SAAVUTETTAVUUS MOBIILIKEHITYKSESSÄ, VERTAILUSSA REACT NATIVE JA FLUTTER

Case: Ahola Digital

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
kevät, 2023

Sanna Koskiranta-Nurmi

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Tekijä Sanna Koskiranta-Nurmi Vuosi 2023

Työn nimi Saavutettavuus mobiilikehityksessä, vertailussa React Native ja Flutter
Case: Ahola Digital

Ohjaaja Pentti Ojaniemi

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli selvittää keinoja, joilla saavutettavuutta voidaan huomioida ja kehittää sovelluskehityksessä. Erityisesti mielenkiinnon kohteena olivat mobiilisovellukset ja saavutettavuus ruudunlukijaa käytettäessä. Opinnäytetyön toimeksiantajan, Ahola Digital Oy Ab, tarpeista johtuen saavutettavuutta ruudunlukijalla vertailtiin erityisesti kahden teknologian - React Nativen ja Flutterin välillä.

Opinnäytetyön tietopohja koostuu kansainvälisen saavutettavuutta ohjaavan WCAG-kriteeristöstä. Saavutettavuutta käsitellään yleisellä tasolla sekä mobiilikehityksen että ruudunlukijan käytön näkökulmasta. Lisäksi käsitellään sitä, kuinka ruudunlukijaa käyttävät henkilöt käyttävät verkkopalveluita ja mobiilisovelluksia ja kuinka mobiilisovelluksia voidaan testata ruudunlukijalla. Opinnäytetyö on toiminnallinen ja siinä tehtiin yksinkertaiset sovellukset React Nativella ja Flutterilla, sovelluksia testattiin ruudunlukijalla ja testeistä tehtiin raportti.

Johtopäätöksenä voidaan todeta, että saavutettavan sovelluksen kehittäminen vaatii paljon tietoa siitä, miten ruudunlukijaa käytetään, käytettävissä olevista ohjelmistokehitysmenetelmistä sekä osaamista ruudunlukijatestauksessa ja tulosten raportoinnissa. Lopputuloksen kannalta merkittäviä eroja React Nativen ja Flutterin välillä ei tässä työssä havaittu. Joitain eroja havaittiin Android- ja iOS-laitteiden toiminnan välillä.

Avainsanat Saavutettavuus, ruudunlukija, mobiilikehitys, React Native, Flutter, testaaminen.

Sivut 51 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology Abstract

Author Sanna Koskiranta-Nurmi Year 2023

Subject Accessibility in mobile development, comparing React Native and Flutter
Case: Ahola Digital

Supervisor Pentti Ojaniemi

ABSTRACT

The purpose of this thesis was to find out ways in which digital accessibility can be taken into consideration and improved within software development. A more detailed attention was given to mobile applications and accessibility with screen readers. The client of the thesis, Ahola Digital Oy Ab, was especially interested in comparing two cross-platform technologies – React Native and Flutter.

The basis of the thesis are the international criteria for accessibility, the WCAG-criteria. First, the central concepts of WCAG, mobile accessibility and screen readers are introduced. Then the focus is shifted towards how screen reader users use web-services on computer and mobile devices. Finally, the accessibility features of React Native and Flutter are introduced along with screen reader testing. In the practical part of the thesis, two simple applications were made with React Native and Flutter. The applications were then tested with screen reader and results were reported.

In conclusion, it can be stated that developing an accessible application requires a lot of knowledge on how people use screen readers, available software development methods and screen reader testing. The analysis demonstrates that there were no significant differences between the two technologies while comparing the accessibility of the applications made. Some differences were observed between Android and iOS devices.

Keywords Accessibility, screen reader, mobile development, React Native, Flutter, testing.

Pages 51 pages and appendices 1 page

Sanasto

AG WG	Accessibility Guidelines Working Group on työryhmä, joka toimii W3C WAI-projektissa.
Flutter	Googlen luoma avoimen lähdekoodin käyttöliittymäohjelmiston kehityspaketti, jolla voidaan kehittää sovelluksia useille eri käyttöjärjestelmille
Käyttöliittymäelementti	Sovelluksen käyttöliittymässä näkyvä osa tai toiminto, esimerkiksi otsikko tai painike.
React Native	Facebookin (nyk. Meta) luoma avoimen lähdekoodin käyttöliittymäohjelmistokehys, jolla voidaan kehittää sovelluksia useille eri käyttöjärjestelmille.
Ruudunlukija	Avustava teknologia, jonka avulla ruudulla näkyvä tieto voidaan välittää kuunneltavassa muodossa.
Saavutettavuus	Verkkopalveluiden käytön mahdollistaminen myös avustavien teknologioiden (esim. ruudunlukija) kanssa.
Semantiikka	Käyttöliittymäelementin merkitys tai tarkoitus.
TalkBack	Android-järjestelmän oma ruudunlukija.
VoiceOver	MacOS- ja iOS-järjestelmien oma ruudunlukija.
WCAG-kriteerit	Web Content Accessibility Guideline – kansainvälinen standardi koskien verkkopalvelujen saavutettavuutta.
Widget	Pienoissovellus tai toiminto, joista Flutter-sovellus koostuu.
W3C	World Wide Web Consortium on kansainvälinen organisaatio, joka kehittää standardeja, jotta internet voisi palvella mahdollisimman laajaa käyttäjäkuntaa.
W3C WAI	W3C:n alaisuudessa toimiva Web Accessibility Initiative (WAI), jonka tavoitteena on tuottaa standardeja ja resursseja verkon saavutettavuuden hyväksi.

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

”Saavutettavuus on tekniikkaa, helppokäyttöisyyttä ja sisällön ymmärrettävyyttä” (Aluehallintovirasto, n.a.). Saavutettavuus digitaalisissa palveluissa tarkoittaa sitä, että palvelua voi käyttää kuka tahansa mahdollisista toimintarajoitteistaan tai vammoistaan huolimatta. Käytännön tasolla digitaalisen palvelun saavutettavuuden tasoa voidaan nostaa lukuisilla tavoilla. Yksi niistä on mahdollistamalla palvelun käyttö teknisten apuvälineiden, kuten ruudunlukijan, avulla. Saavuttavuuden varmistaminen sovelluskehityksessä on tärkeä askel kohti yhdenvertaista digitaalista maailmaa.

Ohjelmistotalo Ahola Digitalissa kehitetään mm. julkisen sektorin palveluihin liittyvää mobiilisovellusta, jonka käyttäjillä voi olla jokin vamma tai toimintarajoite. Nykyisillä teknologioilla saavutettavuuden kehittäminen on todettu työlääksi ja aikaa vieväksi, joten teknologian vaihto on ajankohtaista. Uusiksi vaihtoehdoiksi on ennakkoon valikoitu React Native ja Flutter. Tässä opinnäytetyössä tulen tarkastelemaan saavutettavuuskriteereitä yleisesti ja mobiilikehityksen näkökulmasta sekä valittuja mobiiliteknologioita saavutettavuuden ja ruudunlukijan käytön näkökulmasta. Molemmilla teknologioilla tehdään yksinkertaiset prototyypit ja niiden toimintaa vertaillaan erityisesti ruudunlukijaa käyttämällä. Tavoitteena on saada selville, onko toinen teknologia, React Native tai Flutter, nimenomaan ruudunlukijan kannalta parempi vaihtoehto.

Tutkimuskysymykset:

- Mitkä ovat tärkeimpiä saavutettavuusvaatimuksia mobiilisovellusten kehittämisessä?
- Miten saavutettavuutta ruudunlukijalla voidaan kehittää mobiilisovelluksissa?
- Miten saavutettavuutta ruudunlukijalla voidaan testata mobiilisovelluksissa?
- Onko React Nativessa ja Flutterissa eroja saavutettavuuden suhteen?

2 Digitaalisen palvelun saavutettavuus

Saavutettavuus digitaalisissa palveluissa tarkoittaa sitä, että palvelua voi käyttää kuka tahansa mahdollisista toimintarajoitteistaan tai vammoistaan huolimatta. Saavutettavuusvaatimuksilla halutaan taata ihmisten mahdollisimman yhdenvertainen kohtelu etenkin julkisten palveluiden suhteen. Saavutettavuus on terminä vakiintumassa nimenomaan verkkopalveluihin; esteettömyys -termillä puolestaan tarkoitetaan fyysisen ympäristön helppokulkuisuutta. Voidaan ajatella, että saavutettavuus on digitaalisen maailman esteettömyyttä. (Celia, n.d -a)

Kuva 1 havainnollistaa saavutettavuuden peruselementtejä. Nämä kolme tukijalkaa ovat: teknisesti virheetön toteutus, selkeä ja hahmotettava käyttöliittymä sekä ymmärrettävä sisältö. Tässä opinnäytetyössä käsitellään enimmäkseen osuutta **Teknisesti virheetön toteutus**, johon liittyvät WCAG-standardin noudattaminen, virheetön ja looginen html-koodi, tekninen saavutettavuus sekä avustavien teknologioiden ja näppäimistön käytön mahdollistaminen.
(Celia, n.d. -b)

Kuva 1. Saavutettava verkkopalvelu - Suunnittelun kolme tukijalkaa (Celia, n.d. -b).

2.1 Saavutettavuuden huomioiminen hyödyttää useimpia käyttäjiä

Saavutettavuuden huomioiminen kaikkien verkko- ja mobiilipalveluiden suunnittelussa helpottaa hyvin laajaa käyttäjäkuntaa. Maailmanlaajuisesti on arvioitu, että yli miljardilla ihmisellä on jonkin asteinen kehitysvamma tai muu vamma tai rajoite. Se vastaa noin 15 % maailman väestöstä. Maailman Terveysjärjestö WHO:n mukaan tämä luku tulee kasvamaan dramaattisesti, johtuen mm. kroonisten sairauksien lisääntymisestä sekä väestön ikääntymisen seurauksena. (World Health Organization (WHO), 2021)

Arvioiden mukaan yli miljoonalla suomalaisella on joitain haasteita digitaalisten palveluiden käytössä. Saavutettavuudesta hyötyvät kuitenkin myös ne, joille se ei ole välttämättömyys. Kuka tahansa voi elämänsä aikana kohdata tilanteen, jossa toimintakyky heikkenee väliaikaisesti tai pysyvästi. Toimintakyvyn heikkeneminen voi johtua myös ympäristöstä. Tästä esimerkkinä meluinen ympäristö, jolloin videon tekstitys hyödyttää useimpia käyttäjiä. (Aluehallintovirasto, n.d.a; World Health Organization (WHO), 2021)

Kehitysvammaliiton (n.d. -a) mukaan esimerkiksi seuraavilla vammoilla tai rajoitteilla voi olla vaikutusta verkkopalveluiden käyttöön:

- näkökykyyn liittyvät rajoitteet
- kuuloon liittyvät rajoitteet
- fyysiset ja motoriset rajoitteet
- kognitiiviset vaikeudet
- puhevammat
- neurologiset sairaudet

Näkökykyyn liittyviä rajoitteita voidaan helpottaa huomioimalla esimerkiksi riittävät värikontrastit, sisällön selkeä jaottelu, huomioimalla sisällön suurennustoiminnot ja mahdollistamalla ruudunlukijan sekä näppäimistön käyttö (Näkövammaisten liitto ry, 2019a). Kuuloon liittyvät rajoitteet puolestaan voivat aiheuttaa hankaluutta seurattaessa esimerkiksi videoesitystä, jossa ei ole tekstitystä, tai jos videon tai esityksen eri äänet kuuluvat päällekkäin tai puhujia on monta. (Kuuloliitto ry, n.d.)

Fyysisten ja motoristen rajoitteiden kirjo on laaja. Niihin kuuluvat esimerkiksi tahattomat liikkeet, vapina tai halvaantuminen sekä tuntoaistin rajoitteet, jotkin sairaudet (myös neurologiset sairaudet) ja raajojen puuttuminen. (Kehitysvammaliitto ry, n.d. -b) Näin ollen sivuston käyttö ei välttämättä ole mahdollista hiirellä, ja suunnittelussa olisikin hyvä ottaa huomioon, että toiminnot eivät vaadi suurta tarkkuutta. Elementtien tulisi olla suuria ja käytön näppäimistöllä olla mahdollista. (Helsingin kaupunki, n.d.)

Kognitio viittaa henkilön tiedonkäsittelyyn, ja kognitiivisiin vaikeuksiin kuuluvat mm. muistiin, tarkkaavaisuuteen ja oppimiskykyyn liittyvät hankaluudet (Muistiliitto ry, n.d.). Erilaisia kognitiivisia rajoitteita on runsaasti, ja osa voi aiheutua jostakin neurologisesta sairaudesta tai poikkeavuudesta. Monille käyttäjille, joilla kognitio on heikentynyt, on hyötyä yleisesti hyvästä käytettävyydestä. Siihen liittyy palvelun selkeä ulkoasu sekä selkokielinen ja ytimekäs asiasisältö. Lisäksi voidaan mm. huolehtia, että käyttöliittymän rakenne on mahdollisimman selkeä, graafiset komponentit on nimetty johdonmukaisesti, eikä palvelun käyttö vaadi muistamaan mitä edellisillä sivuilla tai vaiheissa on tehty. (Kehitysvammaliitto ry, n.d. -c)

Myös erilaisissa muissa tilanteissa saavutettavuudesta on hyötyä. Käyttäjällä voi olla vanha päätelaite tai vanha verkkoselain, hidas internet-yhteys tai laitteiden ja internetin käyttäminen ei ole muuten sujuvaa. (Kehitysvammaliitto ry, n.d. -a)

2.2 Saavutettavuutta ohjaava lainsäädäntö - Digipalvelulaki

Digitaalisten palvelujen saavutettavuus on tärkeä osa-alue erityisesti julkisen sektorin palvelujen osalta. Laki digitaalisten palveluiden tarjoamisesta, eli digipalvelulaki, on Suomessa astunut voimaan 1.4.2019. Se määrää, että julkisen sektorin verkkosivujen ja mobiilisovellusten on oltava saavutettavia. Tämä koskee myös yksityisen sektorin palveluita niiltä osin, kuin sen tarjoamiin palveluihin sovelletaan digipalvelulakia. Laki perustuu Euroopan parlamentin ja neuvoston saavutettavuusdirektiiviin (EU) 2016/2102. (Laki digitaalisten palvelujen tarjoamisesta 306/2019)

Muita saavutettavuuteen ohjaavia lakeja ovat esimerkiksi perustuslaki, yhdenvertaisuuslaki sekä laki sähköisen viestinnän palveluista. Näiden lakien tavoitteena on varmistaa

kansalaisten yhdenvertainen kohtelu ja yhdenvertaiset mahdollisuudet käyttää mm. julkisia verkko- ja mobiilipalveluita. (Aluehallintovirasto, n.d.b)

Aluehallintoviraston (n.d. -c) mukaan Digipalvelulaki sisältää kolme päävaatimusta:

1. Digitaalisen palvelun – eli verkkosivuston tai mobiilisovelluksen – tulee täyttää saavutettavuusvaatimukset.
2. Palvelusta tulee laatia saavutettavuusseloste, jossa kuvataan palvelun saavutettavuuden tila ja mahdolliset puutteet.
3. Palvelun käyttäjille on tarjottava mahdollisuus antaa saavutettavuuspalautetta sähköisesti. Palautteeseen on annettava vastaus 14 vrk:n sisällä.

