tekevän Google Chromen- ja Mozilla Firefoxin -verkkoselaimien kaltaista fokuksen hallintaa, jolloin fokusoidun elementin poistuessa

HTML-dokumentista fokus usein menetetään ja sivun selaaminen täytyy aloittaa uudestaan sivun alusta. Selaamisen aloittaminen aina uudestaan sivun alusta ei välttämättä osoittaudu ongelmaksi näppäimistö pohjaiselle selailulle, mikäli verkkosivu tarjoaa sivun rakenteen kannalta järkevän, A-tason onnistumiskriteerin *2.4.1: Bypass Blocks* mukaisen sisällönohittamismekanismiin (Understanding Success Criterion 2.4.1: Bypass Blocks n.d.). Tarkastelluista sivustoista Instagram ja Netflix eivät kuitenkaan kyseistä mekanismeista vaikuta tarjoavan. Twitter ja Reddit -sivustot tarjoavat vastaavan mekanismin näppäimistö komennoilla, mutta näiden verkkosivukohtaisten pikänäppäimien käyttäminen voidaan kokea ongelmalliseksi avustavien teknologioiden omien komentojen ohella. Kyseiset näppäimistö komennot eivät myöskään vaikuta olevan toteutettu A-tason onnistumiskriteerin *2.1.4: Character Key Shortcuts* mukaisesti, sillä näppäimistö komennot toimivat globaalisti eikä niitä ilmeisesti pysty uudelleenmäärittämään tai poistamaan käytöstä. (Understanding Success Criterion 2.1.4: Character Key Shortcuts n.d.)

Netflix-sivuston päänäkymien välillä siirtyessä verkkosivun otsikkoa eli HTML-dokumentin title-elementtiä, ei päivitetä kuvaamaan sivun nykyistä sisältöä, joka voidaan katsoa ongelmalliseksi A-tason onnistumiskriteerin *2.4.2: Page Titled* kannalta. Mikäli HTML-dokumentin otsikko ei kuvaa sivun nykyistä sisältöä tai tarkoitusta, voi osa käyttäjistä kokea haastavaksi tunnistaa kyseinen verkkosivu esimerkiksi verkkoselaimen välilehdistä. Onnistumiskriteerin *2.4.2: Page Titled* tarkemmassa kuvauksessa kuitenkin määritellään, että web-sovellukselle otsikoksi riittää itse sovelluksen nimi, mutta koska vaatimukset verkkosivun määrittelemiseksi web-sovellukseksi ovat epäselvät, jättää tämä onnistumiskriteeri hieman tulkinnanvaraa. (Understanding Success Criterion 2.4.2: Page Titled n.d.)

### **Käsiteltävät onnistumiskriteerit**

SPA-arkkitehtuurille olennaisten onnistumiskriteerien rajaaminen on haastavaa, sillä WCAG 2.1 ei suoranaisesti määrittele SPA-arkkitehtuurin kaltaiselle toiminnallisuudelle saavutettavaa toimintatapaa (Web Content Accessibility Guidelines (WCAG) 2.1 2018). Käsiteltäviksi onnistumiskriteereiksi (ks. Taulukko 4) on pyritty valitsemaan kriteerit, joiden käsitteleminen palvelisi SPA-arkkitehtuurin toiminnallisuutta ja tutkimuksen tarkoitusta parhaiten.

Taulukko 4. SPA-arkkitehtuurin käsiteltävät onnistumiskriteerit

Onnistumiskriteeri	Kriteerin taso
<b>1.3.1: Info and Relationships</b>	A
<b>2.4.2: Page Titled</b>	A
<b>2.4.3: Focus Order</b>	A
<b>2.4.7: Focus Visible</b>	AA
<b>4.1.3: Status Messages</b>	AA

## 5.2 Modaalinen dialogi

Modaali, eli modaalinen dialogi tai ikkuna (engl. modal dialog, modal window), voi olla ongelmallinen erityisesti ruudunlukuohjelmien ja näppäimistön käyttäjille. Mikäli modaalien toteutus on puutteellista, avustavat teknologiat eivät välttämättä saa tietoa modaalien esiintymisestä, jolloin verkkosivun käyttäminen voi olla sekavaa. Modaalien tulisi myös estää interaktio dialogin ulkopuolisen sisällön kanssa, kunnes modaalien odottamaan vuorovaikutukseen on vastattu. Jos fokusta (ks. sivu 15) ei onnistuneesti rajoiteta modaalisen dialogin sisälle, saattavat etenkin ruudunlukuohjelmien käyttäjät jatkaa verkkosivun selaamista, vaikka avautunut modaali odottaisi käyttäjältä jotain tärkeää syötettä. Modaalien puutteellisen fokuksen hallitsemisen seurauksena myös näppäimistökäyttäjillä voi olla vaikeuksia vuorovaikuttaa modaalien kanssa (Understanding Success Criterion 2.4.3: Focus Order n.d.).

### Havainnot verkkosivustoilta

Facebook-sivustolla fokusta ei modaalista suljettaessa palauteta takaisin siihen elementtiin mistä modaali avattiin, vaan fokus siirtyy takaisin sivun alkuun, jolloin näppäimistökäyttäjien on aloitettava sivun selaaminen aina uudestaan sivun alusta.

Tämä voi ymmärrettävästi olla käyttäjälle rasittavaa, mikäli sisältöä on paljon.

Reddit-sivustolla fokuksen siirtämistä ei rajoiteta auki olevan modaalien sisälle, vaan esimerkiksi sarkainnäppäintä hyödyntäen käyttäjä pystyy valitsemaan modaalien ta-

kana olevia, piilotettuja elementtejä. Tässä tapauksessa fokuksen rajoittamatta jättäminen laiminlyö A-tason onnistumiskriteeriä 2.4.3. *Focus Order* (Understanding Success Criterion 2.4.3: Focus Order n.d.).

Instagram- ja Reddit-sivustoilla modaalien avautumisen ilmoittaminen oli vajavaista, eikä millään testatulla ruudunlukuohjelma- ja verkkoselainkombinaatiolla (ks. Taulukko 3) saanut helposti ymmärrettävää tietoa, että käyttöliittymään avautui dialogi. Sivustojen HTML-rakennetta tarkastellessa vaikuttaa siltä, että puutteellisuus aiheutuu dialogin tarkoitusta ilmaisevien attribuutin puutteesta, joka laiminlyö A-tason onnistumiskriteeriä 1.3.1: *Info and Relationships* (Understanding Success Criterion 1.3.1: Info and Relationships n.d.).

### Käsiteltävät onnistumiskriteerit

Taulukko 5. Modaalin käsiteltävät onnistumiskriteerit

Onnistumiskriteeri	Kriteerin taso
<b>1.3.1: Info and Relationships</b>	A
<b>2.1.2: No Keyboard Trap</b>	A
<b>2.4.3: Focus Order</b>	A
<b>4.1.2: Name, Role, Value</b>	A

## 5.3 Ääretön vieritys

Ääretöntä vieritystä (engl. infinite scrolling) hyödyntäessä ongelmallisiksi tilanteiksi voi muodostua muun muassa sisältömuutoksista viestiminen, ääretöntä sisältöä seuraavan sisällön käyttäminen sekä näennäisesti loputtoman sisällön selaamisen rasitus näppäimistöperusteisessa selaamisessa.

Äärettömän vierityksen kanssa joskus hyödynnetään myös virtuaalista vieritystä (engl. virtual scrolling), jolloin vieritettävän elementin HTML-rakenteessa käytetään vain näkymän täyttämiseen tarvittava määrä lapsielementtejä kerrallaan, jolla pyritään parantamaan suorituskykyä silloin kun sisältöä on paljon. Tässä tutkimuksessa ei kyseistä tekniikkaa kuitenkaan hyödynnetä, sillä se oletettavasti lisäisi ratkaisujen

monimutkaisuutta seikoissa, jotka eivät välttämättä ole olennaisia tutkimuksen tarkoitukselle. Samat saavutettavuusseikat kuitenkin pätevät virtuaalista vieritystä hyödynnettäessä.

### Havainnot verkkosivustoilta

Instagram-sivuston profiilinäkymässä on HTML-alatunniste (engl. footer), mutta sen visuaalinen käyttäminen edellyttää selattavan käyttäjäprofiilin kaikkien julkaisujen lataamista, sillä julkaisuja ladataan automaattisesti lisää vieritettäessä sivua alaspäin, jolloin alatunniste siirtyy aina pois näkymästä. Kyseinen alatunniste sisältää monia käyttäjälle mahdollisesti hyödyllisiä linkkejä ja ominaisuuksia, kuten esimerkiksi linkin tukisivulle ja valikon sivuston käyttökielen vaihtamiselle. Vastaava ongelma esiintyy myös Netflix-sivustolla. Ongelman voisi katsoa laiminlyövänsä ainakin A-tason onnistumiskriteeriä 3.2.1: *On Focus* ja AAA-tason onnistumiskriteeriä 3.2.5: *Change on Request* (Understanding Success Criterion 3.2.1: On Focus n.d.; Understanding Success Criterion 3.2.5: Change On Request n.d.).

### Käsiteltävät onnistumiskriteerit

Taulukko 6. Äärettömän vierityksen käsiteltävät onnistumiskriteerit

Onnistumiskriteeri	Kriteerin taso
<b>1.3.1: Info and Relationships</b>	A
<b>2.1.4: Character Key Shortcuts</b>	A
<b>4.1.1: Parsing</b>	A
<b>4.1.2: Name, Role, Value</b>	A

## 5.4 Yhteenveto

Alla oleva taulukko (ks. Taulukko 7) esittää kaikki tutkimuksessa käsiteltävät onnistumiskriteerit ja niiden päällekkäisyydet käsiteltävien käyttöliittymäelementtien ja -toiminnallisuuksien välillä. Käsiteltävien onnistumiskriteerien päällekkäisyys on tutkimuksen tavoitteiden kannalta suotavaa, sillä tutkimuksessa pyritään uudelleenkäytettävien ratkaisujen tuottamiseen. Myös muita olennaisia onnistumiskriteerejä

pyritään ottamaan esimerkki-implementaatioissa huomioon, mutta niitä ei välttämättä käsitellä erikseen.

Taulukko 7. Yhteenveto käsiteltävistä onnistumiskriteereistä

Onnistumiskriteeri	Kriteerin taso	SPA-arkkitehtuuri	Modaali	Ääretön vieritys
<b>1.3.1: Info and Relationships</b>	A	X	X	X
<b>2.1.2: No Keyboard Trap</b>	A		X	
<b>2.1.4: Character Key Shortcuts</b>	A			X
<b>2.4.2: Page Titled</b>	A	X		
<b>2.4.3: Focus Order</b>	A	X	X	
<b>2.4.7: Focus Visible</b>	AA	X		
<b>4.1.1: Parsing</b>	A			X
<b>4.1.2: Name, Role, Value</b>	A		X	X
<b>4.1.3: Status Messages</b>	AA	X		

## 6 Saavutettavuusongelmien korjaaminen ja välttäminen

Esimerkki-implementaatioiden käsittelyssä on tavoitteena käsitellä valittujen onnistumiskriteerien (ks. Taulukko 7) saavutettavuusseikat, eikä esimerkiksi keskittyä siihen kuinka modaalinen dialogi tai ääretön vieritys voidaan implementoida. Käsiteltävistä käyttöliittymäelementeistä ja -toiminnallisuuksista luodaan yksinkertaiset, mutta tositilanteen perustarpeita vastaavat implementaatiot. Implementaatioissa voidaan hyödyntää kolmannen osapuolen kirjastoja edellyttäen, että kirjastojen hyödyntäminen ei ole suoraan olennaista käsiteltävien saavutettavuusongelmien ratkaisemiselle. Ratkaisujen lähdekoodissa käytetään englannin kieltä, jolla pyritään yhtenäisyyteen käytettävien englanninkielisten ohjelmointikielten, termien ja dokumentaatioiden kanssa. Useat tässä opinnäytetyössä esitettävistä koodinäytteistä on yksinkertaistettu, muokattu, syntaksiltaan tiivistetty ja/tai irrotettu niiden todellisesta kontekstista, jotta ne on saatu paremmin sopimaan tämän dokumentin muotoon ja



paremmin tuomaan esiin juuri käsiteltävälle aiheelle olennaiset asiat. Esimerkkisovelluksen koko lähdekoodia ei myöskään käydä läpi tässä dokumentissa. Mikäli haluaa saada paremman käsityksen sovelluksesta ja sen lähdekoodista, kannattaa tutustua sovellukseen verkko-osoitteessa <https://joonaspilli.github.io/thesis-accessible-spa/> ja sen lähdekoodiin verkko-osoitteessa <https://github.com/joonaspilli/thesis-accessible-spa/tree/master/angular/accessibility-example>.

Implementaatioiden toimivuutta testataan seuraavaan taulukkoon (ks. Taulukko 8) listattujen ohjelmistojen avulla. Mahdolliset olennaiset eroavaisuudet tai puutteet ratkaisujen toimivuudessa eri ohjelmistojen välillä pyritään tuomaan esiin.

Taulukko 8. Esimerkki-implementaatioiden testaamiseen käytettävät ohjelmistot

Ohjelma	Versionumero
Google Chrome	76.0.3809.132 (Official Build) (64-bit)
Microsoft Edge	44.17763.1.0
Microsoft Narrator	10.0.17763.1
Microsoft Windows 10 Home	10.0.17763 Build 17763
Mozilla Firefox	69.0 (64-bit)
NVDA	2019.2

### Kehitysympäristö

Esimerkkisovelluksen kehitysympäristö pyritään pitämään yksinkertaisena ja helposti jäljiteltävänä. Myös käytettävissä teknologioissa pyritään suosimaan yksinkertaisempia vaihtoehtoja edellyttäen, että ne ovat käytännöllisiä myös tositilanteessa. Esimerkiksi sovelluksen tyylitiedostoissa käytetään CSS-tyyliohjekieltä suositun SCSS-kielen sijaan, jotta SCSS-kielen osaaminen ei ole edellytys esimerkkien ymmärtämiseksi.

Tutkimuksen esimerkkisovellus luodaan Angularin virallisella komentorivikäyttöliittymällä Angular CLI, josta käytetään versionumeroa 8.3.1. Angular CLI on saatavilla esimerkiksi npm-paketinhallintajärjestelmästä nimellä `@angular/cli`. Angularista itses-

tään käytetään versiota 8.2.3. SPA-arkkitehtuurin soveltamista varten projektiin otetaan käyttöön Angularin oma reititysmoduuli RouterModule, jota nykyinen Angular CLI kysyykin projektia luodessa.

## 6.1 SPA-arkkitehtuuri

*Web Content Accessibility Guidelines (WCAG) 2.1* ja sitä tukevat dokumentit eivät nykyisellään suoraan ohjeista, miten SPA-arkkitehtuuria voitaisiin soveltaa saavutettavasti. Useat onnistumiskriteerit ja ohjeistukset voidaan kuitenkin katsoa enemmän tai vähemmän SPA-arkkitehtuurille olennaisiksi, mutta toiminnan logiikan päättäminen on pitkälti verkkosivujen kehittäjien vastuulla. (Web Content Accessibility Guidelines (WCAG) 2.1 2018.)

### 6.1.1 Fokuksen hallinta

SPA-arkkitehtuurissa ongelmalliseksi osoittautuu etenkin fokuksen hallitseminen (ks. sivu 15). A-tason onnistumiskriteerin 2.4.3: *Focus Order* voidaan katsoa ottavan kantaa SPA-arkkitehtuurin fokuksen hallintaan, sillä kriteeri ohjeistaa, että fokuksen siirtäminen tulisi tapahtua loogisessa järjestyksessä (Understanding Success Criterion 2.4.3: Focus Order n.d.). SPA-sovelluksille tyypillisen dynaamisen HTML:n seurauksena elementtejä lisätään ja poistetaan sivun HTML-rakenteesta sivun vaihtamisen yhteydessä. *Accessible Rich Internet Applications (WAI-ARIA) 1.2* -dokumentin virallinen **luonnos** nykyisellään määrittelee, että mikäli fokusoitu elementti poistetaan HTML-rakenteesta, tulisi fokus siirtää johonkin loogiseen elementtiin (Accessible Rich Internet Applications (WAI-ARIA) 1.2 2019). *WAI-ARIA 1.2* -dokumentin ohjeistus on toistaiseksi vain luonnos eli avoin muutoksille, mutta se tukee 2.4.3: *Focus Order* -onnistumiskriteerin tarjoamaa ohjeistusta hyvin.

Perinteisten, erillisten verkkosivujen välillä siirtyessä fokus yleensä siirtyy sivun alkuun, HTML-dokumentin body-elementtiin. Samantapaista käytöstä voidaan jäljitellä myös SPA-sovelluksissa, mutta on kohtuullista kyseenalaistaa, onko sen jäljitteleminen loogisin ja optimaalisin ratkaisu. Perinteisessä sivun vaihtamisessa fokuksen käyttäytyminen vaikuttaa olosuhteiden pakottamalta ratkaisulta, sillä seuraavan

verkkosivun sisältöä ja sen loogista fokusointikohdetta voi olla teknisesti vaikea ennakoida. SPA-sovelluksissa sivun vaihtamiseen voidaan sovelluksen puolesta reagoida ja seuraavan sivun rakennetta ja sisältöä voidaan ennakoida, mikäli sovelluksen sivujen välillä säilytetään johdonmukainen rakenne.

Toimintatavaltaan SPA-sovellusten sivuja voitaisiin myös verrata välilehditettyyn sisältöön, jonka implementoimiseen W3C:n materiaalit tarjoavat kiitettävästi ohjeistusta (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017; WAI-ARIA Authoring Practices 1.1 2019). Sivunavigoimisen toteuttaminen välilehditetyn sisällön elementeillä ja attribuuteilla voi tietyn tyyppisissä web-sovelluksissa paremmin kuvata kyseisen toiminnallisuuden tarkoitusta, esimerkiksi jos web-sovelluksen sivut mielletään enemmänkin näkymiksi kuin sivuiksi. Kuitenkin jos kyse on erillisistä käytännössä toisistaan riippumattomista verkkosivuista, voi perinteinen navigaatio paremmin kuvata sen linkkien käyttötarkoitusta. Mikäli sovelluksen päänäköymien vaihtaminen nähdään tarpeelliseksi toteuttaa välilehdillä, tulisi sovelluksen välilehtien fokuksen hallintaan ja pikanäppäintoiminnallisuuksiin kiinnittää erityistä huomiota (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Toteutettavan esimerkkisovelluksen sivut kuitenkin mielletään luonteeltaan perinteisemmiksi verkkosivuiksi, joten välilehtien implementointia ei tässä tutkimuksessa käsitellä.

Nykyisten W3C:n saavutettavuusohjeistuksien perusteella on nähtävissä kaksi loogista ratkaisua fokuksen käsittelemiselle SPA-sovelluksen sivun vaihtamisen yhteydessä. Sivun vaihtamisen yhteydessä fokus siirretään joko sivun alkuun tai uutta sisältöä edeltävään, sitä lähellä olevaan ja sitä parhaiten kuvaavaan elementtiin. Seuraavassa listassa esitetään mietteitä molemmista vaihtoehdoista.

- Fokuksen siirtäminen sivun alkuun:
  - Onnistuneen, A-tason onnistumiskriteerin *2.4.1: Bypass Blocks* mukaisen sisällön ohittamismekanismien tarve korostuu.
  - Jäljittelee perinteisten verkkosivujen välillä siirtymistä, jolloin käytös on ennalta-arvattavaa.
  - Uuteen sisältöön pääseminen vaatii näppäimistöä hyödyntäviltä käyttäjiltä todennäköisesti enemmän välivaiheita.

- Fokuksen siirtäminen uutta sisältöä edeltävään, sitä lähellä olevaan ja sitä parhaiten kuvaavaan elementtiin:
  - Käyttäjällä on todennäköisesti nopeampi pääsy haluamaansa sisältöön (uuteen pääsisältöön).
  - Sivustoon tutustumaton käyttäjä ei voi ennakoida käytöstä, joka voi olla käyttäjälle aluksi sekavaa.
  - Toiminnallisuus asettaa pakotteita sisällön sivujen rakenteiden yhtenäistämiseksi, jotka hyvin toteutettuna edistävät saavutettavuutta.
  - Uuden sisällön tarkoitus ja konteksti voidaan selkeästi viestiä etenkin ruudunlukuohjelmien käyttäjille.

On myös mahdollista siirtää fokus uutta sisältöä käärivään isäntäelementtiin, mutta testaamisen perusteella ruudunlukuohjelmat alkavat tällöin lukemaan kyseisen elementin koko sisällön, joka voi olla käyttäjälle sekavaa, etenkin jos sisältöä on paljon. Sivun vaihtamisen jälkeen osa uusista käyttöliittymäelementeistä saattaa myös jatkaa oman sisältönsä lataamista asynkronisesti, jolloin testaamisen perusteella ruudunlukuohjelmat saattavat kyseisiin elementteihin päästyään lukea vanhentunutta tietoa, mikäli ne ohjeistetaan kerralla lukemaan kaiken uudesta sisällöstä.

Tässä tutkimuksessa SPA-arkkitehtuurin fokuksen hallinnan lähestymistavaksi valitaan fokuksen siirtäminen uutta sisältöä edeltävään, sitä lähellä olevaan ja sitä parhaiten kuvaavaan elementtiin, minkä näkisin todennäköisesti parhaiten palvelevan käyttäjien yleisempiä sisällön selaamistarpeita. Esimerkki-implementaatioissa, sivun vaihtamisen yhteydessä fokuksittavan, loogisen elementin roolissa toimii sivukohtaisesti päivittyvä otsikkoelementti, joka lyhyesti kuvaa sivun pääsisällön tarkoitusta. Valitussa lähestymistavassa on myös oleellista, että fokus siirretään aina samaan otsikkoelementtiin, vaikkei navigoinnin aloittanutta elementtiä poistettaisi HTML-dokumentista, sillä *WAI-ARIA Authoring Practices 1.1* -dokumentti ohjeistaa, että käyttäjän toiminnasta aiheutuvan fokuksen siirtämisen tulisi toimia ennalta-arvattavasti (*WAI-ARIA Authoring Practices 1.1* 2019). On kuitenkin hyvä tiedostaa, että absoluuttisen oikeaa tapaa SPA-arkkitehtuurin kaltaiselle toiminnallisuudelle ei W3C:n

materiaaleissa toistaiseksi määritellä, joten verkkosivun kehittäjien tulisi tutkia ja vertailla eri ratkaisumahdollisuuksia kehitettävän sovelluksen rakennetta, toimintoja ja sisällön tarpeita huomioiden.

### **Yhtenäisen rakenteen noudattaminen**

Jotta fokus voitaisiin sivun vaihtamisen yhteydessä aina siirtää uuden sivun sisältöä kuvaavaan otsikkoelementtiin, on sovelluksen hallittavuuden kannalta järkevää pakottaa käyttöliittymään tarvittavan rakenteen noudattamista, jotta sovelluksessa ei olisi sivuja, eli reittejä (engl. routes), jotka eivät omaa niiden tarkoitusta kuvaavaa otsikkoa. Angular-sovelluksien kehittämisessä hyödynnettävä TypeScript mahdollistaa erilaisten tietotyyppien hyödyntämistä, joten halutun rakenteen noudattaminen aloitetaan luomalla uusi rajapinta `AccessibleRoute` (ks. Kuvio 1), joka perii Angularin reitityspaketin `Route`-rajapintaa. Luodulla rajapinnalla pakotetaan otsikointitietojen määrittäminen kaikille sovelluksen reiteille.

```

import { Data, Route } from '@angular/router';

export enum DocumentTitleSection {
  Application = 'application',
  Modifier = 'modifier',
  Notification = 'notification',
  Title = 'title'
}

export type DocumentTitleSectionData =
  Partial<Record<DocumentTitleSection, string>>;

export type DocumentTitleDelimiterData =
  Partial<Record<DocumentTitleSection, string>>;

export interface DocumentTitleData {
  delimiters?: DocumentTitleDelimiterData;
  sections?: DocumentTitleSectionData;
}

export interface RouteTitleData extends DocumentTitleData {
  sections: DocumentTitleSectionData &
    Required<Pick<DocumentTitleSectionData, DocumentTitleSection.Title>>;
}

export interface RouteAccessibilityData {
  heading?: string;
  title: RouteTitleData;
}

export interface AccessibleRouteData extends Data {
  accessibility: RouteAccessibilityData;
}

export interface AccessibleRoute extends Route {
  children?: AccessibleRoutes;
  data: AccessibleRouteData;
}

export type AccessibleRoutes = AccessibleRoute[];

```

Kuvio 1. Esimerkkisovelluksen reittien pääsisällön otsikoimisessa hyödynnettävät tietotyypit

Luoduissa tietotyypeissä määritellään, että `AccessibleRoute`-rajapintaa toteuttavan reitin datarakenteesta tulee löytyä `RouteTitleData`-rajapintaa toteuttava `title`-ominaisuus, josta tulee vähintäänkin löytyä reittiä kuvaava otsikko. Reittien otsikointidataa tullaan käyttämään reittien pääsisällön otsikon määrittämiseen sekä HTML-dokumentin otsikoimiseen (ks. sivu 33). Reitin datarakenteeseen voidaan myös vaihtoehtoisesti suoraan määritellä `string`-muotoinen `heading`-ominaisuus, jota tullaan käyttämään sellaisenaan pääsisältöä kuvaavassa otsikkoelementissä, mikäli pääsisällön otsikko tarvitsee olla eri kuin HTML-dokumentin otsikko. Luotuja tietotyyppejä hyödynnetään esimerkkisovelluksen pääreititysmoduulissa, jossa myös määritellään sovellukselle kolme reittiä (ks. Kuvio 2).

```

const ROUTES: AccessibleRoutes = [{
  path: 'profile',
  component: ProfileComponent,
  data: {
    accessibility: {
      title: {
        sections: { title: 'Profile' },
        delimiters: { modifier: ' of ' }
      }
    }
  }
}, {
  path: 'help',
  component: HelpCenterComponent,
  data: {
    accessibility: {
      title: {
        sections: { title: 'Help Center' }
      }
    }
  }
}, {
  path: '',
  component: HomeComponent,
  data: {
    accessibility: {
      title: {
        sections: {
          title: 'Home',
          modifier: 'Activity Feed'
        }
      }
    },
    heading: 'Your Activity Feed'
  }
}, {
  path: '**',
  redirectTo: ''
} as AccessibleRoute];

@NgModule({
  imports: [RouterModule.forRoot(ROUTES)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Kuvio 2. Näyte esimerkksiovelluksen pääreititysmoduulista

### Pääsisällön otsikon päivittäminen ja fokusoiminen

Esimerkkiovelluksen reiteille määritetyt otsikot halutaan myös esittää käyttäjälle. Sovellukseen luodaan uusi AccessibilityModule-moduuli, joka isännöi erityisesti saatavuuden edistämiseen liittyviä toiminnallisuuksia. Kyseiseen moduulin luodaan reittien otsikointidataa käsittelevä RoutePurposeService-palvelu (engl. service) (ks. Kuvio 3).

```

@Injectable({ providedIn: 'root' })
export class RoutePurposeService {

  public get mainHeading(): Observable<string> {
    return this.mainHeading$.asObservable();
  }

  private readonly mainHeading$: BehaviorSubject<string> =
    new BehaviorSubject('');

  constructor(
    private readonly router: Router
  ) {
    this.enableAutomaticHeadingUpdating();
  }

  public enableAutomaticHeadingUpdating(): void {
    onPathChange(this.router).pipe(
      switchMap((route: ActivatedRoute) => route.data),
      map((data: AccessibleRouteData) => data.accessibility)
    ).subscribe(this.handleNavigationHeadingChange.bind(this));
  }

  public setMainHeading(heading: string): void {
    this.mainHeading$.next(heading);
  }

  private handleNavigationHeadingChange(data: RouteAccessibilityData): void {
    this.setMainHeading(data.heading || data.title.sections.title);
  }
}

```

Kuvio 3. Näyte reittien pääsisällön otsikon päivittämisestä RoutePurposeService-palvelussa

RoutePurposeService-palvelussa pääsisällön otsikon päivittäminen otetaan käyttöön kutsumalla palvelun enableAutomaticHeadingUpdating-metodia, jossa reagoidaan sovelluksen reitittimen navigointitapahtumiin onPathChange-apufunktion (ks. Kuvio 4) välityksellä. Kyseinen työkalufunktio palauttaa RxJS-kirjaston määrittelemän Observablen, joka välittää tapahtumia vain, jos reitin polku tai parametrit muuttuvat. Tällöin otsikon päivittämistä ei käsitellä tilanteissa, joissa se ei ole esimerkkisovellukselle tarpeellista, kuten esimerkiksi käyttäjän navigoidessa sovelluksen saman reitin ankkurilinkkien välillä. Ankkurilinkkinavigoimista voi tapahtua esimerkiksi A-tason onnistumiskriteerin 2.4.1: *Bypass Blocks* mukaisen sisällönohittamismekanismin implementoimisen seurauksena.



```
export const getFragmentlessUrl = (
  router: Router,
  route: ActivatedRoute|null = router.routerState.root.firstChild
): string|undefined => {
  if (route) {
    const url = router.url;
    const fragment = route.snapshot.fragment;
    const cut = fragment ? -fragment.length - 1 : 0;
    return cut ? url.slice(0, cut) : url;
  }
};

export const onPathChange = (router: Router): Observable<ActivatedRoute|null> => {
  return router.events.pipe(
    filter((event: Event) => event instanceof NavigationEnd),
    map(() => getFragmentlessUrl(router)),
    distinctUntilChanged(),
    map(() => router.routerState.root.firstChild)
  );
};
```

#### Kuvio 4. Esimerkkisovelluksen onPathChange-apufunktio

RoutePurposeService-palvelun (ks. Kuvio 3) mainHeading-jäsen määritetään julkiseksi, jolloin sen välittämää otsikkoa voidaan esittää AccessibilityModule-moduuliin lisättävässä MainContentComponent-komponentissa (ks. Kuvio 5).