Käytännössä saavutettavuusvaatimukset tarkoittavat kansainvälistä Web Content Accessibility Guideline 2.1 -ohjeistusta (WCAG -kriteerit). Lisäksi digipalvelulaki määrää, että digitaalisen palvelun saavutettavuuden tila on arvioitava ja siitä luodaan saavutettavuusseloste. Arvioinnin ja selosteen voi joko tehdä itse tai tilata ulkopuoliselta toimijalta, mutta tämä tieto on tuotava esiin selosteessa. Selosteessa on kerrottava palvelun mahdolliset puutteet ja seloste täytyy olla helposti palvelun käyttäjän saatavilla esimerkiksi verkkosivuilla. Lisäksi käyttäjälle on tarjottava sähköinen keino palautteen antamiseksi saavutettavuuden osalta ja palautteeseen on vastattava 14 vrk:n sisällä. Mikäli palautteeseen ei saa vastausta tai vastaus ei ole tyydyttävä, Aluehallintovirastolle voi tehdä saavutettavuuskantelun. (Aluehallintovirasto, n.d.c, n.d.d)

2.3 WCAG 2.1 -saavutettavuusohjeistus

WCAG 2.1, eli Web Content Accessibility Guideline versio 2.1, on kansainvälinen ohjeistus verkkopalveluiden saavutettavuudelle. WCAG ohjeistusta ylläpitää ja kehittää saavutettavuusohjausryhmä (Accessibility Guidelines Working Group), joka puolestaan toimii World Wide Web Consortium, eli W3C, organisaation alla. Yksi W3C:n suunnitteluperiaatteista on ”Web for all” ja tähän liittyen on perustettu Web Accessibility Initiative (WAI), jonka toiminta keskittyy nimenomaan verkkopalveluiden mahdollistamiseen myös henkilöille, joilla on jokin vamma tai rajoite. WAI-aloitteen sisällä toimii mm.

saavutettavuusohjausryhmä (AG WG), joka ylläpitää ja kehittää WCAG-saavutettavuusohjeistusta. (WCAG 2.1, 2018b)

Vaikka verkkosisällön saavutettavuus on lainsäädännön myötä noussut suurempaan tietoisuuteen, se ei ole uusi keksintö. Ensimmäinen versio WCAG-ohjeista (WCAG 1.0) on julkaistu toukokuussa 1999. Ensimmäisessä versiossa keskityttiin lähinnä HTML-kieleen ja tuotiin kriteereihin edelleen käytössä olevat kolme vaatimustasoa: A, AA ja AAA, joista A-taso luo vähimmäisvaatimukset. WCAG 2.0 julkaistiin joulukuussa 2008. Siinä otettiin kantaa useampiin teknologioihin ja ohjattiin kehittäjiä saavutettavuuteen muidenkin kuin HTML-kielen osalta. Siinä otettiin käyttöön neljä periaatetasoa: havaittava, hallittava, ymmärrettävä ja toimintavarma. Nämä periaatteet ovat edelleen voimassa. (Accessible Web, 2020)

Opinnäytetyön kirjoitushetkellä uusin käytössä oleva versio on WCAG 2.1 -kriteeristö, jonka alkuperäinen englanninkielinen versio on julkaistu 5.6.2018 (WCAG 2.1, 2018b). Digipalvelulaki määrää, että digitallisen palvelun tulee täyttää WCAG 2.1 -version A ja AA-tason kriteerit (Aluehallintovirasto, n.d.c).

Kuten mainittu, WCAG-ohjeistuksessa verkkosisällön saavutettavuus on ylätasolla jaettu neljään pääperiaatteeseen: havaittavuus, hallittavuus, ymmärrettävyys ja toimintavarmuus. Jokainen periaate on jaettu pienenpiin ohjeisiin, joiden sisällä puolestaan on useampia testattavissa olevia onnistumiskriteereitä. Onnistumiskriteereihin on merkitty mihin tasoon kriteeri kuuluu - tasoon A, AA vai AAA. Tämän lisäksi englanninkielisenä löytyy ohjeistus riittäviin ja neuvoa-antaviin tekniikoihin, tyypillisisä virheitä ja esimerkkikoodeja. (WCAG 2.1, 2018b, 2019)

WCAG-periaatteiden sisällön lyhyt yhteenveto:

1. **Havaittava**
    a. Anna tekstivastineet ei-tekstuaaliselle sisällölle.
    b. Tekstitykset, kuvatekstit, kuvailutulkkaus tai muu vaihtoehtoinen tapa multimedian tarkasteluun.
    c. Luo sisältöä, joka voidaan esittää eri tavoin myös avustavien teknologioiden kautta, niin että merkitys pysyy samana.
    d. Tee sisällön näkeminen ja kuuleminen helpoksi käyttäjille.
2. **Hallittava**
    a. Kaikki toiminnot tulee olla käytettävissä myös näppäimistöä käyttäen.

   b. Anna käyttäjille tarpeeksi aikaa lukea ja käyttää sisältöä.
   c. Älä käytä sisältöä, joka voi aiheuttaa kohtauksia tai fyysisiä reaktioita.
   d. Auta käyttäjää navigoimaan ja löytämään sisältöä.
   e. Tee käyttäjälle helpoksi antaa syötteitä ohjelmaan, muutenkin kuin näppäimistöllä.
3. **Ymmärrettävä**
   a. Tee tekstistä helposti luettavaa ja ymmärrettävää.
   b. Tee sisällön näkyvyydestä ja toiminnoista ennakoitavaa.
   c. Auta käyttäjää välttämään ja korjaamaan virheitä.
4. **Toimintavarma**
   a. Maksimoi yhteensopivuus nykyisten ja tulevien käyttäjien työkalujen kanssa.

(WCAG 2.1, 2018a)

WCAG-kriteeristön nykyinen versio ei erikseen ota kantaa mobiilisovellusten saavutettavuuteen, vaan niihin sovelletaan samaa ohjeistusta, kuin muidenkin verkkopalveluiden kanssa. (WCAG 2.1, 2018b)

2.4 Kehitteillä olevat WCAG-ohjeistuksen versiot

Uusi versio WCAG-ohjeistuksesta, WCAG 2.2, on suunniteltu julkaistavaksi huhtikuussa 2023. Sen rakenne perustuu WCAG 2.1 -ohjeistukseen ja on taaksepäin yhteensopiva. WCAG 2.2 luonnosversio on W3C saavutettavuustyöryhmän tuottama dokumentti, joka on käynyt useita hyväksyntäkierroksia, mutta johon voi vielä tulla muutoksia ennen varsinaista julkaisua (W3C, n.d. §3.2.2).

Seuraavia onnistumiskriteereitä odotetaan tulevaksi uusina WCAG 2.2 -ohjeistukseen:

- Kohdistimen ulkonäköön (kontrasti ja paksuus) liittyvät vaatimukset (WCAG kriteeri 2.4.11, AA-taso)
- Elementin kohdistin ei saa peittyä osittain tai kokonaan (WCAG-kriteeri 2.4.12 AA-taso ja 2.4.13 AAA-taso)
- Vaihtoehto elementtien vetoliikkeille (WCAG kriteeri 2.5.7, A-taso)
- Kohdistettavan elementin vähimmäiskoko (WCAG 2.5.8, AA-taso).
- Johdonmukainen käyttöapu (WCAG kriteeri 3.2.6, A-taso)
- Saavutettava tunnistautuminen (WCAG kriteeri 3.3.7 AA-taso ja 3.3.8 AAA-taso)

- Käyttäjältä aiemmin pyydetyn tiedon tarpeeton uudelleensyöttö (WCAG-kriteeri 3.3.9, A-taso).
  (WCAG 2.2, 2023)

Kehitteillä on myös kokonaan uusittu WCAG 3 -ohjeistus, jonka ei oleteta valmistuvan vielä muutamaan vuoteen. WCAG 3 -versiosta on tarkoitus tehdä helpommin ymmärrettävä ja siinä on tarkoitus ottaa huomioon enemmän käyttäjien rajoitteita tai vammoja sekä erityyppisiä verkkosisältöjä. WCAG 3 -kriteeristöön on suunniteltu erilainen rakenne, erilainen vaatimustenmukaisuus-malli ja verkkosisältöä laajempi soveltamisala. (W3C, 2022)

3 Saavutettavuuden huomioiminen mobiilisovelluksissa

Kuten edellä mainittiin, WCAG-ohjeistus ei erikseen ota kantaa mobiilisovellusten saavutettavuuteen. Mobiilisovelluksia koskevat samat vaatimukset kuin verkkosivujakin. WCAG-ohjeiden tueksi on kuitenkin tuotu erikseen liite mobiilisaavutettavuutta varten - Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile, jossa tuodaan esiin kootusti WCAG-ohjeistuksen vaikutuksia mobiilisovelluksille. Se sisältää ohjaavia käytänteitä, kuinka WCAG-ohjeistusta voidaan soveltaa mobiililaitteissa toimivissa sovelluksissa ja verkkosovelluksissa, verkkosisällöissä, natiivi- ja hybridisovelluksissa. Alla on ko. liitteen perusteella tehty tiivistetty listaus mobiilisaavutettavuudessa erityisesti huomioitavista seikoista, joiden sisältöä on avattu seuraavissa kappaleissa.

1. **Havaittava**
    a. Pieni ruudun koko.
    b. Zoomaus eli suurennus ja loitonnus.
    c. Kontrasti.
2. **Hallittava**
    a. Ulkoisen näppäimistön käyttö kosketusnäytöllisen laitteen kanssa.
    b. Kosketuselementtien koko ja etäisyys toisistaan.
    c. Kosketusnäytön ohjauseleet.
    d. Laitteen manipuloinnin eleet.
    e. Painikkeiden sijoittaminen ruudulle helppokäyttöisiksi.
3. **Ymmärrettävä**
    a. Mahdollisuus näytön suunnan muuttaminen pysty- tai vaaka-asentoon.
    b. Toimintojen johdonmukainen asettelu.
    c. Tärkeiden elementtien asettelu niin, että sivua ei tarvitse vierittää.
    d. Elementin toiminto täytyy olla selvästi osoitettu.
    e. Muokattujen kosketuseleiden käyttöön on annettava ohjeet.
4. **Toimintavarma**
    a. Aseta virtuaalinen näppäimistö vaaditun datatyypin mukaan.
    b. Anna käyttäjälle helppoja tiedonsyöttötapoja.
    c. Tue alustakohtaisia ominaisuuksia.

(W3C, 2015)

3.1 Havaittavuuteen liittyviä saavutettavuuskäytänteitä

Koska mobiililaitteissa on pieni näyttö verrattuna tietokoneisiin, kullekin sivulle annettavan tiedon määrä on hyvä minimoida. Jos mobiiliversio suunnitellaan responsiiviseksi, voidaan sama sisältö tuoda käyttäjälle esiin mobiililaitteelle paremmin sopivassa muodossa. Esimerkiksi päävalikko voidaan tuoda sivusta avattavan valikkopainikkeen taakse (ns. hampurilaisvalikko).

Suurennus ja loitonnustoiminnot ovat useimmiten mobiililaitteissa sisäänrakennettuja, ja monissa laitteissa käytössä olevaa elettä ”pinch zoom” (tarkennus tai laajennus kahdella sormella) ei saa estää. Lisäksi käyttäjälle voi tarjota mahdollisuuden suurentaa tekstiä sivuston tai sovelluksen asetuksissa ja tukea käyttöjärjestelmätasolla käyttäjän itse muokkaamaa tekstin kokoa.

Kontrastin suhteen mobiilikäytössä olisi syytä olla tarkempi kuin tietokoneversiossa, nimenomaan siksi että ruutu on huomattavasti pienempi. WGAC-ohjeiden kontrastin raja-arvot AA-tasolla ovat pienelle tekstille 4,5:1 tai 3:1 suurelle tekstille. Kontrastin raja-arvot ovat AAA-tasolla pienelle tekstille 7:1 tai 4,5:1 suurelle tekstille. Pieneksi tekstiksi katsotaan alle 18pt normaali fontti tai alle 14pt lihavoitu fontti. Suureksi tekstiksi katsotaan 18pt tai suurempi normaali fontti sekä 14pt tai suurempi lihavoitu fonttikoko. Mobiililaitteissa tekstiä voidaan pitää suurena, jos se on käyttöalustan tavanomaista fonttikokoa 120 % suurempi ja lihavoitu tai 150 % suurempi.

(W3C, 2015)

3.2 Hallittavuuteen liittyviä saavutettavuuskäytänteitä

Ulkoisen näppäimistön käyttö on tehtävä mahdolliseksi myös kosketusnäyttöisissä laitteissa ja sovelluksissa. Useimpiin mobiililaitteisiin voi yhdistää näppäimistön Bluetooth- tai USB-yhteyden kautta. Ulkoisen näppäimistön tukeminen mobiilisti auttaa esimerkiksi henkilöitä, joilla on haasteita sorminäppäryyden tai liikkuvuuden kanssa. Kun sovellusta käytetään näppäimistöllä (tai ruudunlukijalla) kohdistimen liikejärjestyksen täytyy olla selkeä ja kohdistimen täytyy olla koko ajan selkeästi näkyvillä. Lisäksi sisällössä ei saa olla ns. näppäimistöansaa, eli näppäimistön fokus ei saa juuttua mihinkään elementtiin. (W3C, 2015)

WCAG-ohjeistuksen mukaan kosketusalueiden tulee olla tarpeeksi suuria ja riittävän erillään toisistaan, jotta käyttäjä ei vahingossa osu väärään painikkeeseen. Kosketusalueen pitäisi laitteen näytöllä olla vähintään 9mm * 9mm. (W3C, 2015) Kuitenkin tuntoaistin mekaniikkaa koskevassa tutkimuksessa on selvitetty, että ihmisen keskimääräinen sormenpään halkaisija on 16-20mm (Dandekar ym., 2003, s. 685). Googlen Material Design suosittaa Androidille kosketusalueen kooksi vähintään 48 * 48 dip (density independent pixel) ja kosketusalueiden

väliin vähintään 8 dp leveää aluetta (Google, n.d. -a). Apple puolestaan suosittelee käytettäväksi iOS-sovelluksissa vähintään 44 * 44 pisteen kokoista aluetta, mutta ei anna suosituksia kosketusalueita erottavalle alueelle (Apple Inc., n.d.c). Nämä ovat kutakuinkin samankokoiset alueet, vaikka Android- ja iOS-alustoilla on hieman erilainen tapa ilmoittaa alueiden koko ja laskea pikseleitä. Molemmista tulee n. 9 * 9 mm kokoinen alue laitteen näytölle. (Kearney, 2015)

Kosketusnäytön eleet sovelluksessa pitäisi olla mahdollisimman yksinkertaiset ja mikäli sovellus käyttää muokattuja kosketuseleitä, niistä on ohjeistettava käyttäjää. Jos sovelluksessa voidaan käyttää esimerkiksi laitteen ravistamista tai kallistamista jonkin toiminnon suorittamiseen, sen suorittaminen on voitava tehdä myös käyttäen tavanomaisia kosketuseleitä tai näppäimistöllä. Lisäksi käyttäjää on autettava välttämään virheitä. Tässä voi auttaa mm. elementtien aktivointi hiiren klikkauksen noustessa (mouseUP event) tai kosketuksen päättyessä (touchEnd event). Käyttäjä ehtii virheellisen valinnan huomatessaan viedä fokuksen pois elementin päältä, ja näin välttää virheellisen valinnan tekemisen. (W3C, 2015). Virheen sattuessa siitä on informoitava käyttäjää. Yksi tapa on ilmoittaa virheestä huomiodialogilla (alert-dialog). Huomiokentän kuuluu saada ruudunlukijan fokus automaattisesti ja kertoa ainakin mikä virhe on kyseessä - mahdollisesti myös miten virheen voi korjata. Kun huomiokenttä kuitataan, fokuksen pitäisi mahdollisuuksien mukaan siirtyä takaisin missä se oli ennen huomiokentän ilmestymistä. (AG WG, 2023)