```

@Component({
  selector: 'app-main-content',
  template: `
    <main aria-labelledby="main-content-heading">
      <h2 #mainContentHeadingEl id="main-content-heading" tabindex="-1">
        {{ mainContentHeading$ | async }}
      </h2>
      <ng-content></ng-content>
    </main>
  `
})
export class MainContentComponent implements OnInit, OnDestroy {

  private static readonly FOCUS_DEBOUNCE: number = 100;

  public readonly mainContentHeading$: Observable<string> =
    this.routePurposeService.mainHeading;

  private readonly destroyed$: Subject<boolean> = new Subject();
  @ViewChild('mainContentHeadingEl', { static: true })
  private readonly mainContentHeadingEl: ElementRef;

  constructor(
    private readonly routePurposeService: RoutePurposeService,
    private readonly router: Router
  ) { }

  public ngOnDestroy(): void {
    this.destroyed$.next(true);
  }

  public ngOnInit(): void {
    onUrlChange(this.router).pipe(
      takeUntil(this.destroyed$),
      withLatestFrom(this.router.events),
      filter(([ route, routerEvent ]: [ ActivatedRoute, Event ]) => (
        routerEvent instanceof NavigationEnd && routerEvent.id > 1)
      ),
      debounceTime(MainContentComponent.FOCUS_DEBOUNCE)
    ).subscribe(() => this.mainContentHeadingEl.nativeElement.focus());
  }
}

```

Kuvio 5. Esimerkkisovelluksen MainContentComponent-komponentti

MainContentComponent-komponentissa reagoidaan sovelluksen reitittimen onnistuneisiin sivun vaihtoihin, joiden seurauksena fokusoidaan komponentin HTML-rakenteen h2-otsikkoelementti. Jotta HTML-elementti, joka ei oletuksena salli fokusointia, voitaisiin fokusoida kutsumalla sen focus-metodia, tulee elementille määritellä tabindex-attribuutti. Määritettäessä elementin tabindex-attribuutin arvoksi ”-1”, pystyy elementin fokusoidaan JavaScriptillä tai osoittimella, mutta ei näppäimistöllä, joka ei tässä tilanteessa olekaan tarpeellista. (WAI-ARIA Authoring Practices 1.1 2019.) On myös hyvä huomioida, että komponentissa reagoitavista navigointitapah- tumista suodatetaan pois sovelluksen ensimmäinen navigointi eli Angular-sovelluk- sen käynnistyminen, sillä *WAI-ARIA Authoring Practices 1.1* ohjeistaa yleisemmissä tilanteissa välttämään elementtien fokusoiduista heti HTML-dokumentin lataamisen

jälkeen (WAI-ARIA Authoring Practices 1.1 2017). Otsikon fokusoimisen voisi toteuttaa myös reagoimalla RoutePurposeService-palvelun mainHeading-observablen tapahtumiin, mutta esimerkkisovelluksessa pääotsikon muokkaaminen mahdollistetaan myös suoraan komponenteista, minkä seurauksena pääotsikon fokusoiminen ei luultavasti ole tarpeellista.

Olennaista implementoidussa otsikkoelementin fokusoimisessa on myös fokusoimisen viivästyttäminen. Ilman pientä viivästystä (esimerkissä 100 ms) otsikkoelementin fokusoimiseen, testaamisen käytetyt ruudunlukuohjelmat päätyivät lukemaan otsikon edellisen arvon eli edellisen sivun otsikon. Vaihtoehtoisesti saman ongelman voi osittain korjata kutsumalla komponentin ChangeDetectorRef-viitteen detectChanges-metodia ennen otsikkoelementin fokusoimista, mutta testaamisen perusteella kyseinen ratkaisu ei toimi luotettavasti eri ruudunlukuohjelma- ja verkkoselainkombinaatioiden välillä.

Kun fokus siirtyy MainContentComponent-komponentin HTML-rakenteen otsikkoelementtiin, elementin visuaalinen tyyli vaihtuu useimmissa selaimissa osoittamaan, että fokus on kyseisen elementin kohdalla. Visuaalisista tyylyisistä tämä voi olla joillekin projekteille epätoivottavaa, eikä kyseistä otsikkoelementtiä haluta välttämättä näyttää visuaalisesti ollenkaan. AA-tason onnistumiskriteeri 2.4.7: *Focus Visible* kuitenkin ohjeistaa, että kaikilla näppäimistökäyttöisillä käyttöliittymillä tulisi visuaalisesti näkyvä näppäimistöfokuksen ilmaisu, joten koska sovelluksen otsikkoelementti fokusoidaan, tulee se ja sen fokuksen ilmaisu tyyli olla visuaalisesti näkyvissä (Understanding Success Criterion 2.4.7: Focus Visible n.d.).

MainContentComponent-komponentin HTML-rakenteessa käytetään main-elementtiä, jolla viestitään erityisesti avustavien teknologioiden käyttäjille, että kyseisen elementin sisältö on dokumentin pääsisältö. Main-elementtiä, tai sitä vastaavaa role="main" -HTML-attribuutin omaavaa elementtiä, suositellaan HTML-dokumentin rakenteessa käytettävän vain kerran. Esimerkkisovelluksessa main-elementti myös nimetään aria-labelledby-attribuutin avulla, mikä voi auttaa avustavia teknologioita hyödyntäviä käyttäjiä tunnistamaan kyseinen elementti ja sen tarkoitus. Main-rooli

on yksi *WAI-ARIA 1.1* -dokumentin määrittämistä maamerkkialueista, joita käyttämällä voidaan helpottaa muun muassa ruudunlukuohjelmakäyttäjien verkkoselaimista. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.)

Lopuksi MainContentComponent-komponentti otetaan käyttöön esimerkksiovelluksen juurikomponentin HTML-rakenteessa ja sovelluksen pääreitittimen RouterOutlet-direktiivi sijoitetaan app-main-content-valitsimen (engl. selector) sisälle (ks. Kuvio 6), jolloin sovelluksen reitittimen ulostulo sijoitetaan MainContentComponent-komponentin HTML-rakenteen (ks. Kuvio 5) main-elementin sisältämän ng-content-tunnisteen paikalle.

```
<app-main-content>  
  <router-outlet></router-outlet>  
</app-main-content>
```

Kuvio 6. MainContentComponent-komponentin käyttöönotto esimerkksiovelluksen juurikomponentin HTML-rakenteessa

### 6.1.2 HTML-dokumentin otsikon päivittäminen

A-tason onnistumiskriteeri *2.4.2: Page Titled* ohjeistaa, että verkkosivuilla tulisi olla niiden aihetta tai merkitystä kuvaava otsikko. Onnistumiskriteerin tarkemmassa kuvauksessa myös ohjeistetaan, että web-sovelluksille otsikoksi riittää pelkkä sovelluksen nimi. *WCAG 2.1* ei kuitenkaan tarjoa tarkempaa määritelmää web-sovellukselle, jolloin verkkosivun määrittelemine web-sovellukseksi jää tulkinnanvaraiseksi. Mikäli web-sovelluksen HTML-dokumentin otsikkoa kuitenkin päivitetään kuvaamaan sovelluksen nykyistä sisältöä, voidaan käyttäjiä helpottaa tunnistamaan sovellus esimerkiksi verkkoselaimen välilehdistä. (Understanding Success Criterion 2.4.7: Focus Visible n.d.)

#### **HTML-dokumentin otsikon rakenne**

Esimerkksiovelluksen pääsisällön otsikon päivittämistä varten luotiin tietotyyppiä (ks. Kuvio 1), joista osaa tullaan hyödyntämään myös sovelluksen HTML-dokumentin otsikon päivittämisessä. HTML-dokumentin otsikon hallitsemista varten kuitenkin luodaan vielä muutama tietotyyppi lisää, jotka on koottu seuraavaan kuvioon (ks. Kuvio 7).

```

export type DocumentTitleStructure = DocumentTitleSection[];

export enum DocumentTitleLevel {
  Application = 'application',
  Route = 'route',
  Context = 'context'
}

export type DocumentTitleLevelStructure = DocumentTitleLevel[];

export type DocumentTitleDescription = Partial<Record<DocumentTitleLevel, DocumentTitleData>>;

export class DocumentTitle {
  private readonly defaultTitleData: DocumentTitleData = {
    delimiters: { notification: ' ', title: ' ', modifier: ' - ', application: ' | ' }
  };
  private digestedData: DocumentTitleData|null = null;
  private readonly titleLevelStructure: DocumentTitleLevelStructure = [
    DocumentTitleLevel.Application,
    DocumentTitleLevel.Route,
    DocumentTitleLevel.Context
  ];
  private readonly titleStructure: DocumentTitleStructure = [
    DocumentTitleSection.Notification, DocumentTitleSection.Title,
    DocumentTitleSection.Modifier, DocumentTitleSection.Application
  ];

  constructor(private readonly data: DocumentTitleDescription = {}) { }

  public getTitle(include: DocumentTitleStructure = this.titleStructure): string {
    const data = this.digestedData || (this.digestedData = this.digestData());
    const { sections, delimiters } = data;
    if (sections) {
      return this.titleStructure
        .filter((section: DocumentTitleSection) => include.includes(section))
        .reduce((accTitle: string, curSection: DocumentTitleSection) => {
          const section = sections[curSection] || '';
          if (accTitle && delimiters && section) {
            accTitle += delimiters[curSection] || '';
          }
          return accTitle += section;
        }, '');
    } else {
      return '';
    }
  }

  public setTitle(
    level: DocumentTitleLevel,
    data: DocumentTitleData,
    merge: boolean = false
  ): void {
    this.data[level] = merge ? this.mergeData(this.data[level], data) : data;
    this.digestedData = null;
  }

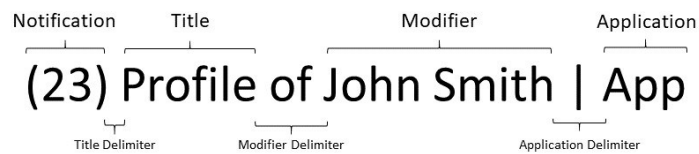
  private digestData(): DocumentTitleData {
    return this.mergeData(
      { ...this.defaultTitleData },
      ...this.titleLevelStructure.map((key: string) => this.data[key] || {}),
    );
  }

  private mergeData(...data: (DocumentTitleData|undefined)[]): DocumentTitleData {
    return data.reduce((acc: DocumentTitleData, cur: DocumentTitleData) => ({
      sections: { ...acc.sections, ...cur.sections },
      delimiters: { ...acc.delimiters, ...cur.delimiters }
    })), {});
  }
}

```

Kuvio 7. Esimerkkisovelluksen HTML-dokumentin otsikon hallitsemisessa hyödynnet-  
täviä tietotyypppejä

Esimerkkisovelluksessa on päädytty ratkaisuun, jossa dokumentin otsikko muodostuu neljästä osasta (section); Notification, Title, Modifier ja Application, sekä niiden erottimista (delimiter), mitä havainnollistetaan seuraavassa kuviossa (ks. Kuvio 8). Ratkaisun tarkoituksena on säilyttää yhtenäinen otsikkorakenne koko sovelluksessa, millä pyritään otsikon helppoon ymmärrettävyyteen.



Kuvio 8. Havainnollistus esimerkkisovelluksen HTML-dokumentin otsikon rakenteesta

### **HTML-dokumentin otsikon päivittäminen**

Esimerkkisovelluksessa otsikkoa muokataan RoutePurposeService-palveluun lisättävällä setDocumentTitle-metodilla (ks. Kuvio 9), jolle annettava DocumentTitleData-tietotyyppi (ks. Kuvio 7) muotoinen otsikointidata välitetään DocumentTitle-luokan (ks. Kuvio 7) instanssille käsiteltäväksi. Kyseistä metodia kutsutaan myös automaattisesti sivun vaihtamisen yhteydessä sovelluksen reititysmoduuliin (ks. Kuvio 2) määritetyllä otsikointidatalla. Datasta tuotettu merkkijono asetetaan HTML-dokumentin otsikoksi käyttämällä Angularin platform-browser-paketin Title-palvelua.

```

private readonly documentTitle: DocumentTitle = new DocumentTitle({
  application: { sections: { application: this.titleService.getTitle() } }
});

constructor(private readonly router: Router, private readonly titleService: Title) {
  this.enableAutomaticTitleUpdating();
}

public enableAutomaticTitleUpdating(): void {
  onPathChange(this.router).pipe(
    switchMap((route: ActivatedRoute) => route.data),
    map((data: AccessibleRouteData) => data.accessibility.title)
  ).subscribe(this.handleNavigationTitleChange.bind(this));
}

public getDocumentTitle(): string {
  return this.documentTitle.getTitle();
}

public setDocumentTitle(
  level: DocumentTitleLevel,
  data: DocumentTitleData,
  merge: boolean = false
): void {
  this.documentTitle.setTitle(level, data, merge);
  this.titleService.setTitle(this.documentTitle.getTitle());
}

private handleNavigationTitleChange(data: RouteTitleData): void {
  this.setDocumentTitle(DocumentTitleLevel.Route, data);
}


```

Kuvio 9. Näyte HTML-dokumentin otsikon päivittämisestä esimerkksiovelluksen RoutePurposeService-palvelussa

Esimerkksiovelluksessa HTML-dokumentin otsikon muokkaaminen tapahtuu kolmella eri tasolla; Application, Route ja Context, joille määritetty otsikointidata yhdistetään yhteen, HTML-dokumentin title-elementissä esitettävään, merkkijonoon. Otsikointitasoista Route korvataan jokaisen onnistuneen sivun vaihtamisen yhteydessä uudella reititysmoduulissa määritetyllä otsikointidatalla (ks. Kuvio 2), jolloin se on reittikohtainen. Otsikointitasojen tarkoituksena on helpottaa väliaikaista ja tilannekohtaista otsikonpäivittämistä, sillä kyseisessä ratkaisussa otsikon koko merkkijonoa ei tarvitse määrittää jokaisen päivityksen yhteydessä ja otsikko on helppo palauttaa alempien otsikointitasojen määrittämään dataan. Seuraavassa listassa ja kuviossa (ks. Kuvio 10) käydään lyhyesti läpi esimerkksiovelluksen otsikkotasologiikkaa.

- Context:
  - Tarkoitettu väliaikaiseen otsikon määrittämiseen.
  - Asetetaan yleensä RouterOutletin kirjoittamien komponenttien toimesta.
  - Ylikirjoittaa Route-tason datan.

- Route:
  - Tarkoitettu reittilaajuiseen otsikon määrittämiseen.
  - Asetetaan yleensä automaattisesti RoutePurposeService-palvelussa uuteen reittiin siirryttäessä, käyttäen sovelluksen reititysmoduulissa määriteltyä dataa.
  - Nollataan sivun vaihtamisen yhteydessä.
  - Ylikirjoittaa Application-tason dataa.
- Application:
  - Tarkoitettu sovelluslaajuiseen otsikon määrittämiseen.

	Notification	Title Delimiter	Title	Modifier Delimiter	Modifier	Application Delimiter	Application
 Oletusarvot Application Route Context	(23)		Profile	of	John Smith		App
HTML-dokumentin <title>	(23)		Profile	-	Confirmation dialog		App

Kuvio 10. Havainnollistus esimerkisovelluksen HTML-dokumentin otsikon muodostamisesta otsikkotasolla

### 6.1.3 Sivun vaihdosta ilmoittaminen

AA-tason onnistumiskriteeri *4.1.3: Status Messages* ohjeistaa, että verkkosivun tilaa ja tapahtumia koskevia statusviestejä esittämissä elementeissä tulisi käyttää tarkoituksen mukaisia attribuutteja, jotta niitä voidaan esittää käyttäjälle tavalla, missä statusta ilmaisevaa elementtiä ei tarvitse fokusoida. Tarkoituksena on siis, että käyttäjä voidaan tehdä tietoiseksi verkkosivun tärkeistä tapahtumista ja muutoksista niin, ettei hänen toimintaansa turhaan keskeytetä. Onnistumiskriteerin kohteena ovat erityisesti näkörajoitteiset käyttäjät, jotka selaavat verkkosivua ruudunlukuohjelman avustuksella. (Understanding Success Criterion 4.1.3: Status Messages n.d.) Onnistumiskriteerin tarjoamaa ohjeistusta voidaan soveltaa myös SPA-arkkitehtuurin mukaisesta sivunvaihtumisesta viestimiseen.

#### Statusviestien antaminen

Statusviestien käsittelemistä varten esimerkisovelluksessa otetaan käyttöön seuraavaan kuvioon kootut tietotyypit (ks. Kuvio 11).