Painikkeiden tulee olla sijoitettuna niin, että niiden käyttö on helppoa, tapahtui käyttö sitten yhdellä tai kahdella kädellä, vasen- tai oikea-kätisesti. Käyttäjällä voi myös olla rajoittunut liikerata peukalossa tai muissa sormissa. Joissain mobiilikäyttöjärjestelmissä on esimerkiksi mahdollista siirtää väliaikaisesti näyttöä vaaka- tai pysty suunnassa, jotta yhdellä kädellä käyttö sujuisi helposti. Tämä voi olla yksi ratkaisutapa painikkeiden sijoiteluun. (W3C, 2015)

3.3 Ymmärrettävyyteen liittyviä saavutettavuuskäytänteitä

Vaatimus käytettävyydestä vaaka- ja pystyasennossa johtuu siitä, että esimerkiksi jotkut henkilöt voivat fyysisten rajoitteiden vuoksi käyttää puhelinta vain toisessa asennossa. Laite voi olla esimerkiksi kiinnitettynä pyörätuoliin. Mikäli sovellusta ei voi käyttää kuin yhdessä asennossa, ruudunlukijan täytyy saada siitä tieto, jotta käyttäjä ei tee vahingossa vääriä

ohjauseleitä. Sovelluksen toistuvien toimintojen tulee olla johdonmukaisia eri näkymissä, kun sovellusta käytetään samankokoisella näytöllä. Näytön asettelu voi muuttua, jos sovellusta käytetään näytön ollessa vaaka- tai pystyasennossa, mutta näkyvien osien ja toimintojen tulee olla johdonmukaisia käytettäessä sovellusta tietyssä laitteen asennossa. (W3C, 2015)

Sivun tärkeimmät elementit pitäisi voida lukea ennen kuin sivua joutuu vierittämään. Tämä auttaa mm. niitä henkilöitä, jotka käyttävät näytön suurennustoimintoa. Näin he eivät joudu vierittämään ja suurentamaan sisältöä, vaan voivat tarkastaa sovelluksen tärkeimmät osat yhdellä kertaa. (W3C, 2015)

Elementit, jotka tekevät saman toiminnon - kuten linkkiteksti ja siihen liittyvä ikoni, on ryhmiteltävä yhteen. Tällä saadaan isompi kosketusalue ja vältytään avustavien teknologioiden osalta usealta fokuksen saavalta kohteelta. (W3C, 2015)

Toiminnallisia elementtejä ovat esimerkiksi painikkeet ja linkit. Niiden on erotuttava muista näytöllä olevista elementeistä sekä visuaalisesti että ohjelmallisesti. Suosituksena on, että noudatetaan käyttäjille ennestään tuttua tyypillistä visuaalista suunnittelua, elementtien asettelua, sekä värityylien ja ikonien käyttöä. Tämä helpottaa sovelluksen tai sivuston käyttöä. Esimerkiksi painikkeilla, pudotusvalikoilla ja valintaruuduilla on tyypilliset visuaaliset ilmeet sekä ohjelmallinen rooli, jotka auttavat eri käyttäjiä tunnistamaan nämä elementit. Tyypillinen valikkopainike on kolme päällekkäistä viivaa, ns. hampurilaisvalikko. Tavanomainen tyyli linkeille on värillinen ja alleviivattu teksti. Sisällössä takaisin-painike on tavanomaisesti vasemmalle osoittava nuoli, joka on asetettu näytön vasempaan yläkulmaan. (W3C, 2015)

Jos mobiilisisällössä käytettään muokattuja kosketuseleitä, niiden käyttö on ohjeistettava. Lisäksi toiminnot on voitava suorittaa myös tavanomaisia kosketuseitä ja näppäimistöä käyttämällä. Tämä liittyy WCGA kriteereihin 3.3.2 Nimilaput ja ohjeet sekä 3.3.5 Ohjeet. Tekstinsyöttökentillä täytyy olla aina kuvaavat nimilaput (label) ja mahdollisesti myös ohjeet, etenkin jos kentässä halutaan käyttäjältä tietynlaista tietoa tai se pitää syöttää tietyssä muodossa, kuten päivämäärä muodossa PP.KK.VVVV. Ohjeet on oltava saavilla aina kun käyttäjä niitä tarvitsee. (W3C, 2015)

3.4 Toimintavarmuuteen liittyviä saavutettavuuskäytänteitä

Toimintavarmuus-periaatteeseen liittyy oikean virtuaalisen näppäimistön asettaminen vaaditun tietotyypin mukaan. Esimerkiksi jos käyttäjän on annettava puhelinnumero, tiedonsyöttöön asetetaan virtuaalinen numeronäppäimistö. Näin syötteeseen ei vahingossa tule esimerkiksi kirjaimia ja virheiden määrä voidaan vähentää. (W3C, 2015)

Tiedon syöttämiseen pitäisi olla helppoja tapoja ja varsinaisen tekstin kirjoittaminen millä tahansa välineessä olisi hyvä minimoida. Tiedon antamiseen suositellaan mahdollisuuksien mukaan valintapainikkeita, valintaruutuja, pudotusvalikoiden käyttöä sekä automaattista tiedon syöttöä tunnettujen tietojen kohdalla (kuten päivämäärä, kellonaika ja sijainti). (W3C, 2015)

W3C suosittelee, että mobiilisovelluksessa tai verkkosisällössä tuetaan kunkin alustan (iOS, Android, jne.) tunnusomaisia piirteitä esimerkiksi suurennustoimintoihin ja fonttikoon valitsemiseksi. (W3C, 2015)

3.5 Saavutettavuus React Native-teknologiassa

React Native (v. 0.71) on avoimen lähdekoodin JavaScript-sovelluskehys, jolla voidaan kirjoittaa natiiveja mobiilisovelluksia Android- ja iOS-alustoille. React Native on alun perin Facebookin (nykyinen omistaja Meta) vuonna 2015 julkaisema sovelluskehys. Nykyisin Facebook on ylläpitäjä ja kehittäjäyhteisö tekee kehitystyötä. Yhteisötoteutuksilla tuetaan myös verkkosovelluksia, Windows-, MacOS- sekä Ubuntu-käyttöjärjestelmiä.
React Native-sovellusta rakennetaan funktioilla tai luokilla käyttäen komponentteja, joihin voidaan lisätä erilaisia ominaisuuksia (properties, props). React Nativella voidaan ylläpitää pääasiassa yhtä koodikantaa sekä Android- että iOS-alustoille. Dokumentaatiosta nähdään, että osa ominaisuuksista toimii vain toisella alustalla. (Meta Platforms Inc., 2022). Tässä työssä käsitellään viralliseen dokumentaatioon liitettyjä saavutettavuusominaisuuksia, vaikka muitakin kirjastoja voidaan käyttää React Nativella kehitettäessä. (Meta Platforms, Inc., 2022b)

React Nativessa on saavutettavuus-ohjelmarajapinnat (accessibility API) sekä Android- että iOS-alustoille. Näillä rajapinnoilla otetaan yhteys käyttöjärjestelmän omiin helppokäyttötoimintoihin, kuten ruudunlukijaan. React Native-dokumentaatiosta löytyy luettelo kussakin versiossa olevista saavutettavuusominaisuuksista (accessibility properties), joita voidaan käyttää eri komponenteissa. Verrattaessa esimerkiksi React Nativen versioita 0.70 (julkaistu 5.9.2022) ja 0.71 (julkaistu 12.1.2023), voidaan huomata, että saavutettavuusominaisuuksien määrä on lisääntynyt 16:sta 31:een. Osa koskee sekä iOSia että Androidia, jotkut vain toista alustaa. Jotkut ominaisuudet koskevat samaa toimintoa, mutta Androidille ja iOSille on omat nimetyt ominaisuudet, kuten myöhemmin huomataan. (Meta Platforms Inc., 2022). Uusia versioita on viime vuosina julkaistu noin 4 kertaa vuodessa, joten kehitys on jatkuvaa ja muutoksia voi tulla nopeasti.

Jos mikä tahansa näkymä tai erillinen komponentti halutaan tuoda saavutettavaksi, sille annetaan ominaisuus accessible={true}. Tähän liitetään ominaisuus accesssibilityLabel, jonka avulla ruudunlukijalle tuodaan elementin nimilappu. Kuten Ohjelmakoodi 1 React Native -saavutettavuusesimerkki. on esitetty, komponenttiin TouchableOpacity on liitetty ominaisuudet accessible={true} sekä accessibilityLabel="Tap me!", jonka ruudunlukija lukee käyttäjälle. Esimerkkikoodilla luodaan kosketettava-alue, jossa lukee teksti ”Press me!”. Koska siitä on tehty saavutettava, elementti ryhmitetään yhdeksi valittavaksi komponentiksi, jossa ruudunlukija lukee ”Tap me! Kaksoisnapauta aktivoidaksesi”. (Meta Platforms Inc., 2023a)

Ohjelmakoodi 1 React Native -saavutettavuusesimerkki. (Meta Platforms Inc., 2023a)

```
<TouchableOpacity
  accessible={true}
  accessibilityLabel="Tap me!"
  onPress={onPress}>
  <View style={styles.button}>
    <Text style={styles.buttonText}>Press me!</Text>
  </View>
</TouchableOpacity>
```

Ominaisuuden accessible={true} avulla saadaan myös yhdistettyä elementtejä yhdeksi solmukohdaksi (node) semanttisessa puurakenteessa. Tällöin ruudunlukija tulkitsee ne yhdeksi elementiksi ja lukee molemmat sisällöt kerralla, eikä käyttäjän tarvitse erikseen siirtyä seuraavaan elementtiin. Jos taas halutaan piilottaa tiettyjä elementtejä

ruudunlukijalta, käytettään Androidille ominaisuutta importantForAccessibility=”no” tai ”no-hide-descendants” ja iOSille accessibilityElementsHidden={true}. (Meta Platforms Inc., 2023a)

AccessibilityLabel ominaisuudella, voidaan siis ruudunlukijalle antaa esimerkiksi kuvaavampaa tietoa elementin toiminnasta. Jos tämä ei riitä, voidaan ominaisuudella accessibilityHint antaa vieläkin tarkentavampaa tietoa. Esimerkiksi käyttöliittymän painikkeessa voi lukea teksti ”takaisin”. Ruudunlukijalle voidaan antaa tarkentava tieto ”siirry takaisin” accessibilityLabel-ominaisuudella ja accessibilityHint-ominaisuudella ”siirtyy takaisin edelliseen näkymään”.(Meta Platforms, Inc., 2023)

AccessibilityRole -ominaisuutta voidaan käyttää kertoaan avustavalle teknologialle kulloinkin fokuksessa olevan komponentin tarkoituksen. Se voi olla esimerkiksi painike (button) tai otsikko (header). AcceccibilityRole-ominaisuutta voidaan käyttää tilanteissa, joissa elementin tarkoitus on jokin muu, kuin sen luontainen rooli antaa ymmärtää, tai jos rooli ei tule selvästi esille ruudunlukijaa käytettäessä. Esimerkiksi kuvaa käytetään linkkinä, tai tekstiä käytetään painikkeena. (Meta Platforms, Inc., 2023). React Nativella voidaan luoda erityyppisiä kosketettavia elementtejä (touchable components), esimerkiksi perinteisten painikkeiden tai linkkien sijaan. Niihin on rakennettu erilaisia toimintoja, jotka aktivoituvat elementtiä painettaessa. Esimerkiksi touchableHighligh -elementin tausta korostuu painettaessa. Joissain tapauksissa voi olla hyvä havaita, mikäli käyttäjä painaa pitkään tiettyä elementtiä. Pitkään kosketukseen voidaan tuoda sovelluksessa lisätoimintoja. Kosketettavissa elementeissä tämä voidaan käsitellä lisäämällä niihin onLongPress -ominaisuus. Kaikki kosketettavat elementit (touchable components) ovat oletusarvoisesti saavutettavia. (Meta Platforms Inc., 2022) Ajantasaiset React Nativen saavutettavuusominaisuudet löytyvät React Native-dokumentaatiosta (Meta Platforms, Inc., 2023).

AccessibilityInfo rajapinnan avulla sovelluksessa voidaan tutkia esimerkiksi, onko laitteen ruudunlukija päällä vai ei sekä ilmoittamaan mikäli ruudunlukijan tila muuttuu. Rajapinnassa voidaan käyttää eri metodeja, joista osa on tarkoitettu iOS-järjestelmälle, mutta muutamia voidaan käyttää sekä Androidille että iOSille. Esimerkiksi metodilla setAccessibilityFocus() voidaan ruudunlukijan käynnistyessä kohdistus ohjata tiettyyn React-komponenttiin.

Metodiin announceForAccessibility() voidaan antaa merkkijono, jonka ruudunlukija lukee äänen. (Meta Platforms, Inc., 2022a).

React Nativen dokumentaatiosta löytyy ohjeita koodin testaukseen, testien kirjoittamiseen sekä suosituksia mm. automaatiotestauksen viitekehyksistä (Meta Platforms Inc., 2023c). React Native-testikirjasto on kevyt ratkaisu, jolla voidaan testata React Native-elementtejä. Sen avulla voidaan esimerkiksi suorittaa toimintoja, kuten painaa painiketta tai hakea vastaavuuksia elementeistä. Voidaan esimerkiksi tehdä kyselyjä, joissa haetaan esimerkiksi tiettyä elementin nimilappua (label, accessibilityLabel tai accessibilityLabelledby). Samoin voidaan tehdä kyselyjä elementin roolin tai tilan suhteen. Rajapinnassa on myös mahdollista hakea elementtejä, jotka eivät ole saavutettavia (isHiddenFromAccessibility -funktio). (Callstack Open Source, n.d.)

3.6 Saavutettavuus Flutter-teknologiassa

Flutter (v. 3.7.1) on Googlen kehittämä avoimen lähdekoodin teknologia, jossa käytetään Dart-ohjelmointikieltä. Sillä voidaan luoda natiiveja sovelluksia niin Android-, kuin iOS -käyttöliittymiinkin, web-sovelluksia sekä Windows, macOS ja Linux -yhteensopivia työpöytäsovelluksia yhdestä koodikannasta. (Flutter, n.d.d)

Flutter-sovellus rakennetaan elementeistä tai luokista, joita Flutterissa kutsutaan termillä ”widget” – suomeksi on joissain yhteyksissä käytetty termiä pienoisohjelma. Flutter-dokumentaatiosta löytyvän kirjaston valmiita widgettejä voi käyttää sellaisenaan, muokata tai rakentaa kokonaan uusia. Saavutettavuus-widgettejä ovat: Semantics, ExcludeSemantics ja MergeSemantics. (Flutter, n.d.a)

Semantics-widget tuo hakukoneille, ruudunlukijoille ja muille avustaville teknologioille esiin kuvauksen elementtien käyttötarkoituksista. Semantics-widgetin kanssa voidaan käyttää noin 50 eri ominaisuutta (property), joiden kautta elementtiä voidaan määritellä. Näitä ominaisuuksia ovat esimerkiksi painike, otsikko, lomakkeen tekstikentän nimilappu tai onko jokin kenttä muokattavissa tai vain luettavaksi tarkoitettu. (Flutter, n.d.a) Monissa widgeteissä on jo sisäänrakennettuna semantiikka, mutta esimerkiksi mukautetut widgetit voidaan rakentaa Semantics-widgetin sisään, jolloin ruudunlukijat tunnistavat

pienoissovelluksen tarkoituksen (Lizama, 2021). Vertaa React Native AcceccibilityRole -ominaisuus.

ExcludeSemantics-widgetin avulla voidaan piilottaa tiettyjä sovelluksen osia avustavilta teknologioilta. MergeSemantics -widgetillä puolestaan voidaan yhdistää pienoisohjelmia yhdeksi solmukohdaksi saavutettavuuspuussa. (Flutter, n.d.a) Vertaa esimerkiksi React Nativen ominaisuuksia importantForAccessibility ja accessible={true}.

Flutterissa on myös sisäänrakennettuna saavutettavuusominaisuuksia, kuten dynaaminen fonttikoko sekä tuki VoiceOver- ja TalkBack-ruudunlukijoille. Sovelluksen virheenkorjaukseen voi käyttää Semantics Debuggeria asettamalla showSemanticsDebugger arvoon tosi. Näin saadaan emulaattoriin näkyville visualisesti se, mitä ruudunlukija kertoo käyttäjälle. Tämä voi auttaa huomaamaan saavutettavuusongelmia sovelluksessa. (Lizama, 2021)

Flutterissa voidaan käyttää testikirjastoa, jossa myös saavutettavuuteen liittyen testiluokkia. Esimerkiksi:

- AccessibilityGuideline API – tarkistaa täyttääkö sovelluksen käyttöliittymä Flutterissa määritellyt saavutettavuussuositukset.
- CustomMinimumContrasGuideline – tarkistaa täyttävätkö erikseen määritetyt elementit värikontrastien vähimmäisvaatimukset.
- MinimunTextContrastGuideline – tarkistaa täyttävätkö tekstiosiot WCAG 2.0 kriteeristön värikontrastien vähimmäisvaatimukset.
- LabelledTapTargetGuideline – tarkistaa onko kaikkiin klikattaviin elementteihin liitetty nimilappu (label).
- MinimumTapTargetGuideline – tarkistaa että kaikki klikattavat elementit ovat tarpeeksi suuria.
- SemanticsController – mahdollistaa semanttisen puurakenteen testaamisen etsimällä rakenteen solmukohdat.

Näillä testiluokilla voidaan tutkia mm. täyttääkö sovellus, tai sen osa, saavutettavuuskriteeristön vaatimukset esimerkiksi tekstikoon, värikontrastin, elementtien koon ja elementtien välisen tilan suhteen. (Flutter, 2023a)

4 Ruudunlukija

Ruudunlukija käyttää Text-To-Speech (TTS) teknologiaa kääntääkseen näytöllä näkyvän tekstin tai tiedon käyttäjälle kuuluvaksi puheeksi. Ruudunlukija voi myös tuottaa pistekirjoitusta ulkoiselle pistekirjoituslaitteelle. Tietokoneella käytettäviä ruudunlukijoita on erilaisia eri käyttöjärjestelmille ja niitä ohjataan tavallisimmin näppäinkomennoilla. Mobiililaitteilla ruudunlukijaa ohjataan pääasiassa kosketuseleillä. (Watson, n.d.)

Jotta ruudunlukija pystyy tulkitsemaan sen mitä näytöllä näkyy, ohjelmointi on täytynyt tehdä käyttäen elementtejä, joissa on semanttiset ominaisuudet. Semanttisten ominaisuuksien avulla käyttöliittymän kuultu versio tarjoaa käyttäjälleen mahdollisimman paljon tuttuja tapoja olla vuorovaikutuksessa eri elementtien kanssa. Kaikilla elementeillä on käytännössä yksi tai useampi seuraavista semanttisista ominaisuuksista: rooli, nimi, arvo (ei kaikilla) tai tila (ei kaikilla). Ruudunlukija tulkitsee elementin nimen eri attribuuttien, kuten ”title”, ”placeholder” ja ”label”, perusteella. (Dodson, 2018)

Ruudunlukija tulkitsee sovelluksen tai sivuston semanttista puurakennetta ja etsii sen solmukohdista (node) tunnistettavia objekteja ja niiden ominaisuuksia. Tällainen objekti voi olla esimerkiksi painike-elementti. Jos painike on tehty käyttäen sille tyypillistä elementtiä (button), sen semanttinen rakenne on valmiiksi sellainen, jonka ruudunlukija tunnistaa ja välittää tiedon oikein käyttäjällä. Painike-elementistä ruudunlukija ilmoittaa painikkeen sisällä olevan tekstin sekä roolin. Jos painike on tehty painike-elementillä, ja sen sisällä on tekstiosio ”Lähetä tilaus”, ruudunlukija lukee ”Lähetä tilaus, painike”. Käyttäjä saa tarpeellisen tiedon painikkeen tarkoituksesta. Koska painikkeen aktivoiminen tapahtuu aina samalla tavalla, käyttäjä voi toimia tilanteen vaatimalla tavalla. Jos taas painikkeena toimiva elementti on vain tyylitelty painikkeen näköiseksi, mutta rakennettu vaikkapa yleistä elementtiä käyttäen, siitä puuttuu painikkeen rooli. Tällaisen elementin kohdalla ruudunlukija ilmoittaa vain osion mahdollisesti sisältämän tekstin, mutta ei roolia. Käyttökokemus ruudunlukijalla on selkeämpi, jos käytetään kullekin toiminnallisuudelle luontaisesti tarkoitettuja elementtejä. (Dodson, 2018). Dodson viittaa HTML-kieleen, jolla rakennetaan verkkosivuja. React Nativessa tai Flutterissa vastaava osa voisi olla jokin yleinen elementti, jolla ei ole omaa nimeä tai roolia.

4.1 TalkBack ja VoiceOver

Mobiililaitteissa on useimmiten valmiiksi asennettuna käyttöjärjestelmän oma ruudunlukija. Applen iOS -laitteissa ruudunlukija on VoiceOver (toistaiseksi uusin versio 16.0) ja Android-laitteissa TalkBack (toistaiseksi uusin versio 13.0). Molemmat voidaan aktivoida päälle tai pois päältä laitteen asetusten tai pikanäppäinkomennon kautta ja niitä voidaan jonkin verran muokata käyttäjän tarpeiden ja toiveiden mukaan. Molemmat käyttöjärjestelmät tarjoavat myös harjoittelutilan, jossa käyttäjä voi kokeilla ruudunlukijan käyttämistä. VoiceOver ja TalkBack ruudunlukijoita käytetään kosketuksella, yhden tai useamman sormen pyyhkäisyeleillä ja yhden tai useamman napautuksen eleillä. Molemmissa on myös käytössä näytöllä olevien kohteiden tutkiminen. Kun käyttäjä laittaa sormen näytölle ja liikuttaa sitä, ruudunlukija kertoo mitä elementtiä käyttäjä kulloinkin koskettaa. (Apple Inc., n.d.b, n.d.a; Google Support, n.d.a)

TalkBack ja VoiceOver käyttävät samoja perustoimintoja, kuten:

- seuraavaan kohteeseen liikkuminen
    - pyyhkäisy oikealle
- edelliseen kohteeseen liikkuminen
    - pyyhkäisy vasemmalle
- kohteen valitseminen, esimerkiksi painikkeen painaminen
    - kaksoisnapautus

  (Apple Inc., n.d.a; Google Support, n.d.b)

VoiceOverissa on roottori -toiminto, jonka avulla ruudunlukijan asetuksia voidaan muokata käytön aikana. Kun roottori on otettu käyttöön laitteen asetuksista, se voidaan aktivoida näytöllä puristamalla peukalo ja etusormi yhteen ja vääntämällä kuvitteellista ruuvia näytöllä. Roottorin avulla voidaan mm. vaihtaa ruudunlukijan kieli, navigointitapa merkkeihin tai otsikoihin tai muokata puhenopeutta. Laitteen asetuksissa voidaan lisätä ja poistaa toimintoja roottoriin. (Apple Inc., n.d.b, n.d.d)

Vastaavasti TalkBackillä voidaan muokata ruudunlukijan navigointiin, puhenopeuteen ja kieleen sekä lukuasetuksiin (ruudunluku kappaleittain, riveittäin, sanoittain tai kirjoitusmerkeittäin). Näihin valintoihin päästään pyyhkäisemällä kolmella sormella oikealle.

Kun haluttu toiminto on etsitty, voidaan päästä ko. toimintoa käyttää pyyhkäisemällä ruudulla joko ylös tai alas. TalkBack-valikossa voidaan valita tai muokata yleisluontoisempia asetuksia, kuten TalkBack-asetukset, äänikomennot ja näyttöhaku tai ruudun pimennys. TalkBack-valikkoon päästään joko napauttamalla kolmella sormella tai pyyhkäisemällä sormella yhtäjaksoisesti ylhäältä alas ja vasemmalle. TalkBack mahdollistaa lähes kaikkien eleiden ja toimintojen muokkaamisen omaan käyttöön sopivaksi. (Google Support, n.d.c)

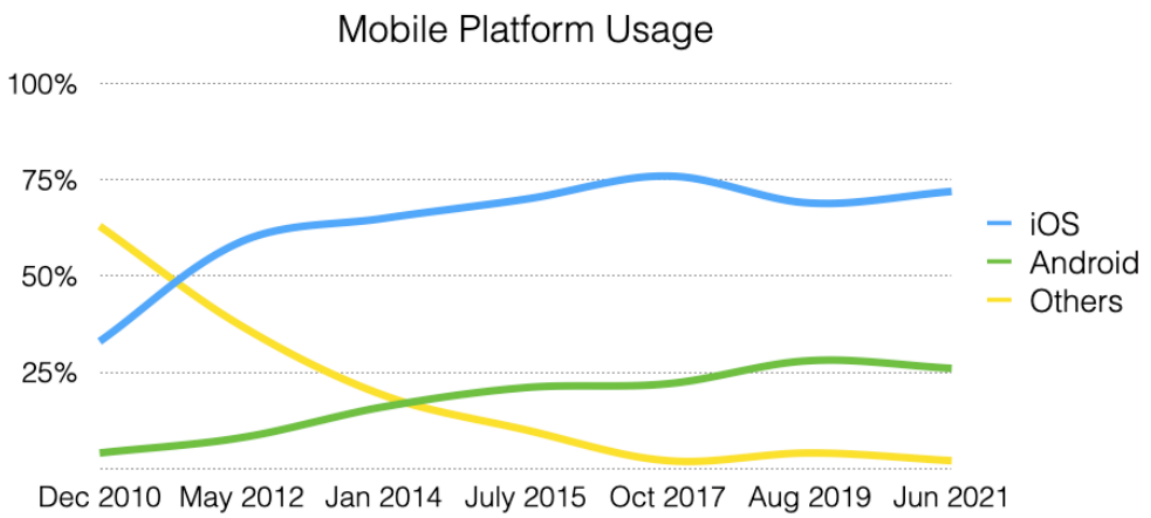
Sekä VoiceOverin että TalkBackin asetuksissa voidaan muokata kuullun tiedon määrää. VoiceOverin ”monisanaisuus”-asetuksesta on mahdollista muokata kertooko ruudunlukija vihjeet, välimerkit, isot kirjaimet ja upotetut linkit. Asetuksissa voi ottaa käyttöön tai pois käytöstä ”Puhu aina ilmoitukset” -asetus, jolla määritetään puhelimeen tulevien ilmoitusten ja viestien ääneen lukeminen. (Apple Inc., n.d.b)

TalkBackissä lähes vastaava asetus on ”Puheen määrä”. Valittavissa on esiasetetut vallinnat matala ja korkea tai muokattu puheen määrän taso. Muokkaamalla voi itse määrittää puhevalintoja, kuten käyttövihjeiden kuuluminen. (Google Support, n.d.d)

4.2 Käyttäjien kokemuksia ruudunlukijan käytöstä

WebAIM (Web Accessibility in Mind) on voittoa tavoittelematon organisaatio, joka on vuodesta 1999 lähtien tuottanut saavutettavuusratkaisuja tavoitteenaan tuoda verkkopalvelut myös niille henkilöille, joilla on jokin vamma tai rajoite. WebAIM mm. järjestää koulutuksia, kehittää saavutettavuuden arviointityökaluja ja tekee tutkimustyötä. WebAIM on tehnyt mm. tutkimusta ruudunlukijan käytöstä vuosien 2009–2021 välillä. Viimeisimmässä kyselyssä oli mukana 1528 henkilöä, joista n. 58 % oli Pohjois-Amerikasta, n. 24 % Euroopasta, ja loput muualta maailmasta. Kyselyyn vastanneista ruudunlukijan käyttäjistä n. 72 % käytti pääasiallisesti iOSia, 26 % Androidia ja loput jotain muuta. Kuva 2 nähdään että tulokset ovat pysyneet samansuuntaisina jo vuodesta 2014 saakka. Tutkimustuloksissa on raportoitu, että eurooppalaisista vastaajista 71 % ilmoitti käyttävänsä iOS-laitteita. (WebAIM, 2021)

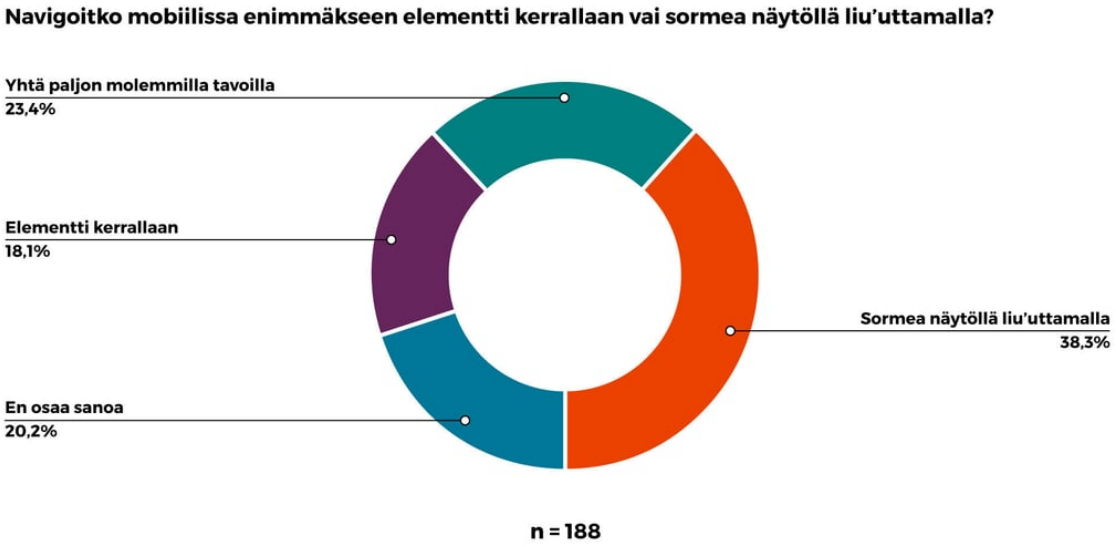
Kuva 2. Mobiilikäyttöjärjestelmien suosio ruudunlukijan käyttäjien keskuudessa (WebAIM, 2021).