```

export enum LiveMessageImportance {
  Assertive = 'assertive',
  Polite = 'polite'
}

export interface LiveMessageOptions {
  isAlert?: boolean;
}

export class LiveMessage implements Required<LiveMessageOptions> {
  public readonly isAlert: boolean = false;
  constructor(
    public message: string,
    public readonly importance: LiveMessageImportance,
    options: LiveMessageOptions = {}
  ) {
    Object.assign(this, options);
  }
}

export class AssertiveStatusMessage extends LiveMessage {
  constructor(message: string, options?: LiveMessageOptions) {
    super(message, LiveMessageImportance.Assertive, options);
  }
}

export class PoliteStatusMessage extends LiveMessage {
  constructor(message: string, options?: LiveMessageOptions) {
    super(message, LiveMessageImportance.Polite, options);
  }
}

export type StatusMessage = LiveMessage|PoliteStatusMessage|AssertiveStatusMessage;

```

Kuvio 11. Esimerkkisovelluksen statusviestien käsittelyssä hyödynnettävät tietotyypit

Esimerkkisovelluksen AccessibilityModule-moduuliin luodaan StatusAnnouncement-Service-palvelu (ks. Kuvio 12), jonka announce-metodia voidaan käyttää statusviestien antamiseen.

```

@Injectable({ providedIn: 'root' })
export class StatusAnnouncementService {

  public get statusAnnouncements(): Observable<StatusMessage> {
    return this.statusAnnouncements$.asObservable();
  }

  private readonly statusAnnouncements$: Subject<StatusMessage> = new Subject();

  public announce(message: StatusMessage): void {
    this.statusAnnouncements$.next(message);
  }
}

```

Kuvio 12. StatusAnnouncementService-palvelu

Jotta avustavat teknologiat pystyisivät lukemaan annetut statusviestit, tulee ne esittää sovelluksen HTML-rakenteessa. AA-tason onnistumiskriteerin 4.1.3: *Status Messages* mukaisia viestejä voidaan antaa määrittämällä statusviestin sisältävälle HTML-

elementille `aria-live`-attribuutti, joka hyväksyy arvokseen `"off"`, `"assertive"` tai `"polite"`. Arvolla `"off"` ilmaistaan, että attribuutin omaavan elementin muutoksista ei tarvitse ilmoittaa käyttäjälle ja arvoilla `"assertive"` ja `"polite"` ilmaistaan muutoksien ilmoittamisen tärkeyttä. Mikäli arvoksi määritellään `"assertive"`, avustavien teknologioiden tulisi ilmoittaa elementissä tapahtuneista muutoksista välittömästi, jonka seurauksena esimerkiksi ruudunlukuohjelman nykyinen puhe saatetaan keskeyttää ja muut jonossa mahdollisesti olevat statusviestit saatetaan jättää ilmoittamatta. Arvo `"polite"` puolestaan ilmaisee, että avustavien teknologioiden tulisi ilmoittaa muutoksista seuraavana otollisena hetkenä. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.) On tärkeää myös huomioida, että mikäli ARIA live -alueita lisätään sivun HTML-rakenteeseen dynaamisesti, avustavat teknologiat eivät välttämättä osaa ilmoittaa elementistä sen lisäyksen yhteydessä. ARIA live -elementit tulisi siis lisätä HTML-rakenteeseen ennen niiden muokkaamista, jotta avustavat teknologiat osaisivat reagoida niihin. (ARIA live regions n.d.)

Esimerkkisovelluksen `AccessibilityModule`-moduuliin luodaan statusviestien esittämiseen käytettävä `StatusAnnouncerComponent`-komponentti, jonka HTML-rakenteeseen (ks. Kuvio 13) luodaan HTML-elementit kullekin `aria-live`-attribuutin mahdolliselle tärkeytasolle.

```
<p
  *ngFor="let messageType of messageTypes; index as i; trackBy: msgTrackBy"
  [attr.role]="statusMessages[messageType]?.isAlert ? 'alert' : 'status'"
  [attr.aria-live]="messageType"
  aria-atomic="true"
  class="a11y-vhidden">
  {{ statusMessages[messageType]?.message }}
</p>
```

Kuvio 13. Esimerkkisovelluksen `StatusAnnouncerComponent`-komponentin HTML-rakenne

`Aria-live`-attribuuttia käytettäessä tulisi myös huomioida sen ohella käytettävä, vapaaehtoinen attribuutti, `aria-atomic`. `Aria-atomic`-attribuutille voidaan määritellä toisuusarvo, `"true"` tai `"false"`, joilla ilmaistaan avustaville teknologioille, tulisiko muutosten tapahtuessa attribuutin omaavan elementin rakenne esittää käyttäjälle kokonaisuudessaan. Arvolla `"true"` avustavat teknologiat ohjeistetaan muutosten jälkeen

ilmoittamaan käyttäjälle muuttuneen elementin koko sisältö. Arvolla "false" sisälöstä ilmoitetaan vain muuttuneet osat. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.) Ainoastaan muuttuneiden osien ilmoittaminen käyttäjälle voi olla hyödyllistä esimerkiksi muokattaessa listaelementtiä, jolloin ei välttämättä ole järkevää lukea käyttäjälle koko listaa uudestaan vaan pelkästään sen muuttuneet osat. Statusviestien aria-atomic-attribuuttien kanssa on kuitenkin syytä useimmissa tilanteissa käyttää arvoa "true". Testaamisen perusteella ainakin NVDA-ruudunlukuohjelmaa käytettäessä Mozilla Firefox -verkkoselaimella statusviestin vaihtuessa aria-atomic-attribuutin ollessa "false", lukee NVDA uudesta viestistä vain ne **kirjaimet**, jotka ovat muuttuneet edellisen ja nykyisen viestin välillä, joka voi ymmärrettävästi olla käyttäjälle sekavaa.

StatusAnnouncerComponent-komponentin HTML-rakenteen aria-live-elementeissä käytetään myös role-attribuuttia, jonka arvoksi määritellään joko "status" tai "alert". Arvolla "status" ilmaistaan elementin sisällön olevan tiedottavaa ja sitä käyttämällä epäsuorasti asetetaan elementin aria-live-arvoksi "polite" ja aria-atomic-arvoksi "true". Arvo "alert" puolestaan ilmaisee sisällön olevan tiedottavaa, tärkeää ja yleensä aikasensitiivistä. Arvolla "alert" myös epäsuorasti asetetaan aria-live-arvoksi "assertive" ja aria-atomic-arvoksi "true". Käytettävän roolin valitseminen tulisi perustua statusviestin tarkoitukseen ja tärkeyteen. Roolien epäsuorasti asettamia arvoja voidaan myös ylikirjoittaa määrittämällä kyseiset arvot itse. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.)

StatusAnnouncerComponent-komponentin HTML-rakenteessa hyödynnetään a11y-vhiddin CSS-luokkaa, jonka tyylisääntöjen tavoitteena on piilottaa HTML-elementti visuaalisesti, mutta silti pitää se luettavana ruudunlukuohjelmilla (ks. Kuvio 14) (CSS in Action: Invisible Content Just for Screen Reader Users n.d.).

```
.ally-vhidden {  
  position: absolute !important;  
  left: -10000px !important;  
  top: auto !important;  
  width: 1px !important;  
  height: 1px !important;  
  overflow: hidden !important;  
}
```

Kuvio 14. CSS-tyyliohjeet elementin piilottamiseksi visuaalisesti

StatusAnnouncerComponent-komponentin luokkaan (ks. Kuvio 15) joudutaan testaamisen perusteella implementoimaan statusviestien esittämisen viivästyttämistä ja viestien automaattista poistamista, sillä muuten osa testatuista ruudunlukuohjelma- ja verkkoselainkombinaatioista (ks. Taulukko 8) ilmoittavat viestit epätoivotulla tavalla. Poistamalla jo esitetyt statusviestit vältytään myös tilanteilta, joissa esimerkiksi ruudunlukuohjelman käyttäjä päätyisi lukemaan sivun HTML-rakenteeseen jääneitä, sisällöltään vanhentuneita statusviestejä.

```

@Component({
  selector: 'app-status-announcer',
  templateUrl: './status-announcer.component.html',
  styleUrls: ['./status-announcer.component.css']
})
export class StatusAnnouncerComponent implements AfterViewInit, OnDestroy {

  private static readonly MSG_CLEAR_DEBOUNCE: number = 200;
  private static readonly MSG_UPDATE_DELAY: number = 100;

  public readonly messageTypes: string[] = Object.values(LiveMessageImportance);
  public readonly statusMessages:
    Record<LiveMessageImportance, StatusMessage|null> =
    Object.values(LiveMessageImportance)
      .reduce((acc: object, cur: string) => (acc[cur] = null) || acc, {});

  private readonly destroyed$: Subject<boolean> = new Subject();

  constructor(
    private readonly changeDetector: ChangeDetectorRef,
    private readonly statusAnnouncementService: StatusAnnouncementService
  ) { }

  public msgTrackBy(index: number, messageType: string): string {
    return messageType;
  }

  public ngAfterViewInit(): void {
    this.initMessageUpdating();
    this.initMessageClearing();
  }

  public ngOnDestroy(): void {
    this.destroyed$.next(true);
  }

  private initMessageClearing(): void {
    this.messageTypes.forEach((importance: string) => {
      this.statusAnnouncementService.statusAnnouncements.pipe(
        takeUntil(this.destroyed$),
        filter((msg: StatusMessage) => msg.importance === importance),
        debounceTime(StatusAnnouncerComponent.MSG_CLEAR_DEBOUNCE)
      ).subscribe(this.resetMessage.bind(this));
    });
  }

  private initMessageUpdating(): void {
    this.statusAnnouncementService.statusAnnouncements.pipe(
      takeUntil(this.destroyed$),
      delay(StatusAnnouncerComponent.MSG_UPDATE_DELAY),
      tap((msg: StatusMessage) => {
        this.resetMessage(msg);
        this.changeDetector.detectChanges();
      })
    ).subscribe(this.setMessage.bind(this));
  }

  private resetMessage(msg: StatusMessage): void {
    this.setMessage(new LiveMessage('', msg.importance), true);
  }

  private setMessage(msg: StatusMessage, empty?: boolean): void {
    if (typeof this.statusMessages[msg.importance] !== 'undefined') {
      this.statusMessages[msg.importance] = empty ? null : msg;
    }
  }
}

```

Kuvio 15. Esimerkkisovelluksen StatusAnnouncerComponent-komponentin luokka

StatusAnnouncerComponent-komponentin `initMessageUpdating`-metodissa alustetaan reagoiminen StatusAnnouncementService-palvelusta välitettäviin viesteihin lyhyellä 100 millisekunnin viiveellä. Ilman viivettä osalla testaamiseen käytetyistä ruudunlukuohjelma- ja selainkombinaatioista (ks. Taulukko 8) statusviestejä ei lueta esimerkiksi sivun vaihtumisen yhteydessä. Statusviestien ilmoittamatta jättäminen sivun vaihtumisen yhteydessä vaikuttaa johtuvan HTML-dokumentissa tapahtuvien rakennemuutoksien käsittelemisestä sekä implementoidusta fokuksen siirtämisestä (ks. sivu 23). Ennen uuden viestin päivittämistä StatusAnnouncerComponent-komponentin HTML-rakenteeseen on testaamisen perusteella myös tarpeellista tyhjentää statusviestin esittävä HTML-elementti mahdollisesta edellisestä statusviestistä, sillä muuten osa ruudunlukuohjelma- ja selainkombinaatioista lukevat uuden viestin kahdesti.

StatusAnnouncerComponent-komponentin `initMessageClearing`-metodissa alustetaan statusviestien automaattinen poistaminen komponentin HTML-rakenteesta pienen viiveen jälkeen, minkä tarkoituksena on mahdollistaan identtisten statusviestien antaminen peräkkäin. Jos viestejä ei poisteta HTML-rakenteesta, testaamiseen käytettävät ruudunlukuohjelmat eivät lue uutta viestiä, mikäli se on sama kuin edellinen viesti. Testaamiseen perusteella viiveen tulisi olla vähintään 100 millisekuntia, sillä muuten osa ruudunlukuohjelmista ei lue viestiä lainkaan. `initMessageClearing`-metodissa alustettavan toiminnallisuuden seurauksena viestien poistaminen komponentin `initMessageUpdating`-metodissa on yleisimmissä tilanteissa turhaa, mutta esimerkkitsovelluksessa se tehdään varmuuden vuoksi.

Viestien päivittäminen ja poistaminen alustetaan StatusAnnouncerComponent-komponentin `AfterViewInit`-elämänkaarikoukussa (engl. lifecycle hook), jota kutsuttaessa viestejä esittävät ARIA live -alueet (ks. Kuvio 13) ovat jo olemassa HTML-dokumentin rakenteessa, jolloin avustavat teknologiat osaavat reagoida niissä tapahtuviin muutoksiin (`AfterViewInit` n.d.; ARIA live regions n.d.).

## Sivun vaihdosta ilmoittaminen

Esimerkkisovelluksessa halutaan sivun vaihtamisen jälkeen ilmoittaa käyttäjälle uuden sivun otsikko. Toiminnallisuuden tarkoitus sopii hyvin jo luodulle RoutePurposeService-palvelulle, joten implementaatio lisätään siihen (ks. Kuvio 16).

```

constructor(
  private readonly router: Router,
  private readonly statusAnnouncementService: StatusAnnouncementService
) {
  this.enableNavigationAnnouncements();
}

public enableNavigationAnnouncements(): void {
  onPathChange(this.router).pipe(
    switchMap((route: ActivatedRoute) => route.data),
    map((data: AccessibleRouteData) => data.accessibility)
  ).subscribe(this.handleNavigationAnnouncement.bind(this));
}

private handleNavigationAnnouncement(data: RouteAccessibilityData): void {
  this.statusAnnouncementService.announce(
    new PoliteStatusMessage(data.title.sections.title)
  );
}

```

Kuvio 16. Näyte sivun vaihdoksen ilmoittamisesta esimerkkisovelluksen RoutePurposeService-palvelussa

### 6.1.4 Yhteenveto

#### Menettely

Seuraavassa listassa kuvataan esimerkkisovelluksen abstraktimpi menettelytapa käyttäjän siirtyessä toiselle sivulle. Teknisellä tasolla tapahtumien kulku ei välttämättä etene juuri listan kuvaamassa järjestyksessä.

1. Käyttäjä napsauttaa linkkiä, joka laukaisee navigointitapahtumaketjun sovelluksessa. Käytettävän reitittimen RouterOutlet-direktiivin paikalle sijoitetaan uudelle reitille määritetty komponentti, joka edustaa uuden sivun sisältöä.
2. HTML-dokumentin otsikko ja pääsisällön otsikko päivitetään kuvaamaan nykyisen sivun aihetta tai tarkoitusta.
3. Uutta sisältöä suoraan edeltävä pääsisällön otsikkoelementti fokusoidaan.
4. Uuden sivun otsikko kirjoitetaan sivun HTML-rakenteen ARIA live -alueeseen, jonka seurauksena avustavat teknologiat lukevat sivun otsikon käyttäjälle.

## Käsitellyt onnistumiskriteerit

Taulukko 9. Yhteenveto SPA-arkkitehtuurille valittujen onnistumiskriteerien käsitte-lystä

Onnistumiskriteeri	Taso	Miten huomioitu
<b>1.3.1: Info and Relationships</b>	A	Main-maamerkkialuetta käytetään ilmaisemaan verkkosivun pääsisältöä ja se nimetään aria-labelledby-attribuutin avulla.
<b>2.4.2: Page Titled</b>	A	HTML-dokumentin otsikkoa päivitetään kuvaamaan nykyisen sivun aihetta tai tarkoitusta.
<b>2.4.3: Focus Order</b>	A	Sivun vaihtamisen jälkeen fokus siirretään uuden sivun sisältöä suoraan edeltävään otsikkoelementtiin.
<b>2.4.7: Focus Visible</b>	AA	Kun fokus on pääsisällön otsikkoelementissä, se ilmaistaan selkeästi.
<b>4.1.3: Status Messages</b>	AA	Role-, aria-live- ja aria-atomic-attribuutteja käytetään tarkoituksen mukaisesti statusviestien esittämiseen.

## 6.2 Modaalinen dialogi

Esimerkkisovelluksen modaaleihin liittyvät toiminnallisuudet on toteutettu pääasiassa "dialogi" -nimikkeen alle, sillä implementoinnissa on yritetty pitää mielessä myös mahdollinen laajennettavuus ei-modaalisiin dialogeihin.

### 6.2.1 Dialog ja alertdialog -rootit

Esimerkkisovelluksen dialogien hallitsemista varten otetaan käyttöön seuraavaan kuvioon kootut tietotyypit (ks. Kuvio 17).