Vuonna 2021 toteutettiin saavutettavuuskysely, jossa ensimmäistä kertaa kartoitettiin näkövammaisten henkilöiden ruudunlukijan käyttötottumuksia Suomessa. Kyselyn toteutti ohjelmistotalo Eficode, Näkövammaisten liitto sekä Annanpura Oy ja siihen vastasi 244 henkilöä. Vastaajista melkein puolet (48 %) oli ikäkategoriassa yli 60-vuotiaita, 41–60 vuotiaita oli 36,9% vastaajista, 21-40 vuotiaita 13,9% ja alle 20-vuotiaita oli 1,2 % vastaajista. (Kallionpää & Kiiskilä, 2021; Näkövammaisten liitto ry, 2021)

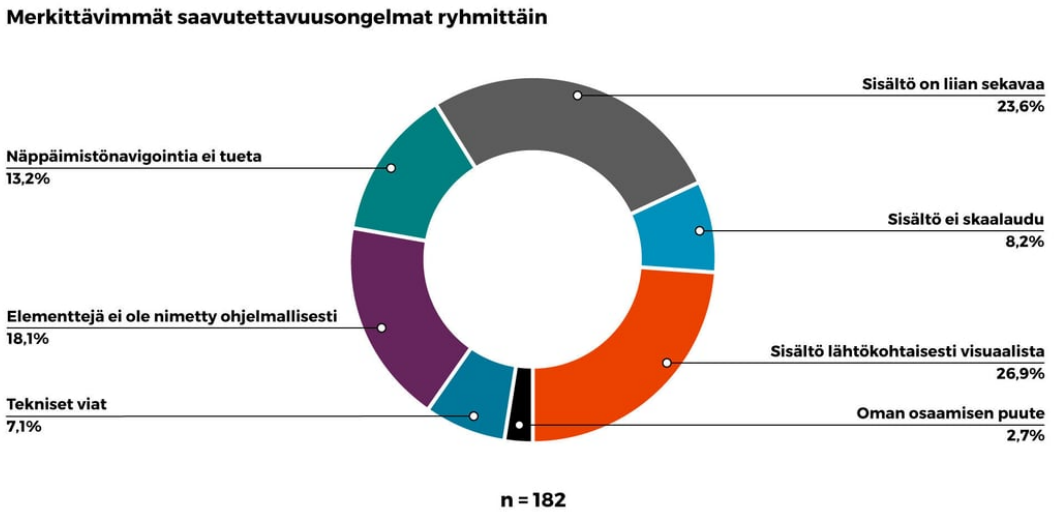
Vastausten perusteella suosituimmat ruudunlukijat mobiililaitteella oli VoiceOver (78 %), TalkBack (11,3 %) ja Nuance Talks (4,2 %). Kyselyssä selvitettiin mm. kuinka ruudunlukijan käyttäjät navigoivat sisältöä. Tietokoneella navigoitaessa n. 47 % käyttäjistä käytti navigoimiseen otsikoita. Kuva 3 esittää erilaisia tapoja navigoida mobiililaitteilla. Navigointi tapahtui pääasiassa sormea näytöllä liu’uttamalla (vastaajista 38 %), elementti kerrallaan navigoi n. 18 % vastaajista, yhtä paljon molemmilla tavoilla 23 % ja en osaa sanoa 20 % vastaajista. (Kallionpää & Kiiskilä, 2021, s. 24)

Kuva 3. Ruudunlukijakysely: navigointi mobiililaitteella (Kallionpää & Kiiskilä, 2021, s. 24).



Kyselyssä oli avoimia haastattelukysymyksiä, joiden pohjalta on ryhmitelty käyttäjien kohtaamat merkittävimmät saavutettavuushaasteet. Kuva 4 esittää seuraavat vastausryhmät ja niiden prosenttiosuudet. Noin puolet raportoiduista saavutettavuusongelmista liittyy siihen, että sisältö on lähtökohtaisesti visuaalista (26,9 % vastauksista) ja sisältö on liian sekavaa (23,6 % vastauksista). Muita isoja ongelmia oli, että elementtejä ei ole nimetty ohjelmallisesti (18,1 %) ja näppäimistönavigointia ei tueta (13,2 %). Lisäksi ongelmia on aiheuttanut sisällön skaalautumattomuus (8,2 %), tekniset viat (7,1 %) ja oman osaamisen puute (2,7 %). (Kallionpää & Kiiskilä, 2021, ss. 36-37.)

Kuva 4. Verkkopalveluiden merkittävimmät saavutettavuusongelmat (Kallionpää & Kiiskilä, 2021, s. 37).



4.3 Ruudunlukijan huomioiminen sovelluskehityksessä

Ruudunlukijaa käytettäessä sovelluksen sisällön ymmärtäminen tapahtuu vaiheittain, yksi kohde kerrallaan. Näkevät käyttäjät voivat silmäillä ruudun sisällön (otsakkeet, sisällön jaottelu, kuvat, taiteellinen tyyli, jne.) nopeasti läpi ja saada kokonaisymmärryksen sisällöstä. Sovelluksen suunnittelussa on hyvä osata huomioida se, että ruudunlukijan käyttäjät etenevät sovelluksessa kohta kerrallaan ja muodostavat mielikuvan sovelluksesta vain kuullun perusteella. Ruudunlukijan käyttäjille sisällön ymmärtämiseen kuluu enemmän aikaa ja he ovat riippuvaisia siitä, että sisältö on jaoteltu tarkoituksenmukaisesti ja ohjelmallisesti oikein. (Sapega, 2021)

Jotta koko sisältöä ei tarvitse käydä läpi, ruudunlukijoilla voi navigoida sisältöä. Navigoinnin kannalta on tärkeää, että sisältö on ohjelmoitu oikein. Mobiilisivustolla tai sovelluksessa on esimerkiksi linkki, jolla ohitetaan sivuilla toistuvat rakenteet, otsikot on merkitty ohjelmallisesti, tekstikappaleet on jaoteltu, lomake-elementit ja linkit on merkitty ohjelmallisesti ja linkeissä on kuvaavat ja uniikit nimet. Kun elementit on merkitty oikein ohjelmallisesti, ruudunlukija osaa kertoa mikä elementti on kyseessä tai elementin perusteella voi navigoida sisältöä. Kuten WCAG-ohjeissakin määritellään,

sovellussuunnittelussa olisi otettava huomioon ainakin tekstin lukemisen helppous, sovelluksen yksinkertainen rakenne ja helppo navigoitavuus.

Sisältö jaotellaan loogisiin osiin ja merkittään oikeilla ja loogisilla otsikkotasolla (h1, h2, jne.), lisäksi esimerkiksi lomakekentillä täytyy olla kuvailevat nimilaput (engl. label), joista esimerkiksi ruudunlukijan käyttäjä saa tarvitsemansa tiedon kunkin kentän tarkoituksesta. Nämä vaatimukset liittyvät WCAG-kriteeriin 2.4.6 Otsikot ja nimilaput. (W3C, n.d.b)

4.4 Ruudunlukijatestaaminen mobiilisovelluksissa

Ruudunlukijalla testaaminen vaatii kulloisenkin ruudunlukijan käytön opettelua, joka vaatii oman aikansa. Etenkin tietokoneisiin asennettavien ruudunlukijoiden, kuten NVDA, ruudunlukijoiden käyttöön liittyy paljon näppäinkomentoja, jotka toisaalta sujuvoittavat ja nopeuttavat käyttöä, mutta vaativat myös käyttörutiinia. (Lamminen, 2020)

Mobiililaitteiden ruudunlukijoiden käyttö perustuu erilaisiin pyyhkäisyeleisiin sekä ruudun napautuksiin yhden tai useamman kerran. Kuten jo edellä on mainittu TalkBackissä ja VoiceOverissa on jonkin verran eroja, vaikka samat perustoiminnallisudet löytyvät molemmista.

Ruudunlukijan käyttäjän on kuuntelemalla saatava selkeä kuva sovelluksesta, sen tarkoituksesta ja mahdollisesti sovelluksen avulla suoritettavista tehtävistä, kuten esimerkiksi ostoksen tekemisestä verkkokaupassa. Jos sovellusta testataan ulkoisella näppäimistöllä, voidaan melko luotettavasti todeta, onko eri elementtien kohdistusjärjestys haluttu ja saako saavatko kohdistuksen kaikki ne elementit, jotka ovat käyttäjän kannalta tarkoituksen mukaisia. Esimerkiksi elementit, joilla on vain koristeellinen tarkoitus, eivät saa kohdistusta avustavia teknologioita käytettäessä (WCAG 2.1. kriteeri 1.1.1 ja Näkövammaisten liitto). Ruudunlukija saattaa kuitenkin joissain kohdissa saada eri kohdistusjärjestyksen kuin näppäimistöllä selattaessa, joten kohdistusjärjestys on syytä tarkistaa ruudunlukijalla (Pesonen, 2021).

Ruudunlukijalla testattaessa olisi tärkeää yrittää käyttää sovellusta, kuten oikea käyttäjä ja tarkistaa löytyykö sovelluksen pääkohdat helposti ja onnistuuko navigointi otsikoittain ja

oikeassa järjestyksessä. Laitteen lukuasetuksia voidaan sekä TalkBack- että VoiceOver -ruudunlukijoissa muuttaa ainakin niin, että pyyhkäisyeleellä siirryttäessä seuraava elementti on sana, rivi, otsikko, ohjain tai linkki. Näiden ohjaineleiden toimivuus on syytä tarkistaa omassa sovelluksessa. Lukuasetuksista valitaan esimerkiksi ”linkit” ja tarkistetaan, siirtyykö kohdistus seuraavaan tai edelliseen linkkiin pyyhkäisyeleellä ja löytyykö kaikki linkit tällä tavoin. Linkkitekstien täytyy olla selkeitä ruudunlukijalla kuunneltaessa ja linkin tulee aueta kaksoisnapautuksella. Ruudunlukijan tulee ilmoittaa, mikäli linkki aukeaa uuteen välilehteen tai ikkunaan tai jos esimerkiksi kuva toimii linkkinä. Selkeä ja hyvä kielenkäyttö korostuu, kun käyttäjä saa vain auditiivista informaatiota. Samaan tyylin voidaan testata kaikki muutkin elementit sivulla. (Näkövammaisten liitto ry, 2019c, 2019b)

Ruudunlukijan tulee ilmoittaa selvästi mikä käyttöliittymäelementti on kyseessä ja miten eri lomakekenttiä tulee käyttää. Jos on kyseessä tekstinsyöttökenttä, kirjoitustilaan on päästävä kaksoisnapautuksella. Myös toimintapainikeet, kuten valinruutu (checkbox), valintakytkin (toggle), valintanappi (radiobutton) tai painike (button) aktivoituvat mobiililaitteilla kaksoisnapautuksella. Kaikkien eri elementtien toimivuus tarkistetaan ja saako ruudunlukijalla ylipäätään saman käsityksen sovelluksen toiminnoista kuin näkevä käyttäjä saisi. Lomakkeen tulee ohjata käyttäjää täyttämään lomake oikein, mutta virheen sattuessa käyttäjän on saatava siitä palaute. Lisäksi virheen korjaamisen tulee sujua mahdollisimman helposti. (Näkövammaisten liitto ry, 2019b; Suomi.fi Design System, n.d.)

Ruudunlukijatestaamisessa kiinnitetään huomiota siihen, saako käyttäjä oikeanlaista palautetta eri elementtien nimistä ja rooleista. Esimerkkinä painike, joka lähettää tilauksen verkkokaupassa. Tämän tyyppisen elementin nimi voisi olla ”lähetä tilaus” ja rooli on painike. Ruudunlukijalla kuullaan elementin nimi ja rooli, siis ”lähetä tilaus, painike”. Elementillä voi olla myös tila, kuten esimerkiksi valintaruudulla ”valittu” / ”ei valittu”. Erityyppisten elementtien kohdalla kuuluu saada erilaista palautetta ruudunlukijalta. Eri elementtien testaamisen yhteydessä arvioidaan, saako käyttäjä oikean kuvan siitä mitä elementin aktivoimisen yhteydessä tapahtuu – onko jotain valittu tai ei valittu, ja ilmoitetaanko käyttäjälle tilamuutoksista. (Pesonen, 2021)

5 React Native- ja Flutter-sovellusten vertailu

Käytännön työnä on muokattu kevyet sovellukset käyttäen React Native- ja Flutter-teknologioita. React Nativen mallikoodit on kerätty virallisen dokumentaation koodiesimerkeistä ja koostettu yhden sivun sovellukseksi (Meta Platforms Inc., 2023b). Saavutettavuus toimintoja on lisätty ja joitain ominaisuuksia on poistettu käytöstä kommentoimalla ne ulos, jos ne ovat vaikuttaneet negatiivisesti elementin saavutettavuuteen. Lisäksi päivämääräkentän toimintaa testattiin Expo Go-sovelluksella, erillistä React Native-kirjastoa käytettävällä päivämäärän valitsimella ”react-native-modal-datetime-picker example”. Tämä valittiin siitä syystä, että kirjastoa päivitetään aktiivisesti ja sillä oli kattava dokumentaatio (Mazzarolo, 2016/2023). Esimerkki tallennettiin Expo tilille ja se on käytettävissä Expo Go-sovelluksen kautta.

Flutter Form App sovelluksen mallikoodi löytyy Flutter Samples verkkosivulta, jossa kerrotaan noudatetun Flutterin parhaita käytäntöjä mallisovellusten luomisessa (Flutter, n.d.b). Virallisia mallikoodeja on käytetty mm. siitä syystä, että on haluttu minimoida mahdollinen elementtien väärinkäyttö ja sitä kautta tulosten vääristyminen.

React Native- ja Flutter-sovellukset eivät ole identtiset, mutta molemmissa on samantyylisiä elementtejä, joita käytetään erityisesti lomakkeissa.

React Native-sovelluksia voi rakentaa kahdella eri tavalla, joko React Native-projekteina tai Expo Go -projekteina. Expo on joukko työkaluja ja palveluja, joiden avulla React Native-sovelluksen rakentamisen voi aloittaa nopeasti. Expo Go on ladattava sovellus, jonka kautta voidaan mm. testata Expo-projektina rakennettuja sovelluksia sekä Android- että iOS-mobiililaitteilla. (Meta Platforms Inc., 2023d)

Molemmissa teknologioissa on käytössä ns. ”hot reload” toiminnallisuus, jonka avulla muutokset koodiin voidaan nähdä heti esimerkiksi emulaattorissa tai tietokoneeseen USB-kaapelilla yhdistetyssä mobiililaiteessa. (Bigio, 2016; Flutter, n.d.c)

React Nativella kehitysmuodoksi valittiin Expo-projekti testaamisen helpottamiseksi. Ladattavan Expo Go -sovelluksen avulla testaaminen onnistui helposti myös iOS-laitteella, vaikka oma varsinainen kehitysympäristö oli Windows-käyttöjärjestelmässä. Expon

käyttäminen ei välttämättä ole suositeltavaa vaativamman sovelluksen kehittämisessä, sillä kaikki React Nativen ohjelmistorajapinnat eivät ole käytettävissä Expolla (Meta Platforms Inc., 2023d). Yksinkertaiseen käyttöliittymäkomponenttien testaamiseen se oli kuitenkin hyvä ja nopea ratkaisu.

Flutter -kehitysympäristön pystyttämiseen oli yksityiskohtaiset ohjeet käytettävissä (Flutter, n.d.c). Flutterilla ei ole Expoa vastaavaa sovellusta helpottamaan testausta, joten tässä tapauksessa kehittäminen tehtiin Android-laitetta käyttäen. Kehittämisen aikana Flutter-sovellusta ajettiin oikealla Android-puhelimella virheenkorjaustilassa (debug mode). Näin oli mahdollista koko ajan testata muutoksia ruudunlukijaa käyttäen. Lopuksi Flutter Form App sovelluksesta julkaistiin Ahola Digitalin sisäinen sovellus, jota voitiin käyttää ennakkoon määritetyillä iOS-laitteilla.

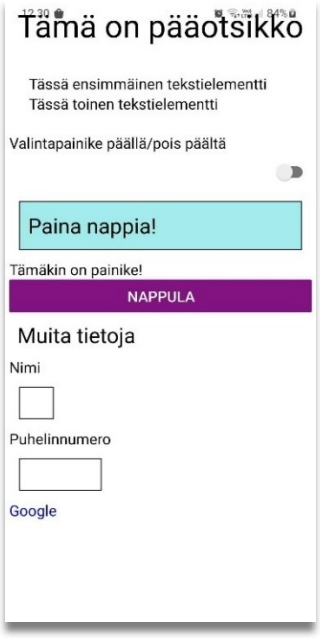
Käytetyt järjestelmät, ohjelmat ja laitteet:

- Käyttöjärjestelmä: Windows 10 Pro
- Tekstieditori: Visual Studio Code
- Android-laite: Samsung Galaxy A53, Android-versio 13, TalkBack-versio: 13.5
- IOS-laite: iPhone7, IOS-versio 15.7.3, jonka mukana päivittynyt VoiceOver.

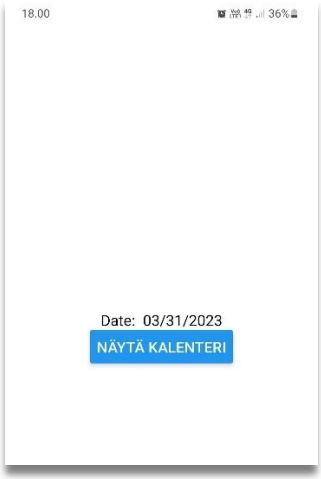
5.1 React Native-testisovellus

React Native-testaus tehtiin kahdella eri sovelluksella, joista Kuva 5 näkyvällä sovelluksella tehtiin suurin osa testauksista. Tämä varsinainen testisovellus luotin Expo-projektina. Uuden projektin luominen tapahtui komennolla ”npx create-expo-app <name of project>”. Komento luo valmiin sovelluspohjan. Ohjelman käynnistäminen tapahtuu siirtymällä ensin projektikansioon, jonka jälkeen käynnistetään ohjelma komennolla ”npx expo start”. Testilaitteeseen asennetussa Expo Go-sovelluksessa avataan luotu projekti esimerkiksi skannaamalla tekstieditorin terminaalissa näkyvä QR-koodi. Ohjelman muokkaus tapahtuu tekstieditorin App.js-tiedostossa. Tallentamisen jälkeen muutokset näkyvät suoraan testilaitteella. (Meta Platforms Inc., 2023d) Oikealla laitteella testaaminen mahdollistaa laitteen oman ruudunlukijan käytön, joka oli tässä testissä välttämätöntä.

Testisovellukseen muokattiin käyttöliittymäelementtejä, joiden toimintaa ruudunlukijaa käyttämällä testattiin sekä Android- että iOS-laitteella. Testaaminen ruudunlukijalla videoitiin. Videoiden perusteella, käyttöliittymäelementtien toiminta ruudunlukijalla purettiin ja tulokset esitetään Kuva 5. Käyttöliittymä React Nativella.



Kuva 6. Päivämäärän esitystapa testisovelluksessa



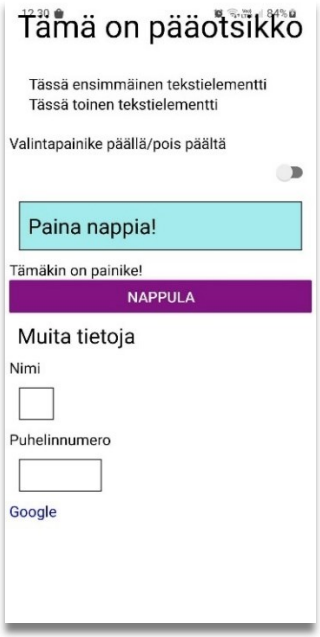
Taulukko 1:ssä. Siinä näkyvät testatut elementit, niiden odotettu toiminta ruudunlukijalla ja varsinainen toiminta sekä Android- että iOS-laitteella. Oletettu toiminta perustuu React

Native-dokumentaatiossa annettuihin kuvauksiin. React Nativella tehdyn testisovelluksen käyttöliittymä Kuva 5:ssä. Päivämäärää testattiin erillisellä sovelluksella ja Kuva 6 näkyy kuvakaappaus päivämäärän esitystavasta, joka on muodossa ”kuukausi/päivä, vuosi”.

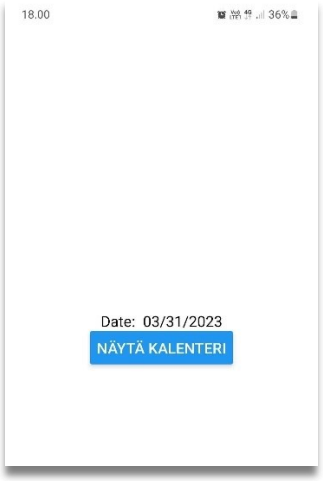
Testatut käyttöliittymäelementit:

1. Otsikko (header)
2. Tekstielementti (text)
3. Valintapainike (switch)
4. Painike, joka on tehty yleisestä <View> -elementistä
5. Painike, joka on tehty <Pressable> -elementistä
6. Painike, joka on tehty <Button> -elementistä
7. Huomiokenttä (alert), joka ilmoittaa, että painiketta on painettu
8. Tekstimuokkauskenttä
9. Tekstimuokkauskenttä, johon odotetaan numeerista syötettä
10. Linkki sovelluksen ulkopuoliseen sivustoon
11. Päivämäärä (testattu erillisellä sovelluksella)

Kuva 5. Käyttöliittymä React Nativella.



Kuva 6. Päivämäärän esitystapa testisovelluksessa



Taulukko 1. React Native-sovelluksen ruudunlukijatestaus.

Elementti / Rooli	**Saavutettavuus-ominaisuudet**	**Odotettu toiminta ruudunlukijalla**	**Toiminta Android-laitteessa**	**Toiminta iOS-laitteessa**
1. <Text> elementti, rooli otsikko	accessibilityRole =”header”	Ruudunlukija ilmoittaa elementin tekstisisällön sekä roolin.	Toimi odotetusti.	Toimi odotetusti.

2. <View> elementti, sisällä 2 x <Text> elementti	View: accessible= {true}	Ruudunlukija yhdistää kaksi elementtiä yhdeksi ja lukee molemmat tekstielementit peräkkäin.	Puutteita. Tekstielementit käyttäytyivät itsenäisinä tekstikappaleina.	Toimi odotetusti. Ruudunlukija luki molemmat tekstikappaleet peräkkäin yhtenä elementtinä.
3. <Switch> elementti, rooli valinta-painike	accessibility Label = "Tee valinta", accessibility Hint= "Laita toiminto päälle tai pois"	Ruudunlukija kertoo elementin tilan (päällä tai pois päältä), saavutettavuus-ominaisuudet (ks. edellinen solu) sekä elementin roolin.	Toimi odotetusti.	Toimi odotetusti. ”Tee valinta, vaihtopainike, pois, Laita toiminto päälle tai pois. Vaihda asetusta kaksoisnapaut-tamalla”
4. <View> elementti / painike-rooli. <View> elementin sisällä <Text>-elementti ”Paina nappia”	View: accessible= {true}, accessibilityRole = ”button”	View- ja Text-elementit yhdistetty ominaisuudella accessible= {true}, joten ruudunlukija huomaa vain yhden elementin, lukee teksti-elementin sisällön ja ilmoittaa painike-roolin.	Puutteita. Elementin rooli ei välittynyt ruudunlukijan tiedoista, vaikka elementti toimi painikkeen tavoin kaksois-napautuksella.	Toimi odotetusti. ”Paina nappia, painike” (VoiceOver ei kerro, että painikkeen aktivointi tapahtuu kaksois-napauttamalla)
5. <View> elementin sisällä sekä <Pressable> elementti, painike-rooli, että <Text> elementti	View: accessible={true }, Pressable: accessibilityLabe l= ”Tämä on pressable elementti….” accessibilityRole ”button” Text: piilotettu ruudunlukijalta	Ruudunlukija huomaa vain yhden elementin ja kertoo accessibilityLabel -sisällön, elementin roolin sekä painikkeen käytön kaksoisnapautuksel la.	Toimi odotetusti. ”Tämä on pressable elementti. Olen painettava tekstielementti. Painike. Kaksoisnapauta aktivoidaksesi”	Toimi odotetusti. Ruudunlukija kertoo accessibilityLabelin sisällön, mutta ei roolia.
6. <Button> elementti / painike-rooli	accessibilityLabe l= ”Vahvista valinta”	Ruudunlukija kertoo painikkeen accessibilityLabel -sisällön, elementin roolin ja painikkeen käytön kaksois-napautuksella.	Toimi lähes odotetusti. Ruudunlukija lukee: ”Vahvista valinta, painike, kaksoisnapauta aktivoidaksesi.”	Toimi odotetusti. ”Vahvista valinta, painike”

			Huom. Jos painiketta ei paineta, vaan siirrytään ruudunlukijalla eteenpäin painikkeen teksti (title) saa fokuksen. Ei odotettu toiminto.	
7. Alert-huomiokenttä	ei erikseen asetettuja ominaisuuksia	Ruudunlukija siirtyy huomiokenttään automaattisesti – olisi hyvä jos sisältö tulisi luetuksi automaattisesti ilman että ruudunlukijan fokusta täytyy siirtää.	Toimi odotetusti. Ruudunlukija ilmoittaa, että käyttäjä on huomiokentässä ja kertoo kentän tekstin. Pyyhkäisyeleellä siirrytään OK-painikkeeseen, jolla huomautus kuitataan.	Toimi lähes odotetusti. Ruudunlukijan fokus siirtyi Alert-huomiokenttään, mutta ruudunlukijalla piti siirtyä eteenpäin, jotta kentän varsinainen teksti kuului.
8. Muokattava tekstikenttä <TextInput>	accessibilityLabel= ”Kirjoita nimesi” acessibilityHint = ”Kirjoita koko nimesi”	Ruudunlukija kertoo elementin roolin (muokkauskenttä), accessibilityLabel, accessibilityHint ja käytettävissä olevat toiminnot (kentän aktivointi kaksois-napautuksella). Kertoo, kun näppäimistö on käytettävissä.	Toimi odotetusti. Ruudunlukija lukee: ”Kirjoita nimesi, kirjoita koko nimesi, muokkauskenttä. Kaksoisnapauta muokataksesi tekstiä” Kaksois-napautuksen jälkeen näppäimistö tulee näkyviin ja Talk Back ilmoittaa ”qwerty näkyy”. Tekstin lisäyksen jälkeen ilmoittaa ”näppäimistö piilotettu”.	Puutteita. Ruudunlukija kertoo muokkauskenttään yhdistetyn tekstin ”Nimi” sekä muokkauskentän accessibilityLabelin sisällön ”Kirjoita nimesi”. Kentän roolia ei kerrota. VoiceOver ei ilmoita, että näppäimistö on käytettävissä.

9. Muokattava tekstikenttä, jossa näytetään numeronäppäimistö <TextInput>	accessibilityLabel= ”Kirjoita puhelin-numerosi” acessibilityHint = ”Maakoodia ei tarvita”	Ruudunlukija kertoo elementin roolin (muokkauskenttä), accessibilityLabel, accessibilityHint ja käytettävissä olevat toiminnot (kaksoisnapautus). Kertoo, kun näppäimistö on käytettävissä.	Toimi odotetusti.	Käytön estävä puutte. Näppäimistöä ei ilmoiteta. Numeronäppäimistössä ei ole painiketta jolla muokkaustilasta pääsee eteenpäin. VoiceOver on suljettava, jotta muokkauskenttä voidaan sulkea.
10. Tekstilinkki	accessibilityLabel= ”Google etusivu” accessibilityHint = ”Avaa linkin uuteen ikkunaan” accessibilityRole = ”Link”	Ruudunlukija ilmoittaa kaikki saavutettavuus-ominaisuudet: accessibilityLabel, accessibilityHint, accssibilityRole	Toimi odotetusti. ”Google etusivu, avaa linkin uuteen ikkunaan, link”.	Toimi odotetusti.
11. Päivämäärä	datetimepicker	Ruudunlukija kertoo päivämäärän paikallisten aika- ja päivämäärä asetusten mukaan	Toimi odotetusti.	Puutteita. VoiceOver lukee ruudulla näkyvät kenoviivat ääneen. Koska päivämäärä on ruudulla muodossa: mm/dd/yyyy käyttäjä ei voi olla varma missä muodossa ruudunlukija antaa päiväyksen
		11 elementtiä testattu	**9 x toimi odotetusti** **2 x puutteita**	**8x toimi odotetusti** **2x puutteellista** **1x käytön estävä puute**

Huomioitavaa on, että testisovellukseen ei tehty erikseen käännöstiedostoa, joten joitain sanoja ruudunlukijalla kuullaan englanniksi.

Suurin puute Androidilla oli painikkeessa, joka oli rakennettu View-elementistä. Sen sijaan muut painikkeiksi tehdyt elementit toimivat odotetulla tavalla. Androidilla tekstielementit, joiden odotettiin olevan yhdistetty yhdeksi elementiksi, toimivat erillisinä. Nämä puutteet eivät vaikuttaneet kriittisiltä.

iOS-laitteella puutteita oli muokattavissa tekstikentissä. Numeronäppäimistön puute oli käytön estävä. Ruudunlukijaa käyttäen kentästä ei päässyt eteenpäin. Androidin numeronäppäimistössä oli ”Valmis” painike, jolla näppäimistön sai suljettua ja lomakkeen käyttöä voitiin jatkaa. Vastaavaa painiketta ei ollut iOS-numeronäppäimistössä. Lisäksi oli puute tekstinmuokkauskentässä, jossa ruudunlukija ei ilmoittanut elementin roolia. Käyttäjän tieto tämän kentän osalta rajoittui annettuun elementin nimilapun (accessibilityLabel) tietoon. Puute voi hankaloittaa käyttöä ruudunlukijalla etenkin, jos elementin nimilapussa ei anneta riittävästi tietoa. Alert-huomiokentässä oli pieni puute. Ruudunlukijan tulisi lukea huomioteksti, ilman että ruudunlukijan fokusta täytyy siirtää erikseen. Androidilla tämä toimi, mutta iOS-laitteella käyttäjän piti osata siirtää fokus huomiotekstiin.

Yksittäisten elementtien testauksen ulkopuolelta voidaan todeta myös, että elementistä toiseen siirtyminen ei ollut aina sujuvaa. Mallikoodiin ei sen suhteen tehty muutoksia testiä varten.

5.2 Flutter-testisovellus

Flutter-sovelluksen luominen voidaan komentoterminaalissa aloittaa komennolla ”flutter create <name_of app>. Komento luo valmiin sovelluspohjan. Ohjelman käynnistäminen tapahtuu siirtymällä ensin projektikansioon, jonka jälkeen käynnistetään ohjelma komennolla ”flutter run”. Kun testilaitteessa on oltava otettu käyttöön ns. kehittäjätila (developer mode) se yhdistetään USB-kaapelilla tietokoneeseen ja ajetaan komento ”flutter run”. Tämä käynnistää ohjelman ajaa sitä virheenkorjaustilassa testilaiteessa.