```

export enum DialogType {
  Alert = 'alrtdialog',
  Dialog = 'dialog'
}

export interface DialogButton {
  text: string;
}

type DialogButtons = { 0: DialogButton } & Array<DialogButton>;

export interface DialogData {
  buttons?: DialogButtons;
  heading: string;
  isModal?: boolean;
  message: string;
}

export class Dialog implements Required<DialogData> {
  public readonly buttons: DialogButtons = [{ text: 'Close' }];
  public readonly dialogType: DialogType = DialogType.Dialog;
  public readonly heading: string;
  public readonly isModal: boolean = true;
  public readonly message: string;

  constructor(data: DialogData) {
    Object.assign(this, data);
  }
}

export class AlertDialog extends Dialog {
  public readonly dialogType: DialogType = DialogType.Alert;
  constructor(data: Omit<DialogData, 'isModal'>) {
    super({ ...data, isModal: true });
  }
}

```

Kuvio 17. Esimerkkisovelluksen dialogien käsittelyssä hyödynnettävät tietotyypit

ARIA 1.1 määrittelee dialogeille kaksi mahdollista roolia, ”dialog” ja ”alrtdialog”. Roolin tulisi viestiä dialogin sisällön tarkoitusta ja kriittisyyttä käyttäjälle, mikä tulisi ottaa huomioon käytettävää roolia valittaessa. Esimerkkisovelluksen tietotyypeissä ARIA 1.1:n dialogityyppejä edustaa luokat Dialog ja AlertDialog. Kyseisissä luokissa molemmat dialogityypit määritetään oletuksena modaalisiksi luokkien isModal-ominaisuudella, mutta vain ”alrtdialog” -roolin kanssa suositellaan aina käytettävän modaalista dialogia. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.) A-tason onnistumiskriteerin 2.1.2: *No Keyboard Trap* ohjeistuksen mukaisesti, modaalisia dialogeja käytettäessä tulisi myös selvästi viestiä käyttäjälle, kuinka modaaleille ominainen fokuksen siirtämisen rajoittaminen voidaan vapauttaa (Understanding Success Criterion 2.1.2: No Keyboard Trap n.d.). Esimerkkisovelluksessa halutaan siis aina viestiä käyttäjälle, miten modaalinen dialogi voidaan sulkea ja täten vapauttaa fokuksen siirtäminen. Tätä pyritään valvomaan mahdollistamalla vuorovaikutus kaikkien dialogien kanssa, joten dialogien nappien minimimääräksi määritetään yksi. Esimerk-

kisovelluksen dialogien kaikki napit ovat kuitenkin yksinkertaisuuden vuoksi toiminnallisuudeltaan identtisiä. Myös dialogidatan heading ja message -ominaisuudet on myös määritelty pakollisiksi, sillä molempia tullaan hyödyntämään dialogin tarkoituksen tunnistamiseen dialogin HTML-rakenteessa.

### **Dialogien käsitteleminen**

Jotta dialogeja voitaisiin helposti avata ja sulkea sovelluksen eri osista, luodaan esimerkkisovellukseen DialogService-palvelu (ks. Kuvio 18) dialogien käsittelemistä varten, mikä toteutetaan muiden dialogitoiminnallisuuksien ohella uuteen DialogModule-moduuliin.

```

@Injectable({ providedIn: 'root' })
export class DialogService {

  public get dialog(): Observable<Dialog|null> {
    return this.dialog$.asObservable();
  }
  public get dialogOpen(): Observable<boolean> {
    return this.dialogOpen$.asObservable();
  }
  public get modalOpen(): Observable<boolean> {
    return this.modalOpen$.asObservable();
  }

  private readonly dialog$: BehaviorSubject<Dialog|null> =
    new BehaviorSubject(null);
  private readonly dialogOpen$: BehaviorSubject<boolean> =
    new BehaviorSubject(false);
  private readonly modalOpen$: BehaviorSubject<boolean> =
    new BehaviorSubject(false);

  constructor(private readonly routePurposeService: RoutePurposeService) { }

  public closeDialog(): void {
    const { isModal } = this.dialog$.getValue() || { isModal: false };
    this.dialog$.next(null);
    this.dialogOpen$.next(false);
    if (isModal) {
      this.modalOpen$.next(false);
    }
    this.routePurposeService.setDocumentTitle(DocumentTitleLevel.Context, {});
  }

  public openDialog(dialog: Dialog|AlertDialog): void {
    if (this.dialog$.getValue()) {
      this.closeDialog();
    }
    this.dialog$.next(dialog);
    this.dialogOpen$.next(true);
    if (dialog.isModal) {
      this.modalOpen$.next(true);
      this.routePurposeService.setDocumentTitle(DocumentTitleLevel.Context, {
        sections: { modifier: dialog.heading },
        delimiters: { modifier: ' - ' }
      });
    }
  }
}

```

### Kuvio 18. Näyte esimerkkisovelluksen DialogService-palvelusta

Esimerkkisovelluksessa dialogin avaaminen tapahtuu välittämällä DialogService-palvelun `openDialog`-metodille Dialog- tai AlertDialog-luokan instanssi. Modaalin avaamisen yhteydessä hyödynnetään aiemmin luotua RoutePurposeService-palvelua (ks. sivu 33) sovelluksen HTML-dokumentin otsikon päivittämiseksi kuvaamaan avatun modaalin sisältöä. Kyseisen dynaamisuuden toteuttaminen perustuu A-tason onnistumiskriteerin 2.4.2: *Page Titled* ohjeistukseen, mutta otsikon päivittämisen tarpeellisuus on tässä tilanteessa hyvin tulkinnanvaraista ja tilanneriippuvaista (Understanding Success Criterion 2.4.2: Page Titled n.d.). Esimerkkisovelluksessa otsikon päivittäminen on kuitenkin päätetty toteuttaa modaalien kanssa, millä myös demonstroidaan toteutettua otsikonpäivittämissimplementaatiota.

## Dialogien esittäminen

Dialogien esittämistä varten luodaan DialogComponent-komponentti, jonka HTML-rakenteessa (ks. Kuvio 19) hyödynnetään luotujen dialogitietotyyppien (ks. Kuvio 17) määrittämää dataa.

```

<div
  *ngIf="dialog"
  [ngClass]="{ 'dialog--modal': (modalOpen$ | async) }"
  class="dialog">
  <div
    #dialogWindowEl
    [appRestrictFocus]="dialog.isModal"
    [appRestrictFocusInitial]="false"
    [attr.role]="dialog.dialogType"
    [attr.aria-modal]="dialog.isModal || null"
    [ngClass]="{ 'dialog__window--modal': (modalOpen$ | async) }"
    id="dialog-window"
    class="dialog__window"
    aria-labelledby="dialog-heading"
    aria-describedby="dialog-message">
    <div class="dialog__content">
      <h1 #dialogHeadingEl id="dialog-heading" class="dialog__heading" tabindex="-1">
        {{ dialog.heading }}
      </h1>
      <p id="dialog-message" class="dialog__message">
        {{ dialog.message }}
      </p>
    </div>
    <div class="dialog__buttons">
      <button
        *ngFor="let dialogButton of dialog.buttons"
        (click)="onClose()"
        type="button"
        aria-controls="dialog-window"
        class="dialog__button">
        {{ dialogButton.text }}
      </button>
    </div>
  </div>
</div>

```

Kuvio 19. Esimerkkisovelluksen DialogComponent-komponentin HTML-rakenne

Saavutettavuuden kannalta komponentin HTML-rakenteessa olennaista on muun muassa attribuuttien role, aria-labelledby ja aria-describedby käyttäminen. Role-attribuutille annetaan arvoksi *WAI-ARIA 1.1*:n määrittämä dialogirooli, "dialog" tai "alertdialog", jolla ohjeistetaan avustavia teknologioita ilmoittamaan käyttäjälle minkä tyyppinen dialogi on kyseessä. Dialogi-roolisille elementeille suositellaan määritettävän aria-labelledby-attribuutti, jolla määritetään dialogia nimeävien elementtien id-arvot. Dialogin nimeämisen tarkoituksena on helpottaa etenkin avustavien teknologioiden käyttäjiä tunnistamaan kyseinen elementti ja sen tarkoitus. Esimerkkisovelluksessa dialogin nimeävänä elementtinä toimii dialogin HTML-rakenteen si-

sältämä h1-otsikkoelementti. Aria-labelledby-attribuutin sijaan voidaan käyttää samaa tarkoitusta ajavaa aria-label-attribuuttia, jota suositellaan käytettävän, jos nimeävää elementtiä ei esitetä visuaalisesti. Aria-label-attribuutille nimeävä teksti annetaan suoraan merkkijonona. Lisäksi suositellaan, että vähintäänkin alertdialog-rootin dialogilla olisi sen tarkoitusta kuvaavien elementtien id-arvot määrittävä aria-describedby-attribuutti. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.)

## 6.2.2 Fokuksen hallinta

### **Dialogin avautuessa**

*WAI-ARIA 1.1* suosittelee, että dialogeilla olisi vähintään yksi fokuoitavissa oleva elementti, johon fokus siirtyisi automaattisesti dialogin avautuessa (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Lisäksi *WAI-ARIA Authoring Practices 1.1* -dokumentissa suositellaan, että dialogin avautuessa tulisi välttää näkymän vierittämistä dialogin pohjalle (WAI-ARIA Authoring Practices 1.1 2019). Esimerkkisovelluksen dialogien napit on sijoitettu dialogi-ikkunan pohjalle, jolloin fokuksen automaattinen siirtäminen kyseisiin nappeihin saattaisi aiheuttaa näkymän vierityksen dialogin pohjalle, mikäli dialogin sisältö ei kerrallaan mahdu käyttäjän näytölle. Täten esimerkiksi sovelluksessa on päädytty dialogin avautuessa automaattisesti fokuoimaan dialogin pääotsikkoelementti, mikä tukee myös loogista fokuksen siirtämisjärjestystä, sillä pääotsikko on dialogin ensimmäinen elementti, jolloin siitä on sujuvaa jatkaa näppäimistö pohjaista selaamista dialogin muuhun sisältöön. Vaihtoehtoisesti esimerkiksi dialogin napit voitaisiin sijoittaa dialogin sisällön yläpuolelle, jolloin myös ne toimisivat mahdollisena fokuointikohteena. Otsikon fokuoiminen tapahtuu DialogComponent-komponentin luokan (ks. Kuvio 20) handleDialog-metodissa.

```

@Component({
  selector: 'app-dialog',
  templateUrl: './dialog.component.html',
  styleUrls: ['./dialog.component.css']
})
export class DialogComponent implements OnInit, OnDestroy {

  public dialog: Dialog|null;
  public readonly modalOpen$: Observable<boolean> = this.dialogService.modalOpen;

  private readonly destroyed$: Subject<boolean> = new Subject();
  @ViewChild('dialogHeadingEl', { static: false })
  private readonly dialogHeadingEl: ElementRef;

  constructor(
    private readonly changeDetector: ChangeDetectorRef,
    private readonly dialogService: DialogService
  ) { }

  public ngOnDestroy(): void {
    this.destroyed$.next(true);
    this.handleDialogClose();
  }

  public ngOnInit(): void {
    this.dialogService.dialog
      .pipe(takeUntil(this.destroyed$))
      .subscribe(this.handleDialog.bind(this));
  }

  public onClose(): void {
    this.dialogService.closeDialog();
  }

  private handleDialog(dialog: Dialog|null): void {
    this.dialog = dialog;
    if (dialog) {
      this.changeDetector.detectChanges();
      this.dialogHeadingEl.nativeElement.focus();
    }
  }
}

```

Kuvio 20. Näyte esimerkisovelluksen DialogComponent-komponentin luokasta

### Modaalin ollessa auki

Dialogin lapsielementin automaattisen fokusoimisen lisäksi modaalien kanssa suositellaan, että käyttäjän fokuksen siirtäminen rajoitettaisiin modaalin sisälle. DialogComponent-komponentin HTML-rakenteessa (ks. Kuvio 19) hyödynnettävällä aria-modal-attribuutilla voidaan ohjeistaa avustavia teknologioita, että attribuutin omaava elementti on modaalinen, jolloin interaktio kyseisen modaalin ulkopuolisen sisällön kanssa on tarkoitus estää. Käytännössä tämä tarkoittaa, että attribuutilla suositellaan esimerkiksi ruudunluohjelmia automaattisesti navigoimaan modaalielementtiin sekä mahdollisesti rajoittamaan käyttäjän elementtinavigoiminen modaalin sisälle. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.) Aria-modal-attribuutti ei kuitenkaan estä normaalia osoitin- tai näppäimistöinteraktiota mo-

daalin ulkopuolisen sisällön kanssa, joten fokuksen rajoittamista varten tarvitaan lisätoimenpiteitä. Esimerkkisovellukseen luodaan RestrictFocusDirective-direktiivi (ks. Kuvio 21), jolla voidaan helposti rajoittaa käyttäjän fokuksen siirtäminen ja interaktio direktiiviä hyödyntävän elementin sisälle.