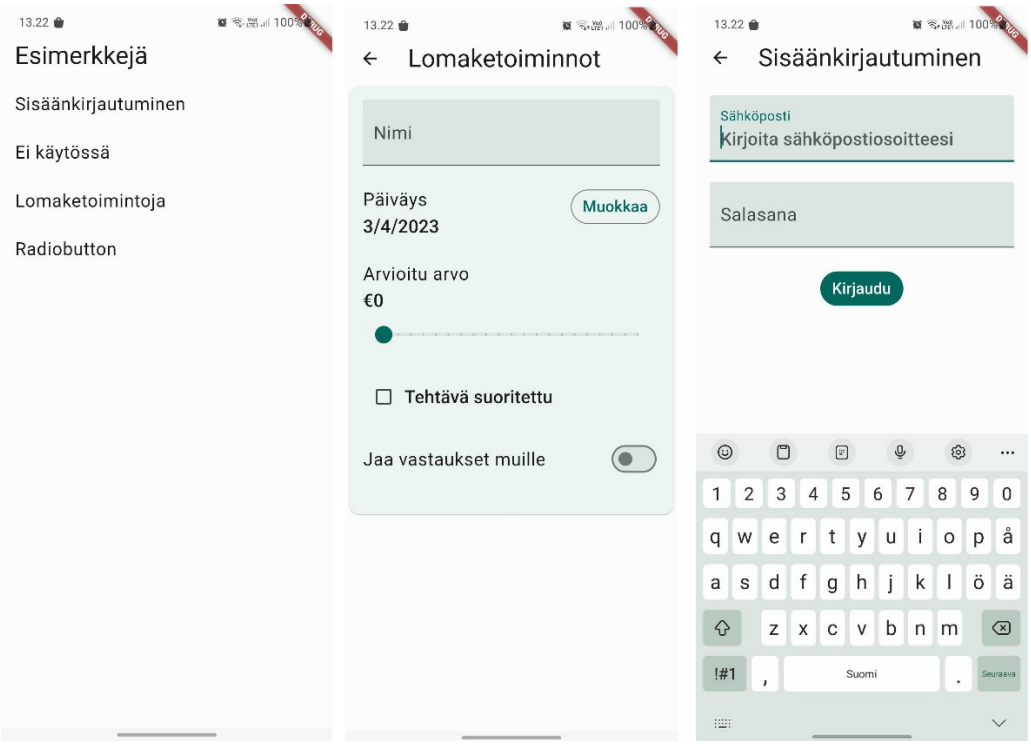
Testisovellukseen käyttöliittymäelementtejä muokattiin saavutettaviksi. Niihin lisättiin esimerkiksi Flutterin ”Semantics” ja ”MergeSementics” widgettejä, sekä vain ruudunlukijalla kuultavia ominaisuuksia, kuten ”semanticsLabel” ja ”hintText”. Vastaavat ominaisuudet

React Nativessa ovat ”accessibilityLabel” ja ”accessibilityHint”. Testaaminen sekä Android- että iOS-laitteilla videoitiin. Videoiden perusteella, käyttöliittymäelementtien toiminta ruudunlukijalla analysointiin ja tulokset esitetään Taulukko 2 Flutter-sovelluksen ruudunlukijatestaus. Siinä näkyvät testatut elementit, niiden odotettu toiminta ruudunlukijalla ja varsinainen toiminta sekä Android- että iOS-laitteella. Oletettu toiminta perustuu ruudunlukijan käyttökokemukseen sekä React Native- ja Flutter-dokumentaation kuvauksiin. Flutter-testisovelluksen käyttöliittymä ilmenee Kuva 7.

Testatut käyttöliittymäelementit:

1. Otsikko (title)
2. Tekstimuokkauskenttä, Nimi
3. Päivämäärä
4. Painike (outlined button), Muokkaa
5. Liukukytkin (slider)
6. Valintaruutu (checkbox)
7. Valintapainike (switch)
8. Linkki sovelluksen sisäinen (back button)
9. Huomiokenttä, joka tulee esiin ”Kirjaudu” painikkeen painamisen jälkeen.

Kuva 7. Käyttöliittymä Flutter FormApp



Taulukko 2 Flutter-sovelluksen ruudunlukijatestaus

Elementti / Rooli	**Saavutettavuus -ominaisuudet**	**Odotettu toiminta ruudunlukijalla**	**Toiminta Android-laitteessa**	**Toiminta iOS-laitteessa**
1. Title, sovelluksen sivun otsikko	Ei lisätty	Ruudunlukija ilmoittaa tekstin ja roolin, esim. ”Lomaketoiminnot, otsikko”	Toimi odotetusti	Toimi odotetusti
2. TextFormField Tekstinmuok-kauskenttä	hintText:’Kirjoita nimesi’ labelText: ’Nimi’	Ruudunlukija kertoo elementin roolin (muokkauskenttä), labelText ja hintText käytettävissä olevat toiminnot (kaksoisnapautus). Aktivoinnin jälkeen lisää tietoa, myös hintText sekä tulee ilmi että merkkitila eli näppäimistö on käytössä. Kertoo, kun näppäimistö on käytettävissä.	Toimi odotetusti.	Toimi odotetusti.
3. Päivämäärä	ei muutoksia	Ruudunlukija kertoo päivämäärän tyypillisessä muodossa yhtenä tietona.	Toimi odotetusti.	Puutteita. Label ja päivämäärä luettiin yhtenä tietona, mutta päivämäärän muoto (USA) ei vastaa paikallista muotoa ja värinymmärryksen mahdollisuus on olemassa.
4. Painike	semanticsLabel: ’Vaihda päivämäärä’	Ruudunlukija kertoo semanticsLabel sisällön + elementin roolin	Toimi odotetusti.	Toimi odotetusti.
5. Liukukytkin, slider	ei muutoksia	Ruudunlukija kertoo liukukytkimen label-tekstin ja arvon sekä elementin roolin ja käyttövihjeet.	Puutteita. Ruudunlukija kertoi label-tekstin, elementin roolin sekä	Puutteita. Ruudunlukija kertoi label-tekstin, elementin roolin sekä

		Muokattaessa kertoo uuden arvon.	käyttövihjeet. Arvoa muokattaessa uusi arvo ei välittynyt oikein ruudunlukijalla.	käyttövihjeet. Arvoa muokattaessa uusi arvo ei välittynyt oikein ruudunlukijalla.
6. Valintaruutu (checkbox)	Merge Semantics, semanticsLabel: ’Merkitse tehtävä suoritetuksi’	Ruudunlukija kertoo elementin tilan (päällä tai pois päältä), labelin ja semanticsLabelin sisällön, elementin tarkoituksen. Käyttö kaksoisnapautuksella. Tilaa muutettaessa tieto välittyy ruudunlukijalla.	Toimi odotetusti	Toimi odotetusti.
7. Valintapainike (switch)	Merge Semantics, semanticsLabel: ’Anna toisten käyttäjien nähdä vastaukset’	Ruudunlukija kertoo elementin tilan (päällä tai pois päältä), labelin ja semanticsLabelin sisällön, elementin tarkoituksen. Käyttö kaksoisnapautuksella. Tilaa muutettaessa tieto välittyy ruudunlukijalla.	Toimi odotetusti	Toimi odotetusti
8. Linkki, sovelluksen sisäinen	Takaisin-painike, sisäänrakennettu sovelluksen AppBar-otsikkopalkkiin	Ruudunlukija kertoo elementin tarkoituksen ja aktivoimisen jälkeen missä näkymässä käyttäjä on.	Toimi odotetusti.	Toimi odotetusti.
9. Huomiokenttä	Ei muutoksia	Ruudunlukija siirtyy huomiokenttään automaattisesti – olisi hyvä, jos sisältö tulisi luetuksi automaattisesti ilman, että ruudunlukijan fokusta täytyy siirtää.	Puutteita. Ruudunlukijan fokus ei siirtynyt huomiokentän tekstiin ilman, että käyttäjän toimia.	Puutteita. Ruudunlukijan fokus ei siirtynyt huomiokentän tekstiin ilman että käyttäjä siirtyy pyyhkäisemällä eteenpäin.

		9 elementtiä testattu	**7 x toimi odotetusti, 2 x puutteita**	**6 x toimi odotetusti, 3 x puutteita**

Tulokset olivat lähes samanlaiset Android- ja iOS-laitteella. Näiden kahden järjestelmän ruudunlukijoilla on joitain termeihin liittyviä eroja, mutta muuten testitulokset ovat todella samantyyliset keskenään.

Suurin ero oli päivämääräntiedon välittyminen ruudunlukijalle. Lähdekoodissa on käytetty muotoa DateFormat.yMd(), jonka pitäisi mukautua käyttäjän asetusten mukaan (Flutter, 2023b). Tämä toimi Androidissa. Sen sijaan iOS kertoi päivämäärän kirjaimellisesti kuten se näkyy näytöllä 3/4/2023 lukien ”3 kautta 4 kautta 2023”. Päivämäärä oli ruudulla formaatissa m/d/y, joten väärinymmärryksen vaara on olemassa.
Liukukytkimen muuttunut tieto ei välittynyt ruudunlukijalla käyttäjälle kummassakaan järjestelmässä oikein, vaan laahasi jäljessä. Vasta kun käyttäjä kävi välillä toisessa elementissä, niin ruudunlukija kertoi näytöllä näkyvän tiedon oikein. Päivämäärä välittyi molemmilla ruudunlukijoilla väärin. Lähdekoodissa on käytetty muotoa DateFormat.yMd(), jonka pitäisi mukautua käyttäjän asetusten mukaan, mutta näin ei käynyt (Flutter, 2023b).

Yksittäisten elementtien testauksen ulkopuolelta voidaan todeta myös, että elementistä toiseen siirtyminen ei ollut aina sujuvaa. Testivideoilla on kuvattu sisäänkirjautumistoiminto, jossa käyttäjä täyttää sähköpostiosoitteensa ja salasanan. Tekstikentät oli ryhmitelty samaan ryhmään. IOS-laitteella ensimmäisen kentän täyttämisen jälkeen eteenpäin pääsee valitsemalla näppäimistöstä ”Seuraava” painikkeen, jolloin ruudunlukijan fokus siirtyy salasanakenttään. Ruudunlukija ei kuitenkaan ilmoittanut kentän nimeä tässä vaiheessa, joten käyttäjä ei tiedä missä lomakkeen kentässä ruudunlukijan fokus on. Androidilla testattaessa vastaavaa ei tapahtunut.

5.3 Vertailun yhteenveto

Yhteenvetona voidaan todeta, että tulokset olivat samansuuntaiset. Vaikka kaikki elementit eivät olleetkaan identtiset React Nativessa ja Flutterissa, niin molemmissa teknologioissa Android- ja iOS-laitteet toimivat lähes samoin. Joitain eroja toki oli, ja esimerkiksi React Nativessa muokattava tekstikenttä voitiin helpommin saattaa saavutettavaksi Androidilla

kuin iOSilla. Jotta molemmille laitteille saatiin kohtuullinen tulos, jouduttiin tinkimään Androidin ominaisuuksista. Yksi käytön kannalta merkittävä ero oli myös iOSin näppäimistössä verrattuna Androidin näppäimistöön. Esimerkiksi React Native-testauksessa iOS-laitteen numeronäppäimistöllä ei päässyt lomakkeella eteenpäin, vaan VoiceOver oli laitettava pois päältä, jotta kentästä päästiin eteenpäin. Tämän elementin kohdalla voisi sanoa, että iOS hankaloitti kehitystä.

Päivämäärän kohdalla oli myös eroja lähinnä Android- ja iOS-laitteille, ei niinkään React Nativen ja Flutterin välillä. Androidin ruudunlukija selvisi paremmin päivämäärän lukemisesta ja lopputulos oli ymmärrettävämpi. Tässä on varmasti osuutta erilaisilla päivämäärän esitystavoilla ja niiden tulkinnalla ruudunlukijoita käyttäen. Pohjois-Amerikkalainen tapa esittää päivämäärä muodossa ”kuukausi/päivä/vuosi” ei välttämättä käänny oikein, vaikka koodissa tuettaisiinkin paikallista muotoa.

Saavutettavuuden kehittäminen Flutterilla osoittautui hieman yksinkertaisemmaksi. Saavutettavuus-widgetit Semantics ja MergeSemantics tuo widgetit ruudunlukijalla käytettäviksi ja osa elementeistä on saavutettavia jo alunperin. Hienosäätöä voi tehdä esimerkiksi semanticsLabelin avulla, joka tuo ruudunlukijan kautta lisätietoa käyttäjälle, mutta ei näy käyttöliittymässä visuaalisesti. React Nativessa vastaava toiminto tehdään accessibilityLabelin avulla. React Nativessa voidaan tehdä samat toiminnot, mutta monissa kohden Androidille ja iOSille on omat komennot, joten sama asia pitää kirjoittaa kahdesti. Lisäksi kaikkia samoja toimintoja ei ole saatavilla Androidille ja iOSille. Toisaalta taustalla voi olla myös ylipäätään erilainen toimintatapa Andoid- ja iOS-laitteissa, joten tämä vaatii kehittäjältä näiden eri järjestelmien tuntemusta, tai vähintään mahdollisuutta testata molemmilla käyttöjärjestelmillä.

6 Johtopäätökset ja pohdinta

Kansainvälinen WCAG-standardi tuo tärkeitä ohjenuoria sovelluskehittäjille, vaikka jättääkin jonkin verran tulkinnan varaa ja on osittain hankalasti luettavaa tekstiä. Saavutettavuus on aiheena laaja ja vaatii aikaa perehtymiseen – pelkkä WCAG-ohjeistuksen läpi lukeminen ei riitä saavutettavan sovelluksen tuottamiseen. Sen lisäksi tietoa on hyvä etsiä eri lähteistä ja kasvattaa ymmärrystä siitä, miten saavutettavat palvelut hyödyttävät erilaisia henkilöitä ja miten esimerkiksi näkövammaiset käyttävät ruudunlukijaa.

Ruudunlukijakyselyissä vanhemmat käyttäjät olivat edustettuina isona käyttäjäjoukkona, näin oli sekä WEBAim'n että Eficoden ja Annanpuran järjestämissä kyselyissä. Taustoja tuntematta on tietenkin hankala ottaa kantaa siihen, onko tietty ikäryhmä yliedustettu kyselyssä tai onko esimerkiksi näkövammaisten liittojen jäsenistö painottunut tiettyyn ikäryhmään. Olisi hyödyllistä saada tietoa myös nuorempien näkövammaisten ruudunlukijan käyttötottumuksista. Oletettavasti verkkopalveluiden käyttötaidot eivät olisi nuoremmilla ainakaan keskivertoa heikommat.

6.1 Ruudunlukijatestaus

Ruudunlukijatestaaminen on manuaalista testaamista ja se liittyy yleisemmin saavutettavuustestaukseen ja käytettävyyteen. Saavutettavuuden testaamiseen löytyy useampia automaattisia työkaluja, kuten esimerkiksi Axe devTools-selainlisäosa, Wave Evaluation Tool -selainlisäosa ja Google Scaner for A11y. Sen sijaan ruudunlukijatestaukseen ei löytynyt automattista työkalua. Ruudunlukijatestaaminen on haastavaa ja siksi kiinnostavaa. On olemassa erilaisia tyylejä käyttää ruudunlukijaa. Siihen vaikuttavat käyttäjän omat kyvyt ja mieltymykset – aivan kuten muuhunkin sovellusten käyttöön. Lisäksi ruudunlukijoiden toimintatapaa ja miten paljon ruudunlukija antaa tietoa, voi kukin käyttäjä muokata itselleen sopivaksi. Näin ollen voisi väittää, että kahden käyttäjän kokemukset samasta sovelluksesta voivat olla hyvin erilaiset.