```

@Directive({ selector: '[appRestrictFocus]' })
export class RestrictFocusDirective implements OnChanges, AfterContentInit, OnDestroy {
  private static readonly FOCUSABLE_QUERY: string = [
    'a[href]', 'button:not([disabled])', 'input:not([disabled])', 'select:not([disabled])',
    '[tabindex]:not([tabindex="-1"]):not([disabled])', 'textarea:not([disabled])'
  ].join(' ');
  private static readonly RESTRICT_EVENTS: string[] = [ 'focus', 'mousedown', 'click' ];
  @Input('appRestrictFocus') private readonly enabled: boolean = true;
  private readonly eventListenerFn: () => void = this.restrictFocus.bind(this);
  private focusDirection: 'prev'|'next' = 'next';
  @Input('appRestrictFocusInitial') private readonly initial: boolean = true;

  constructor(@Inject(DOCUMENT) private readonly document: Document,
    private readonly el: ElementRef) { }

  @HostListener('document:keydown', ['$event.key', '$event.shiftKey'])
  public detectFocusDirection(key: string, shiftKey: boolean): void {
    if (key === 'Tab') {
      if (shiftKey) this.focusDirection = 'prev';
      else this.focusDirection = 'next';
    }
  }

  public ngAfterContentInit(): void {
    this.addListeners();
    if (this.initial) this.focusWithin();
  }

  public ngOnChanges(changes: SimpleChanges): void {
    if (!changes.firstChange && changes.enabled && changes.enabled.currentValue) {
      this.focusWithin();
    }
  }

  public ngOnDestroy(): void {
    this.removeListeners();
  }

  private addListeners(): void {
    RestrictFocusDirective.RESTRICT_EVENTS.forEach((eventName: string) => {
      this.document.addEventListener(eventName, this.eventListenerFn, true);
    });
  }

  private focusWithin(): void {
    const focusableElements =
      this.el.nativeElement.querySelectorAll(RestrictFocusDirective.FOCUSABLE_QUERY);
    if (focusableElements.length) {
      if (this.focusDirection === 'prev') {
        focusableElements[focusableElements.length - 1].focus();
      } else {
        focusableElements[0].focus();
      }
    }
  }

  private removeListeners(): void {
    RestrictFocusDirective.RESTRICT_EVENTS.forEach((eventName: string) => {
      this.document.removeEventListener(eventName, this.eventListenerFn, true);
    });
  }

  private restrictFocus(event: FocusEvent|MouseEvent): void {
    if (this.enabled && !this.el.nativeElement.contains(event.target)) {
      event.stopImmediatePropagation();
      event.preventDefault();
      this.focusWithin();
    }
  }
}

```

Kuvio 21. Näyte esimerkkisovelluksen RestrictFocusDirective-direktiivistä



Esimerkkisovelluksen `RestrictFocusDirective`-direktiivissä huomioitavaa on `detectFocusDirection`-metodi, jolla pyritään havaitsemaan käyttäjän haluama suunta fokuksen siirtämiselle. Toiminnon tarkoituksena on, että direktiivin aiheuttama fokuksen siirtäminen toimisi ennustettavasti ja *WAI-ARIA Authoring Practices 1.1* -dokumentin suosituksen mukaisesti (WAI-ARIA Authoring Practices 1.1 2019).

### **Dialogin sulkeutuessa**

*WAI-ARIA Authoring Practices 1.1* -dokumentti suosittelee dialogin sulkeuduttua automaattisesti siirtämään fokuksen takaisin siihen elementtiin, josta käyttäjä avasi dialogin, mikäli kyseinen elementti on yhä olemassa eikä fokuksen siirtämiselle ole loogisempaa kohdetta (WAI-ARIA Authoring Practices 1.1 2019). Esimerkkisovelluksessa kuvattu toiminnallisuus implementoidaan jo luotuun `DialogService`-palveluun (ks. Kuvio 18), johon lisättäviä `storeFocus`- ja `restoreFocus`-metodeja (ks. Kuvio 22) kutsutaan tilanteen mukaisesti, tallentaen ja palauttaen dialogin avaamista edeltävän fokuksen, mikäli se on mahdollista. Testaamisen perusteella esimerkki-implementaatioissa on myös tarpeellista laukaista sovelluksen muutoksien havaitsemissykli (engl. *change detection cycle*) kutsumalla `ApplicationRef`-viitteen `tick`-metodia ennen fokuksen palauttamista, sillä muuten fokus menetetään dialogin sulkemisen yhteydessä kaikilla testatuilla verkkoselaimilla (ks. Taulukko 8).

```

private prevFocus: HTMLElement|null = null;

constructor(
  @Inject(DOCUMENT) private readonly document: Document,
  private readonly applicationRef: ApplicationRef,
  private readonly routePurposeService: RoutePurposeService
) { }

public closeDialog(): void {
  const { isModal } = this.dialog$.getValue() || { isModal: false };
  this.dialog$.next(null);
  this.dialogOpen$.next(false);
  if (isModal) {
    this.modalOpen$.next(false);
  }
  this.restoreFocus();
  this.routePurposeService.setDocumentTitle(DocumentTitleLevel.Context, {});
}

public openDialog(dialog: Dialog|AlertDialog): void {
  if (this.dialog$.getValue()) {
    this.closeDialog();
  }
  this.storeFocus();
  this.dialog$.next(dialog);
  this.dialogOpen$.next(true);
  if (dialog.isModal) {
    this.modalOpen$.next(true);
    this.routePurposeService.setDocumentTitle(DocumentTitleLevel.Context, {
      sections: { modifier: dialog.heading },
      delimiters: { modifier: ' - ' }
    });
  }
}

private restoreFocus(): void {
  if (this.prevFocus) {
    this.applicationRef.tick();
    if (this.document.body.contains(this.prevFocus)) {
      this.prevFocus.focus();
    }
    this.prevFocus = null;
  }
}

private storeFocus(): void {
  const el = this.document.activeElement;
  this.prevFocus = el instanceof HTMLElement ? el : null;
}

```

Kuvio 22. Fokuksen palauttamiseen liittyvät lisäykset DialogService-palveluun

### 6.2.3 Näppäimistökomennot

WAI-ARIA *Authoring Practices 1.1* -dokumentissa suositellaan, että käyttäjät voisivat sulkea avatun dialogin Esc-näppäintä käyttäen (WAI-ARIA *Authoring Practices 1.1* 2019). Näppäimistökomentojen helppoa määrittämistä varten esimerkisovellukseen luodaan seuraavaan kuvioon (ks. Kuvio 23) kootut tietotyypit, joita tullaan myöhemmin hyödyntämään myös ääretöntä vieritystä implementoidessa (ks. sivu 64).

```

export interface OnEventMatchOptions {
  preventDefault?: boolean;
  stopImmediatePropagation?: boolean;
  stopPropagation?: boolean;
}

export interface KeyboardControlOptions {
  caseInsensitive?: boolean;
  onMatchOptions?: OnEventMatchOptions|null;
  shiftKey?: boolean;
}

export class KeyboardControl {

  private options: Required<KeyboardControlOptions> = {
    caseInsensitive: true,
    onMatchOptions: null,
    shiftKey: false
  };

  constructor(public readonly key: string, options?: KeyboardControlOptions) {
    if (options) {
      this.setOptions(options);
    }
  }

  public matches(
    event: KeyboardEvent,
    onMatch: OnEventMatchOptions|null = this.options.onMatchOptions
  ): boolean {
    const { caseInsensitive, shiftKey } = this.options;
    const key = caseInsensitive ? this.key.toLowerCase() : this.key;
    const eventKey = caseInsensitive ? event.key.toLowerCase() : event.key;
    const matches = eventKey === key && event.shiftKey === shiftKey;
    if (matches && onMatch) {
      if (onMatch.preventDefault) {
        event.preventDefault();
      }
      if (onMatch.stopImmediatePropagation) {
        event.stopImmediatePropagation();
      }
      else if (onMatch.stopPropagation) {
        event.stopPropagation();
      }
    }
    return matches;
  }

  public setOptions(options: KeyboardControlOptions): KeyboardControl {
    this.options = { ...this.options, ...options };
    return this;
  }
}

```

Kuvio 23. Esimerkkisovelluksen näppäimistökomentojen määrittämisessä hyödynnettävät tietotyypit

Luotuja tietotyyppettä hyödynnetään DialogComponent-komponentin luokkaan (ks. Kuvio 20) tehtävillä lisäyksillä (ks. Kuvio 24), joilla mahdollistetaan dialogien sulkeminen Esc-näppäimellä.

```
private static readonly KEY_CLOSE: KeyboardControl = new KeyboardControl('Escape', {
  onMatchOptions: {
    preventDefault: true,
    stopImmediatePropagation: true
  }
});

@HostListener('keydown', ['$event'])
public onClose(event?: KeyboardEvent): void {
  if (!event || (event && DialogComponent.KEY_CLOSE.matches(event))) {
    this.dialogService.closeDialog();
  }
}
```

Kuvio 24. Näppäimistökomentolisäykset esimerkksiovelluksen DialogComponent-komponenttiin

#### 6.2.4 Yhteenveto

##### **Menettely**

Seuraavassa listassa kuvataan esimerkksiovelluksen abstraktimpi menettelytapa käyttäjän avatessa ja sulkiessa modaalisen dialogin.

1. Käyttäjä painaa dialogin avaavaa elementtiä ja viittaus kyseiseen elementtiin tallennetaan muistiin.
2. Modaalinen avautuu ja fokus siirretään automaattisesti sen otsikkoelementtiin. Modaalinen dialogi on merkitty tarkoituksen mukaisella roolilla, jolloin avustavat teknologiat osaavat ilmoittaa dialogista käyttäjälle.
3. Käyttäjän fokuksen siirtäminen ja interaktio rajoitetaan modaalin sisälle ja käyttäjälle viestitään kuinka modaalinen voi sulkea.
4. Käyttäjä sulkee modaalin ja fokus siirretään automaattisesti takaisin dialogin avanneeseen elementtiin.

## Käsitellyt onnistumiskriteerit

Taulukko 10. Yhteenveto modaalisille dialogeille valittujen onnistumiskriteerien käsitelystä

Onnistumiskriteeri	Taso	Miten huomioitu
<b>1.3.1: Info and Relationships</b>	A	Modaalin otsikko ja kuvaava teksti yhdistetään modaalielementtiin aria-labelledby ja aria-describedby -attribuuteilla.
<b>2.1.2: No Keyboard Trap</b>	A	Kun käyttäjän fokuksen siirtäminen rajoitetaan modaalin sisälle, hänelle esitetään selkeä tapa modaalin sulkemiselle.
<b>2.4.3: Focus Order</b>	A	Modaalin avautuessa fokus siirretään sen otsikkoelementtiin ja modaalin sulkeutuessa fokus siirretään takaisin modaalin avanneeseen elementtiin. Modaalin ollessa auki käyttäjän fokuksen siirtäminen rajoitetaan modaalin sisälle, missä fokuksen siirtäminen tapahtuu loogisessa järjestyksessä.
<b>4.1.2: Name, Role, Value</b>	A	Modaalille asetetaan role-attribuutiksi sen tarkoitusta kuvaava dialogirooli; dialog tai alert-dialog. Aria-modal-attribuuttia käytetään viestimään dialogin olevan modaalinen.

## 6.3 Ääretön vieritys

Esimerkkisovelluksen ääretön vieritys -implementaatio toteutetaan sosiaalisia medioita imitoivan aikajanasyötteen muodossa, mutta käsiteltävät saavutettavuusseikat pätevät myös muihin äärettömän vierityksen käyttötilanteisiin.

### 6.3.1 Feed-rooli

WAI-ARIA:n uusimmassa, 1.1 versiossa määritelty feed-rooli sopii äärettömän vierityksen kaltaiselle toiminnallisuudelle erinomaisesti. Feed-roolin elementti tulisi kääriä sisälleen listan article-roolin lapsielementtejä, joita voidaan näkymän vierityksen seurauksena lisätä tai poistaa listan alusta tai lopusta. Kun feed-roolia ja sen oheisia HTML-attribuutteja käytetään oikein, voidaan auttaa avustavia teknologioita sujuvammin reagoimaan feed-elementin sisältömuutoksiin. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.)

Esimerkkisovelluksessa feed-roolia hyödynnetään uuteen FeedModule-moduuliin luotavassa FeedComponent-komponentissa, johon saavutettava ääretön vieritysmekanismi tullaan toteuttamaan. FeedComponent-komponentille mahdollistetaan välitettävän dataa, jolla määritetään esitettävää sisältöä ja feed-elementin statusta. Datat käsittelemistä varten luodaan seuraavaan kuvioon kootut tietotyypit (ks. Kuvio 25).

```

export const randomId = (count: number = 1, radix: number = 36): string =>
  [...Array(count)]
    .map(() => Math.random())
    .concat(performance.now())
    .map((e: number) => e.toString(radix).slice(2))
    .join('')
    .replace('.', '');

export interface FeedStatus {
  busy: boolean;
  checked: boolean;
  error: boolean;
  loading: boolean;
}

export interface FeedItemData {
  content: string;
  heading: string;
  id?: string;
}

export class FeedItem implements Required<FeedItemData> {

  public readonly content: string;
  public readonly describedById: string;
  public readonly heading: string;
  public readonly id: string;
  public readonly labelledById: string;

  constructor(data: FeedItemData) {
    Object.assign(this, data);
    if (!data.id) {
      this.id = randomId();
    }
    this.labelledById = `fi-l-${this.id}`;
    this.describedById = `fi-d-${this.id}`;
  }
}

```

Kuvio 25. Esimerkkisovelluksen ääretön vieritys -implementaatiossa hyödynnettävät tietotyypit

FeedComponent-komponentin (ks. Kuvio 26) feedStatus-syötearvon busy-ominaisuudella hallitaan komponentin HTML-rakenteessa (ks. Kuvio 27) hyödynnettävän aria-busy-attribuutin arvoa. Feed-roolin kanssa suositellaan tarvittaessa hyödyntämään aria-busy-attribuuttia, jolla voidaan viestiä avustaville teknologioille, että attribuutin omaavan elementin HTML-rakennetta ollaan muokkaamassa. Aria-busy-attribuutille määritellään totuusarvo, "true" tai "false", joista "true" viestii, että kyseisen elementin rakennetta ollaan muokkaamassa, eikä avustavien teknologioiden tarvitse viestiä tapahtuvista muutoksista käyttäjälle, ennen kuin attribuutin arvoksi palautetaan oletusarvo "false". Aria-busy-attribuuttia suositellaan hyödynnettävän tilanteissa, joissa HTML-rakenteen muokkaaminen vaatii useita peräkkäisiä DOM-operaatioita, eikä kaikkia muutoksia toteuteta välittömästi kerralla. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017; WAI-ARIA Authoring Practices 1.1 2019.) Aria-busy-attribi-

buutille tulisi myös aina asettaa arvoksi ”true”, mikäli feed-elementti on keskeneräisen latauksen takia toistaiseksi tyhjä article-roolin lapsielementeistä, sillä article-roolin elementti on feed-roolin elementin pakollinen lapsielementti (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Useille peräkkäisille DOM-operaatioille ei ole esimerkkisovelluksessa tarvetta, joten aria-busy-attribuutin arvoa päivitetään vain syötteen sisällön ensimmäisen latauksen yhteydessä.

```

@Component({
  selector: 'app-feed',
  templateUrl: './feed.component.html',
  styleUrls: ['./feed.component.css']
})
export class FeedComponent implements OnChanges {

  public get disableInfiniteScroll(): boolean {
    const { loading, checked, error } = this.feedStatus;
    return loading || checked || error;
  }

  @Input() public readonly feedStatus: FeedStatus;
  @Input() public readonly infiniteScrollContainer: Window = this.window;
  @Input() public readonly items: FeedItem[] = [];
  @Input() public readonly label: string;
  @Input() public readonly scrollDistance: number = 5;
  @Input() public readonly setSize: number = -1;

  @Output() private readonly scrolled: EventEmitter<void> = new EventEmitter();

  constructor(@Inject(WINDOW) private readonly window: Window) { }

  public feedItemTrackBy(index: number, item: FeedItem): string {
    return item.labelledById;
  }

  public onScroll(): void {
    this.scrolled.emit();
  }
}

```

Kuvio 26. Näyte esimerkkisovelluksen FeedComponent-komponentin luokasta

WAI-ARIA 1.1 suosittelee, että feed-elementtiin pyritään lataamaan lisää sisältöä tarpeeksi ajoissa, jotta käyttäjä voisi sujuvasti siirtyä feed-elementin sisällön välillä (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Uuden sisällön latauksen laukaiseminen tarpeeksi ajoissa tulisi siis huomioida ääretöntä vieritystä toteutettaessa. Esimerkkisovelluksessa käyttäjän vierittämisen havaitseminen jätetään FeedComponent-komponentin HTML-rakenteessa (ks. Kuvio 27) hyödynnettävän kolmannen osapuolen infiniteScroll-direktiivin vastuulle. Esimerkissä infiniteScroll-direktiivin infiniteScrollDistance-syötearvoksi on määritetty 5, jolloin direktiivi lähettää scrolled-tapahtumaa, kun direktiivin käyttämästä näkymästä (esimerkissä window-objekti) on



vieritetty 50 % (Angular Infinite Scroll n.d.). Direktiivin lähettämään scrolled-tapahtumaan reagoidaan FeedComponent-komponentin luokassa, jonka seurauksena komponentti lähettää vastaavan tapahtuman. FeedComponent-komponentin lähettämään scrolled-tapahtumaan voidaan reagoida kyseistä komponenttia hyödyntävässä isäntäkomponentissa, esimerkiksi lataamalla lisää sisältöä. Esimerkkisovelluksessa hyödynnettävä infiniteScroll-direktiivi on saatavilla muun muassa npm-paketinhallintajärjestelmästä nimellä ngx-infinite-scroll (käytetty versio 8.0.0).

```
<div class="feed">
  <div
    [attr.aria-busy]="!items.length || feedStatus.busy"
    [attr.aria-label]="label || null"
    [infiniteScrollContainer]="infiniteScrollContainer"
    [infiniteScrollDisabled]="disableInfiniteScroll"
    [infiniteScrollDistance]="scrollDistance"
    infiniteScroll
    (scrolled)="onScroll()"
    role="feed">
    <app-feed-item
      *ngFor="let item of items; index as i; trackBy: feedItemTrackBy"
      [setSize]="setSize"
      [posInSet]="i+1"
      [item]="item">
    </app-feed-item>
  </div>
</div>
```

Kuvio 27. Näyte esimerkkisovelluksen FeedComponent-komponentin HTML-rakenteesta

### 6.3.2 Article-rooli

Feed-roolin elementti tulisi koostua article-roolin lapsielementeistä, joita lisätään tai poistetaan käyttäjän toiminnan seurauksena (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Esimerkkisovelluksen FeedComponent-komponentissa listattavat article-elementit toteutetaan erilliseen FeedItemComponent-komponenttiin (ks. Kuvio 28), jonka isäntäelementin role-attribuutiksi määritetään Angularin HostBinding-dekoraattorilla "article".

```

@Component({
  selector: 'app-feed-item',
  template: `
    <h3 [attr.id]="item.labelledById">{{ item.heading }}</h3>
    <p [attr.id]="item.describedById">{{ item.content }}</p>
    <button type="button" (click)="openItem()">
      Open <span class="a11y-vhidden">{{ item.heading }}</span>
    </button>
  `,
  styleUrls: ['./feed-item.component.css']
})
export class FeedItemComponent implements OnInit {

  @HostBinding('attr.aria-describedby') public hostDescribedBy: string;
  @HostBinding('attr.aria-labelledby') public hostLabelledBy: string;
  @HostBinding('attr.role') public readonly hostRole: string = 'article';
  @HostBinding('attr.tabindex') public readonly hostTabIndex: number = -1;
  @Input() public readonly item: FeedItem;
  @HostBinding('attr.aria-posinset')
  @Input() public readonly posInSet: number|null = null;
  @HostBinding('attr.aria-setsize')
  @Input() public readonly setSize: number|null = null;

  constructor(
    private readonly dialogService: DialogService,
    public readonly elementRef: ElementRef
  ) { }

  public ngOnInit(): void {
    this.hostLabelledBy = this.item.labelledById;
    this.hostDescribedBy = this.item.describedById;
  }

  public openItem(): void {
    this.dialogService.openDialog(new Dialog({
      heading: this.item.heading,
      message: this.item.content
    }));
  }
}

```

Kuvio 28. Näyte esimerkkisovelluksen FeedItemComponent-komponentin luokasta

Jokainen feed-elementin article-lapsielementti suositellaan nimettävän aria-labelledby- tai aria-label-attribuutilla ja kullekin article-elementille tulisi mielellään määrittellä elementin pääsisältöä kuvaavien elementtien id-arvot aria-describedby-attribuutilla (WAI-ARIA Authoring Practices 1.1 2019). Koska HTML5:ssä käytettävien id-arvojen tulee olla uniikkeja, pyritään FeedItemComponent-komponentissa hyödynnettäville FeedItem-objekteille (ks. Kuvio 25) generoimaan uniikit merkkijonot niiden labelledById- ja describedById-ominaisuuksiin, joita voidaan käyttää article-elementtien tarkoitusta kuvaavien elementtien yksilöimiseen (HTML Standard n.d.). Uniikkeja id-arvoja ei tietenkään tarvitse generoida asiakaspuolella vaan ne voidaan muodostaa myös palvelinpuolen datasta, mikäli vastaanotettava data sen mahdollistaa.

Feed-elementin article-elementille voidaan myös määrittää aria-setsize-attribuutti, jolla viestitään ladattavan sisällön kokonaismäärää avustaville teknologioille. Attribuutille määritetään siis arvoksi feed-elementin article-lapsielementtien lopullinen lukumäärä, mikäli se on tiedossa. Jos article-elementtien lopullista lukumäärää ei voida selvittää tai määrä on äärimmäisen suuri tai usein vaihtuva, voidaan arvoksi määrittää ”-1”, millä viestitään, että kokonaislukumäärää ei tiedetä. Aria-setsize-attribuutin ohella voidaan käyttää myös aria-posinset-attribuuttia, jolla viestitään kunkin elementin sijoitusta nykyisessä luettelossa. (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017.)

*WAI-ARIA 1.1* suosittelee avustavia teknologioita fokusoimaan feed-elementin article-elementin, jonka kohdalla avustavan teknologian lukemiskohdistin (engl. reading cursor) on (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Kyseisen toiminnallisuuden tukemiseksi esimerkisovelluksen FeedItemComponent-komponentin isäntäelementin tabindex-attribuutin arvoksi on määritetty ”-1”.

### 6.3.3 Näppäimistökomennot

*WAI-ARIA 1.1* suosittelee tarjoamaan näppäimistökomentoja fokuksen siirtämiseksi feed-elementin article-elementtien välillä, millä pyritään helpottamaan näppäimistö-käyttäjien navigoimista (Accessible Rich Internet Applications (WAI-ARIA) 1.1 2017). Käyttäjille suositellaan tarjottavan näppäimistökomennot fokuksen siirtämiseksi seuraavaan, edelliseen, ensimmäiseen sekä viimeiseen article-elementtiin (WAI-ARIA Authoring Practices 1.1 2019). Kuvaillun toiminnallisuuden toteuttamisessa hyödynnetään esimerkisovelluksen modaalien näppäimistökomentojen toteuttamisessa käytettyjä tietotyyppäjä (ks. Kuvio 23), joiden toiminnallisuutta vielä laajennetaan seuraavaan kuvioon kootuilla tietotyypeillä (ks. Kuvio 29).

```

export class KeyboardControlGroup {
  public readonly controls: KeyboardControl[];

  constructor(...controls: KeyboardControl[]) {
    this.controls = controls;
  }

  public matches(event: KeyboardEvent, options?: OnEventMatchOptions): boolean {
    return this.controls.some(
      (control: KeyboardControl) => control.matches(event, options)
    );
  }

  public setOptions(options: KeyboardControlOptions): KeyboardControlGroup {
    this.controls.forEach(
      (control: KeyboardControl) => control.setOptions(options)
    );
    return this;
  }
}

export type KeyboardControlRecord<K extends string = string> =
  Record<K, KeyboardControl|KeyboardControlGroup>;

export class KeyboardControlCollection<K extends string = string> {
  public readonly controls: KeyboardControlRecord<K>;

  constructor(controls: KeyboardControlRecord<K>) {
    this.controls = controls;
  }

  public match(event: KeyboardEvent): K|undefined {
    for (const key in this.controls) {
      if (this.controls[key].matches(event)) {
        return key;
      }
    }
  }

  public setOptions(options: KeyboardControlOptions): KeyboardControlCollection<K> {
    Object.keys(this.controls).forEach(
      (key: string) => this.controls[key].setOptions(options)
    );
    return this;
  }
}

export type FeedControls = 'next'|'previous'|'last'|'first';

```

Kuvio 29. Tietotyyppilisäykset esimerkisovelluksen näppäimistökomentojen toteuttamiseksi

Lisättyjä tietotyyppettä hyödynnetään FeedComponent-komponentin luokkaan (ks. Kuvio 26) tehtävissä lisäyksissä (ks. Kuvio 30), joilla mahdollistetaan *WAI-ARIA 1.1* -dokumentin mukaiset näppäimistökomento-ominaisuudet feed-roolin elementille. Myös komponentin HTML-rakenteeseen tehdään luokan lisäyksien tukemista varten tarvittavat muutokset (ks. Kuvio 31).

```

private static readonly KB_CONTROLS: KeyboardControlCollection<FeedControls> =
  new KeyboardControlCollection({
    first: new KeyboardControlGroup(
      new KeyboardControl('f'),
      new KeyboardControl('k', { shiftKey: true })
    ),
    last: new KeyboardControlGroup(
      new KeyboardControl('j', { shiftKey: true }),
      new KeyboardControl('l')
    ),
    next: new KeyboardControlGroup(
      new KeyboardControl('j'),
      new KeyboardControl('n')
    ),
    previous: new KeyboardControlGroup(
      new KeyboardControl('k'),
      new KeyboardControl('p')
    )
  }).setOptions({
    onMatchOptions: {
      preventDefault: true,
      stopImmediatePropagation: true
    }
  });

@ViewChild('feedEl', { static: false })
private readonly feedEl: ElementRef;
@ViewChildren(FeedItemComponent)
private readonly feedItemComponents: QueryList<FeedItemComponent>;
private focusedItem: HTMLElement;

constructor(@Inject(DOCUMENT) private readonly document: Document) { }

@HostListener('keydown', ['$event'])
public handleItemFocusChange(event: KeyboardEvent): void {
  if (!this.feedEl.nativeElement.contains(this.document.activeElement)) {
    return;
  }
  const direction = FeedComponent.KB_CONTROLS.match(event);
  if (direction) {
    this.moveItemFocus(direction);
  }
}

public updateFocusedItem(item: HTMLElement): void {
  this.focusedItem = item;
}

private moveItemFocus(dir: FeedControls): void {
  let target: Element|null = null;
  if (dir === 'next' || dir === 'previous') {
    target = this.focusedItem ? this.focusedItem[`${dir}ElementSibling`] : null;
  } else if (dir === 'last' || dir === 'first') {
    target = this.feedItemComponents[dir].elementRef.nativeElement;
  }
  if (target instanceof HTMLElement &&
    this.feedEl.nativeElement.contains(target)) {
    target.focus();
    target.scrollIntoView();
  }
}

```

Kuvio 30. Näppäimistökomentolisäykset esimerkkisovelluksen FeedComponent-komponentin luokkaan

```

<div class="feed">
  <div
    #feedEl
    [attr.aria-busy]="!items.length || feedStatus.busy"
    [attr.aria-label]="label || null"
    [infiniteScrollContainer]="infiniteScrollContainer"
    [infiniteScrollDisabled]="disableInfiniteScroll"
    [infiniteScrollDistance]="scrollDistance"
    infiniteScroll
    (scrolled)="onScroll()"
    role="feed">
    <app-feed-item
      *ngFor="let item of items; index as i; trackBy: feedItemTrackBy"
      [setSize]="setSize"
      [posInSet]="i+1"
      [item]="item"
      (focuswithin)="updateFocusedItem($event)">
    </app-feed-item>
  </div>
</div>

```

Kuvio 31. Näyte esimerkisovelluksen FeedComponent-komponentin päivitetystä HTML-rakenteesta

Myös FeedItemComponent-komponentin luokkaa (ks. Kuvio 28) päivitetään (ks. Kuvio 32) lähettämään referenssi kyseisen komponentin HTML-isäntäelementistä FeedComponent-komponentille, kun fokus siirtyy FeedItemComponent-komponentin HTML-rakenteen sisällä. Tällöin FeedComponent-komponentissa voidaan pysyä ajan tasalla siitä, minkä article-elementin kohdalla fokus on, jolloin näppäimistökomendoilla seuraavaksi fokusoitavan article-elementin selvittäminen käy helposti. Vaihtoehtoisesti saman toiminnallisuuden voisi yksinkertaisemmin toteuttaa esimerkiksi hyödyntämällä ”:focus-within” -CSS-valitsinta FeedComponent-komponentin feed-elementin querySelector-metodilla, mutta testaamiseen käytetyistä verkkoselaimista (ks. Taulukko 8) Microsoft Edge ei tue kyseistä CSS-pseudoluokkaa (:focus-within n.d.).

```

@Output() public readonly focuswithin: EventEmitter<HTMLElement> = new EventEmitter();

constructor(public readonly elementRef: ElementRef) { }

@HostListener('focusin', ['$event'])
public onFocusWithin(): void {
  this.focuswithin.emit(this.elementRef.nativeElement);
}

```

Kuvio 32. Lisäykset esimerkisovelluksen FeedItemComponent-komponentin luokkaan

## Näppäimistökomentojen dokumentoiminen

Feed-roolin elementissä tarjottaville näppäimistökomennoille ei ole yleistä standardia käytettävien näppäimien suhteen, joten mahdolliset näppäinkomennot suositellaan esitettävän käyttäjälle helposti löydettävällä tavalla (WAI-ARIA Authoring Practices 1.1 2019). Täten FeedComponent-komponenttiin määritetyt näppäimistökomennot (ks. Kuvio 30) dokumentoidaan kyseisen elementin HTML-rakenteeseen (ks. Kuvio 33).

```
<div class="feed">
  <p [appVHidden]="true" class="feed_instructions" tabindex="0">
    <b>When inside the feed: </b>
    <span class="feed_instruction">Use <b>N or J</b> to move to the next item. </span>
    <span class="feed_instruction">Use <b>P or K</b> to move to the previous item. </span>
    <span class="feed_instruction">Use <b>L or Shift+J</b> to move to the last item. </span>
    <span class="feed_instruction">Use <b>F or Shift+K</b> to move to the first item.</span>
  </p>
  . . .
</div>
```

Kuvio 33. Näppäimistökomentojen dokumentaation lisäys esimerkksiovelluksen FeedComponent-komponentin HTML-rakenteeseen

Koska näppäimistökomennot on ensisijaisesti suunnattu näppäimistön käyttäjille, on esimerkksiovelluksessa päädytty visuaalisesti piilottamaan komentojen dokumentaatiota käärivä HTML-elementti, kunnes käyttäjä siirtää fokuksen siihen. Tällöin näppäimistökomentojen dokumentaatiota ei turhaan näytetä esimerkiksi mobiililaitteilla selaileville käyttäjille, jotka eivät oletettavasti pystyisi komentoja hyödyntämään. Elementin esittämislogiikka jätetään esimerkksiovelluksen AccessibilityModule-moduulin lisättävän VHiddenDirective-direktiivin (ks. Kuvio 34) vastuulle.

```

@Directive({ selector: '[appVHidden]' })
export class VHiddenDirective implements OnChanges {

  @HostBinding('class.a11y-vhidden') public hostVHidden: boolean = true;

  @Input('appVHidden') private readonly unlessFocused: boolean = false;
  private unlisten: (() => void)|null = null;

  constructor(
    @Inject(DOCUMENT) private readonly document: Document,
    private readonly el: ElementRef,
    private readonly renderer: Renderer2
  ) { }

  public ngOnChanges(changes: SimpleChanges): void {
    this.updateHidden();
    if (changes.unlessFocused) {
      const { previousValue, currentValue } = changes.unlessFocused;
      if (!previousValue && currentValue) {
        this.unlisten = this.renderer.listen(
          this.document,
          'focusin',
          this.updateHidden.bind(this)
        );
      } else if (!currentValue && previousValue && this.unlisten) {
        this.unlisten();
        this.unlisten = null;
      }
    }
  }

  public updateHidden(): void {
    if (this.unlessFocused) {
      const { nativeElement } = this.el;
      const { activeElement } = this.document;
      this.hostVHidden = !nativeElement.contains(activeElement);
    }
  }
}

```

Kuvio 34. Esimerkkisovelluksen VHiddenDirective-direktiivi

### 6.3.4 Yhteenveto

#### Menettely

Seuraavassa listassa kuvataan esimerkkisovelluksen abstraktimpi menettelytapa käyttäjän selatessa ääretöntä vieritystä hyödyntävää feed-roolin elementtiä näppäimistöä käyttäen.

1. Feed-roolin elementti on tyhjä, koska sen sisältöä ollaan vielä lataamassa, jolloin elementillä ei vielä ole yhtään sille pakollisia article-roolin lapsielementtejä. Täten elementin nykyinen ”kiireellinen” tila on merkattu avustavia teknologioita varten.



2. Sisällön lataaminen valmistuu ja feed-elementti täytetään article-roolin lapsielementeistä, jotka esittävät ladattua sisältöä. Feed-roolin elementin ”epäkiireellinen” tila merkataan avustavia teknologioita varten, jolloin ne voivat tarvittaessa reagoida tapahtuneisiin sisältömuutoksiin.
3. Käyttäjä hyödyntää näppäimistöä siirtääkseen fokuksen feed-elementin ensimmäiseen article-elementtiin. Article-elementin sisällön väliset suhteet on merkattu, jolloin avustavat teknologiat osaavat ilmoittaa käyttäjälle article-elementin oleelliset tiedot.
4. Käyttäjä siirtyy feed-elementin seuraavaan article-elementtiin hyödyntäen hänelle esitettyjä näppäimistökomentoja.
5. Feed-elementin seuraava article-elementti fokusoidaan ja verkkosivun näkymää vieritetään, jotta aktiivinen article-elementti tulee näkyviin. Koska käyttäjän fokus on tarpeeksi lähellä feed-elementin nykyisen sisällön loppua, ladataan feed-elementtiin lisää sisältöä, jotta käyttäjän siirtyminen article-elementtien välillä olisi sujuvaa.

## Käsitellyt onnistumiskriteerit

Taulukko 11. Yhteenveto äärettömälle vieritykselle valittujen onnistumiskriteerien käsittelystä

Onnistumiskriteeri	Taso	Miten huomioitu
<b>1.3.1: Info and Relationships</b>	A	Feed-elementin kaikilla article-elementeillä on niitä nimeävät ja kuvaavat elementit, jotka on yhdistetty kuhunkin article-elementtiin aria-labelledby- ja aria-describedby-attribuuteilla.
<b>2.1.4: Character Key Shortcuts</b>	A	Feed-elementin näppäimistökomennot ovat toiminnassa vain, kun feed-elementti tai joku sen lapsielementeistä on fokusoituna.
<b>4.1.1: Parsing</b>	A	Feed-elementin article-elementtejä kuvaileville elementeille generoidaan uniikit id-arvot.
<b>4.1.2: Name, Role, Value</b>	A	Feed-elementin aria-busy attribuuttia päivitetään tarvittaessa viestimään, että elementin HTML-rakennetta muokataan.

## 7 Pohdinta

Opinnäytetyön tavoitteena oli tutkia, kuinka moderneja SPA-sovelluksia voitaisiin toteuttaa W3C:n julkaisemien saavutettavuusohjeiden mukaisesti. Tutkimuksen perusteella SPA-sovelluksien yleisimmät saavutettavuusongelmatilanteet liittyvät yleisesti ottaen etenkin fokuksen hallintaan ja sisällön muutoksista viestimiseen. Tutkimustuloksena saatiin käytännön esimerkkejä erilaisten käyttöliittymäelementtien ja -toiminnallisuuksien saavutettavasta toteuttamisesta, jotka on perusteltu W3C:n saavutettavuusdokumentaatioiden kannalta. Tutkimustuloksia voidaan hyödyntää referenssinä toteutettaessa esimerkkien kaltaisia toiminnallisuuksia moderneihin web-sovelluksiin. Saavutettavuusongelmien merkitystä käyttäjänäkökulmasta pyrittiin tuomaan esiin ongelmia käsiteltäessä.

Tutkimuksessa tuotettujen esimerkkiratkaisujen oli määrä täyttää tutkittavaksi valittujen onnistumiskriteerien vaatimukset. WCAG 2.1:n onnistumiskriteerit ovat ajoittain hieman tulkinnanvaraisia, joten on vaikea sanoa, kuinka täydellisesti valittujen onnistumiskriteerien vaatimukset täytettiin. Onnistumiskriteerien vaatimuksien täyttämiseen kuitenkin kiinnitettiin paljon huomiota ja niiden käytännön tarkoitusta pyrittiin selventämään usean W3C:n julkaiseman dokumentaation avulla. Implementaatioiden toimivuutta ja saavutettavuutta testattiin paljon manuaalisesti, eivätkä myöskään hyödynnetyt saavutettavuuden evaluoimistyökalut löytäneet toteutuksista yhtäkään saavutettavuusongelmaa. On kuitenkin hyvä huomioida, että jotta verkkosivusto voidaan katsoa tietyn WCAG 2.1:n määrittämän saavutettavuustason mukaiseksi, tulee sivuston kaikki sivut olla kaikkien tähdätyn tason ja sitä alempien tasojen onnistumiskriteerien vaatimuksien mukainen (Web Content Accessibility Guidelines (WCAG) 2.1 2018).

Esimerkkiratkaisujen manuaaliseen testaamiseen käytettiin useita eri ruudunlukuohjelma- ja verkkoselainkombinaatioita. Testaamisessa huomattiin, että erilaisten kombinaatioiden toimivuus ja käyttäytyminen voi vaihdella hyvin paljon. Esimerkkiratkaisussa pyrittiin huomioimaan kombinaatioiden toimivuuden vaihtelevuus ja löytämään ratkaisuja, jotka toimisivat luotettavasti ja tarkoituksenmukaisesti kaikkien kombinaatioiden välillä. Kaikille testatuille kombinaatioilla ei kuitenkaan pystytty erinäisistä syistä tuottamaan tavoiteltua toimivuutta ja käytöstä, mutta myöskään merkittäviä puutteita ei lopullisissa ratkaisuissa havaittu.

Toteutetuissa esimerkkiratkaisuissa pyrittiin myös noudattamaan teknologiakohtaisia hyviä käytäntöjä. Hyvät käytänteet ovat usein pitkälti subjektiivisia, mutta toteutuksissa seurattiin pääasiassa Angularin virallista tyyliopasta, jonka perusteella kehitysympäristöön määritettiin useita TypeScript-koodin analyysisääntöjä käytäntöjen noudattamisen pakottamiseksi. Implementaatioille ei toteutettu yksikkö- tai muuta automaatiotestausta, jolloin niiden testattavuutta on vaikea arvioida. Osa ratkaisuista ei myöskään toimi sellaisenaan esimerkiksi hyödyntäessä Angularin palvelinpuolen renderointi (engl. server-side rendering (SSR)) -mahdollisuutta, jonka tukeminen on tosin pyritty pitämään mielessä abstraktoimalla implementaatioissa hyödynnettävät selainrajapinnat.

## 7.1 Mahdolliset jatkotutkimusaiheet

Yksikkö-, E2E- ja muut automaatiotestausmetodit ovat viime vuosina yleistyneet web-kehityksessä. *WCAG 2.1* määrittelee testattavia vaatimuksia sen jokaiselle onnistumiskriteerille ja ainakin osalle kriteereistä olisi oletettavasti hyödyllistä toteuttaa jonkunlaista automatisoitua testaamista. Kuitenkin osalle kriteereistä automaatiotestauksen toteuttamisen voi olla hyvin haastavaa ja epäkäytännöllistä, mikäli kyse on vähänkään monimutkaisemman verkkosivun testaamisesta. Mahdollinen jatkotutkimusaihe voisi siis olla saavutettavuuden automaatiotestauksen, sen mahdollisuuksien, haasteiden ja käytännöllisyyden tutkiminen.

## Lähteet

:focus-within. N.d. Tietoa “:focus-within” -CSS-pseudoluokasta MDN Web Docs -verkkosivustolla. Viitattu 4.11.2019. <https://developer.mozilla.org/en-US/docs/Web/CSS/:focus-within>.

Accessibility of AJAX Applications. 2014. Tietoa Ajax-sovelluksien saavutettavuudesta WebAIM-verkkosivustolla. Viitattu 21.4.2019. <https://webaim.org/techniques/ajax>.

Accessible Rich Internet Applications (WAI-ARIA) 1.1. 2017. Tekninen spesifikaatio verkkosisällön ja web-sovelluksien saavutettavuuden edistämiseksi W3C:n verkkosivulla. Viitattu 1.10.2019. <https://www.w3.org/TR/wai-aria-1.1/>.

Accessible Rich Internet Applications (WAI-ARIA) 1.2. 2019. Accessible Rich Internet Applications (WAI-ARIA) -dokumentin version 1.2 luonnos. Viitattu 9.10.2019. <https://w3c.github.io/aria/>.

AfterViewInit. N.d. Dokumentaatiota Angularin AfterViewInit-elämänkaarikoukusta Angularin verkkosivulla. Viitattu 7.11.2019. <https://angular.io/api/core/AfterViewInit>.

Angular Infinite Scroll. N.d. ngx-infinite-scroll -Angular-direktiivin ohjeistus projektin GitHub-sivulla. Viitattu 3.11.2019. <https://github.com/orizens/ngx-infinite-scroll/blob/master/README.md>.

ARIA live regions. N.d. Ohjeistus ARIA-live attribuuttien käyttämisestä MDN Web Docs -verkkosivustolla. Viitattu 11.8.2019. [https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA\\_Live\\_Regions/](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions/).

CSS in Action: Invisible Content Just for Screen Reader Users. N.d. CSS-tyylioheitekoita HTML-sisällön esittämiseen vain ruudunlukijaohjelmien käyttäjille. Viitattu 10.10.2019. <https://webaim.org/techniques/css/invisiblecontent/>.

Digitaalisten palvelujen ensisijaisuus. N.d. Tietoa digitaalisten palvelujen ensisijaisuudesta. Viitattu 13.8.2019. <https://vm.fi/digipalvelujen-ensisijaisuus/>.

Flanagan, D. 2011. JavaScript: The Definitive Guide. 6. p. O'Reilly Media, Inc.

Henry, S. & McGee, L. N.d. Accessibility. Viitattu 17.4.2019. <https://www.w3.org/standards/webdesign/accessibility>.

HTML Standard. N.d. HTML5:n spesifikaatio WHATWG:n verkkosivulla. Viitattu 3.11.2019. <https://html.spec.whatwg.org/>.

Kananen, J. 2008. Kvali: Kvalitatiivisen tutkimuksen teoria ja käytänteet. Jyväskylä: Jyväskylän ammattikorkeakoulu.

L 327/84. Annettu 20.12.2018. Commission Implementing Decision (EU) 2018/2048 of 20 December 2018 on the harmonised standard for websites and mobile applications drafted in support of Directive (EU) 2016/2102 of the European Parliament and of the Council. Viitattu 14.8.2019. [https://eur-lex.europa.eu/eli/dec\\_impl/2018/2048/oj/](https://eur-lex.europa.eu/eli/dec_impl/2018/2048/oj/).

Miksi saavutettavuus on tärkeää? N.d. Tietoa saavutettavuuden tärkeydestä Celian tuottamalla Saavutettavasti-sivustolla. Viitattu 20.4.2019. <https://www.saavutettavasti.fi/tietoa-saavutettavuudesta/miksi-saavutettavuus-on-tarkeaa/>.

Saavutettavuus. N.d. Tietoa saavutettavuusdirektiivistä Suomen Valtiovarainministeriön verkkosivulla. Viitattu 10.8.2019. <https://vm.fi/saavutettavuusdirektiivi/>.

Screen Reader User Survey #8 Results. 2019. Ruudunlukuohjelmien käyttäjätutkimuksen tulokset WebAIM-verkkosivustolla. Viitattu 7.10.2019. <https://webaim.org/projects/screenreadersurvey8/>.

Top sites. N.d. Päivittyvä listaus Internetin suosituimmista verkkosivustoista. Viitattu 10.7.2019. <https://www.alexa.com/topsites/>.

Understanding Conformance. N.d. Tietoa Web Content Accessibility Guidelines 2.1 -dokumentin määrittämien onnistumiskriteerien vaatimuksien täyttämisestä W3C:n verkkosivulla. Viitattu 8.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/conformance>.

Understanding Success Criterion 1.3.1: Info and Relationships. N.d. Tarkempi kuvaus onnistumiskriteeristä 1.3.1: Info and Relationships W3C:n verkkosivulla. Viitattu 8.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/no-keyboard-trap.html>.

Understanding Success Criterion 2.1.2: No Keyboard Trap. N.d. Tarkempi kuvaus onnistumiskriteeristä 2.1.2: No Keyboard Trap W3C:n verkkosivulla. Viitattu 8.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/no-keyboard-trap.html>.

Understanding Success Criterion 2.1.4: Character Key Shortcuts. N.d. Tarkempi kuvaus onnistumiskriteeristä 2.1.4: Character Key Shortcuts W3C:n verkkosivulla. Viitattu 7.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/character-key-shortcuts.html>.

Understanding Success Criterion 2.4.1: Bypass Blocks. N.d. Tarkempi kuvaus onnistumiskriteeristä 2.4.1: Bypass Blocks W3C:n verkkosivulla. <https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks.html>.

Understanding Success Criterion 2.4.2: Page Titled. N.d. Tarkempi kuvaus onnistumiskriteeristä 2.4.2: Page Titled W3C:n verkkosivulla. Viitattu 16.8.2019. <https://www.w3.org/WAI/WCAG21/Understanding/page-titled.html>.

Understanding Success Criterion 2.4.3: Focus Order. N.d. Tarkempi kuvaus onnistumiskriteeristä 2.4.3: Focus Order W3C:n verkkosivulla. Viitattu 12.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/focus-order>.

Understanding Success Criterion 2.4.7: Focus Visible. N.d. Tarkempi kuvaus onnistumiskriteeristä Understanding Success Criterion 2.4.7: Focus Visible W3C:n verkkosivulla. Viitattu 9.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/focus-visible.html>.

Understanding Success Criterion 3.2.1: On Focus. N.d. Tarkempi kuvaus onnistumiskriteeristä Understanding Success Criterion 3.2.1: On Focus W3C:n verkkosivulla. Viitattu 8.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/on-focus.html>.

Understanding Success Criterion 3.2.5: Change On Request. N.d. Tarkempi kuvaus onnistumiskriteeristä Understanding Success Criterion 3.2.5: Change On Request W3C:n verkkosivulla. Viitattu 8.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/change-on-request.html>.

Understanding Success Criterion 4.1.3: Status Messages. N.d. Tarkempi kuvaus onnistumiskriteeristä Understanding Success Criterion 4.1.3: Status Messages W3C:n verkkosivulla. Viitattu 10.10.2019. <https://www.w3.org/WAI/WCAG21/Understanding/status-messages.html>.

WAI-ARIA Authoring Practices 1.1. 2019. Ohjeistusdokumentti WAI-ARIA 1.1:n soveltamiseen W3C:n verkkosivulla. Viitattu 1.10.2019. <https://www.w3.org/TR/wai-aria-practices-1.1/>.

Web Content Accessibility Guidelines (WCAG) 2.1. 2018. Ohjeistusdokumentti verkkosisällön saavutettavuuden edistämiseen W3C:n verkkosivulla. Viitattu 15.8.2019. <https://www.w3.org/TR/WCAG21/>.

Yleistä tietoa saavutettavuudesta. N.d. Tietoteksti Celian verkkosivulla. Viitattu 17.4.2019. <https://www.celia.fi/saavutettavuus/>.