Ruudunlukijalla saa kuitenkin jo pienimuotoisella testaamisella tietoa siitä, onko sovelluksen rakenne looginen ja saako käyttäjä tarvitsemansa tiedon käyttöliittymäkomponenttien toiminnasta. Erilaisilla elementeillä on oletettuja toimintatapoja, joita voidaan testata

kohtuullisen helposti. Hyvänä lähtötasona voi pitää sitä, että mikään toiminto ei tule käyttäjälle yllätyksenä. Kokonaisen sovelluksen tai sivuston testaaminen onkin jo sitten haastavampaa, ja vaatii testaajaltakin enemmän tietämystä.

6.2 Kehitysprojekti

Jotta molemmilla teknologiolla päästäisiin mahdollisimman hyvää lopputulokseen, käyttöön haettiin mahdollisimman paljon valmista lähdekoodia ns. virallisista lähteistä (React Native sekä Flutter-dokumentaatioista ja esimerkkisovelluksista). Vaikka molemmat teknologiat tuovat esiin saavutettavuutta ja saavutettavuusominaisuuksia, huomattiin, että koodiesimerkeissä niitä ei ollut huomioitu.

Projektin käytännön osassa testauksen pääpaino oli yksittäisten elementtien ruudunlukijakäyttäytymisessä. Testaamisen aika huomattiin, että vaikka React Nativen ja Flutterin dokumentaatioissa annettiin tietynlaisia käyttäytymismalleja tietyille elementeille tai ominaisuuksille, niissä saattoi kuitenkin olla eroja käytännössä. Erot voivat johtua käytössä olleesta testilaitteesta, järjestelmäversiosta tai jostain musta seikasta. Se on kuitenkin myös todellisuutta varsinaisten sovellusten käyttäjien keskuudessa. Käytössä on erilaisia laitteita, erilaisia käyttöjärjestelmäversioita ja erilaisia laite- ja sovellusasetuksia – silti pitäisi pystyä kehittämään sovellus, joka olisi mahdollisimman hyvin saavutettava mahdollisimman monelle käyttäjälle.

Projektin aikana huomattiin myös, että Android- että iOS-laitteiden ruudunlukijakäyttäytymisessä voi olla osittain merkittäviäkin eroja. Kehittämisen aikana on hyödyllistä päästä testaaman sovellusta molemmilla laitteilla. Tämä tuli esiin erityisesti React Nativella. Kun pääasiallista testaamista suoritettiin Android-laitteella, toiminta iOS-laitteella ei ollut toivottua ja joitain kompromisseja jouduttiin tekemään. Flutterilla muokattu testisovellus sen sijaan toimi hyvin samantyylisesti sekä Androidilla että iOSilla.

Käyttäjätutkimuksien tuloksista ilmeni, että suurin osa ruudunlukijan käyttäjistä oli valinnut käyttönsä Applen laitteen. Jos tähän on johtanut se, että macOS- ja iOS-laitteet ovat olleet edelläkävijöitä saavutettavuudessa, voidaan tämän projektin osalta todeta, että Android ei

ole suinkaan iOSia huonompi. Esimerkiksi React Nativella tekstimuokkauskenttä oli helpompi saada saavutettavaksi Androidille.

6.3 React Native vs. Flutter

React Native- ja Flutter-teknologiolla voidaan molemmilla kehittää ns. cross-platform sovelluksia eli useilla alustoilla toimivia sovelluksia. Niissä on kuitenkin eroja toisiinsa nähden. React Native käyttää JavaScriptiä ja JSX-syntaksilaajennusta, Flutterissa taas ohjelmoidaan Dart-kielellä. React Nativessa on peruskoodipohja, jota Meta ylläpitää, mutta paljon käytetään kolmansien osapuolten kehittämiä kirjastoja. Flutterissa puolestaan kehitys on keskittynyt Googlelle. Molemmissa on omat hyvät puolensa. React Nativen kehitys ei ole kovin riippuvainen yhdestä tietystä osapuolesta – Flutterilla taas on vahva tuki Googlella eikä ole syytä olettaa, että Google lopettaisi oman mobiilikehitysteknologiansa tukea.

Stack Overflow on verkkosivusto, jossa ratkotaan mm. ohjelmointiin liittyviä kysymyksiä. Se suorittaa vuosittain kyselyn sivuston käyttäjien keskuudessa. Vuoden 2022 kyselyssä (Stack Overflow, 2022) todetaan että Java Script oli kaikkein suosituin ohjelmistokehityksen kieli jo 10 vuotta peräkkäin. Sillä oli noin 65 % kannatus maailmanlaajuisesti. Dart-kielellä puolestaan n. 6,5 %. Kyselyyn vastanneita oli reilut 70000. Vertasin lukuja muutaman edellisen vuoden tuloksiin - Dart-kielen suosio on hiljalleen kasvanut, vaikka onkin edelleen melko vähäinen verrattuna Java Scriptiin, jota käytetään runsaasti muussakin ohjelmistokehityksessä. Samassa kyselyssä todettiin, että React Native ja Flutter olivat suosituimmat cross-platform teknologiat, molemmat noin 12,5 % kannatuksella. Vaikka molemmat teknologia ovat suunnilleen yhtä suosittuja, on Java Script osaajia määrällisesti enemmän, joten rekrytointi pitäisi olla helpompaa.

Tässä projektissa tehtyjen saavutettavuustestien perusteella React Nativessa ja Flutterissa ei ole merkittäviä eroja, joten valinta näiden kahden teknologian välillä on syytä tehdä muilla perusteilla.

6.4 Opinnäytetyön hyödyt ja jatkotoimet

Tämä opinnäytetyö on tuonut hyötyä etenkin toimiin saavutettavuuden testauksessa Ahola Digitalissa. Vaikka WCAG-kriteerit eivät tähän mennessä ole tehneet eroja mobiili- ja verkkosovellusten saavutettavuuteen, on selvää, että niiden käytössä on kuitenkin eroja. Näin ollen hyvä ymmärtää, mitä erityispiirteitä mobiilisovelluksissa on huomioitava, jotta niiden käyttö avustavilla teknologioillakin olisi mahdollisimman sujuvaa.

Saavutettavuus on läheisesti yhteydessä käytettävyyteen. Etenkin avustavia teknologioita käyttävien henkilöiden kannalta voisi sanoa jopa, että sovellus ei voi olla käytettävä ilman että se on saavutettava. Saavutettavuus ei kuitenkaan vielä takaa sovelluksen käytettävyyttä ja siksi olisikin tärkeää, että sovellusten kehittäjät ja suunnittelijat ymmärtävät kuinka erilaiset henkilöt mobiililaitteita tai tietokoneita - avustavien teknologioiden kanssa tai ilman. Seuraava askel mobiilikehityksessä voisi olla nimenomaan käytettävyyden kehittäminen. Käytettävyyden tutkimiseen on erikoistunut Nielsen Norman Group, joka ovat tehnyt käytettävyysohjeistuksen liittyen saavutettavaan verkkosuunnittelun (Pernice & Nielsen, 2001).

7 Yhteenveto

Tutkimusaihe oli melko laaja, mutta erityisesti oman työni kannalta hyvin mielenkiintoinen. Tutkimuskysymykset keskittyivät nimenomaan mobiilisovellusten saavutettavuuteen, joten se auttoi rajaamaan sisältöä. Saavutettavuudesta löytyy melko paljon materiaalia, mutta iso osa siitä koskee tietokoneella käytettäviä verkkosivuja ja -sovelluksia. Mobiilisovellusten erityispiirteitäkin on kuitenkin käsitelty, ja onnistuin löytämään sekä erityisesti mobiilisovelluksia koskevia saavutettavuuskriteereitä, saavutettavan mobiilikehityksen erityispiirteitä yleisesti sekä React Nativea ja Flutteria koskevia toimia. Myös mobiilitestaamisen suhteen löytyi joitakin erityisohjeita. Näiden suhteen tutkimuskysymyksiin vastaaminen onnistui. Sain opinnäytetyöprojektin aikana kerättyä näistä aiheista paljon lisätietoa ja aion omassa työssäni vielä jatkaa tämän aiheen parissa.

Viimeinen tutkimuskysymys koski React Nativen ja Flutterin eroja saavutettavuuden suhteen. Siihen olikin hankalampaa löytää vastausta. Teknologiat ovat kovin erilaisia, mutta molemmilla voidaan saavuttaa hyviä tuloksia. Teknologioiden välillä ei tässä opinnäytetyössä tehdyn testauksen yhteydessä löydetty suuriakaan eroja. Molemmilla voidaan saavuttaa hyvin saman tyyliset tulokset, kun testataan tiettyjä komponentteja. Lopulta kyse on ehkä enemmänkin siitä, kumpi teknologia tuntuu itselle luontevammalta. Saavutettavuuteen löytyy keinoja molemmista. Koska kuitenkin testaaminen on iso osa saavutettavan sovelluksen luomista, kallistuu vaaka hienoisesti React Nativen puolelle.

Opin paljon tämän opinnäytetyöprosessin aikana, erityisesti sovellustestaamisesta ja siitä miten yksinkertainenkin saavutettavuustoiminto voi vaatia runsaasti pohtimista ja eri vaihtoehtojen kokeilua. Esimerkiksi placeholderin lisääminen lomakekenttään voi aiheuttaa täydellisen muutoksen ruudunlukijan välittämään tietoon. Opin myös, että vaikka internetistä löytyy paljon valmista koodia ja esimerkkitapauksia, niissä harvoin on otettu huomioon saavutettavuutta. Toisaalta myös jokainen sovellus on erilainen, ja se miten käyttökokemuksen saa etenemään vaivattomasti, vaatii työtä. Yksittäinen elementti voi olla saavutettava, mutta myös siirtymien tulisi sujua helposti. Aloittelevan sovelluskehittäjän tai -testaajan on hyvä varata runsaasti aikaa erilaisten vaihtoehtojen kokeiluun.

Lähteet

Accessible Web. (2020, lokakuuta 12). *WCAG Version History*. https://accessibleweb.com/wcag/wcag-version-history/

AG WG. (2023, helmikuuta 16). *SCR18: Providing client-side validation and alert | WAI | W3C*. https://www.w3.org/WAI/WCAG22/Techniques/client-side-script/SCR18

Aluehallintovirasto. (n.d.a). Kenelle saavutettavuus on tärkeää? *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/kenelle-saavutettavuus-on-tarkeaa/

Aluehallintovirasto. (n.d.b). Muita lakeja. *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/muita-lakeja/

Aluehallintovirasto. (n.d.c). Saavutettavuuden lait ja standardit. *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/

Aluehallintovirasto. (n.d.d). Tietoa saavutettavuusselosteesta. *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/tietoa-saavutettavuusselosteesta/

Aluehallintovirasto. (n.a.). Yleistä saavutettavuudesta. *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/

Apple Inc. (n.d.a). *IPhonen VoiceOver-eleiden opetteleminen*. Apple Support. https://support.apple.com/fi-fi/guide/iphone/iph3e2e2281/16.0/ios/16.0

Apple Inc. (n.d.b). *Laita VoiceOver päälle ja harjoittele VoiceOveria iPhonessa*. Apple Support. https://support.apple.com/fi-fi/guide/iphone/iph3e2e415f/16.0/ios/16.0](https://support.apple.com/fi-fi/guide/iphone/iph3e2e415f/16.0/ios/16.0

Apple Inc. (n.d.c). *Layout—Foundations—Human Interface Guidelines—Design—Apple Developer*. https://developer.apple.com/design/human-interface-guidelines/foundations/layout/#best-practices

Apple Inc. (n.d.d). *Tietoja VoiceOver-roottorista iPhonessa, iPadissa ja iPod touchissa*. Apple Support. https://support.apple.com/fi-fi/HT204783

Bigio, M. (2016, maaliskuuta 24). *Introducing Hot Reloading · React Native*. https://reactnative.dev/blog/2016/03/24/introducing-hot-reloading

Callstack Open Source. (n.d.). *API | React Native Testing Library*. https://callstack.github.io/react-native-testing-library/docs/api

Celia. (n.d -a). *Saavutettavuus*. Celia. https://www.celia.fi/saavutettavuus/

Celia. (n.d. -b). *Verkkopalvelujen saavutettavuusohjeita*. Celia. https://www.celia.fi/saavutettavuus/verkkopalvelujen-saavutettavuus/

Dandekar, K., Raju, B. I., & Srinivasan, M. A. (ei pvm.). 3-D Finite-Element Modelsof Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense. *Journal of Biomechanical Engineering*, *2003*(125), 682–691. https://doi.org/10.1115/1.1613673

Dodson, R. (2018, marraskuuta 18). *Semantics and screen readers*. Web.Dev. https://web.dev/semantics-and-screen-readers/

Flutter. (2023a). *flutter_test library—Dart API*. https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html

Flutter. (2023b, tammikuuta 3). *DateFormat class—Intl library—Dart API*. https://api.flutter.dev/flutter/intl/DateFormat-class.html

Flutter. (n.d.a). *Accessibility widgets*. https://docs.flutter.dev/development/ui/widgets/accessibility

Flutter. (n.d.b). *Flutter samples*. https://flutter.github.io/samples/form_app.html

Flutter. (n.d.c). *Hot reload*. https://docs.flutter.dev/development/tools/hot-reload

Flutter. (n.d.d). *Multi-Platform*. Flutter. https://flutter.dev/multi-platform

Google. (n.d. -a). *Accessibility – Material Design 3*. https://m3.material.io/foundations/accessible-design/accessibility-basics

Google Support. (n.d.a). *TalkBack—Androidin esteettömyys Ohjeet*. https://support.google.com/accessibility/android/topic/3529932?hl=fi&ref_topic=9078845](https://support.google.com/accessibility/android/topic/3529932?hl=fi&ref_topic=9078845

Google Support. (n.d.b). *TalkBack-eleiden käyttäminen—Androidin esteettömyys Ohjeet*. https://support.google.com/accessibility/android/answer/6151827?hl=fi&ref_topic=10601570#zippy=%2Cperusnavigointi%2Cversio-ja-uudemmat](https://support.google.com/accessibility/android/answer/6151827?hl=fi&ref_topic=10601570#zippy=%2Cperusnavigointi%2Cversio-ja-uudemmat

Google Support. (n.d.c). *TalkBackilla liikkuminen laitteella—Androidin esteettömyys Ohjeet*. https://support.google.com/accessibility/android/answer/6006598?hl=fi&ref_topic=10601571

Google Support. (n.d.d). *TalkBackin Android-asetusten muuttaminen—Androidin esteettömyys Ohjeet*. https://support.google.com/accessibility/android/answer/6283655?hl=fi&ref_topic=10601571#zippy=%2Cpuheen-m%C3%A4%C3%A4r%C3%A4n-muuttaminen

Helsingin kaupunki. (n.d.). *Fyysiset ja motoriset rajoitteet—Saavutettavuusmalli—Helsingin kaupunki*. Saavutettavuusmalli. https://saavutettavuusmalli.hel.fi/fyysiset-ja-motoriset-rajoitteet/

Kallionpää, R., & Kiiskilä, R. (2021). *Suomalaisten ruudunlukijäkäyttäjien tottumukset ja haasteet verkkopalveluiden käytössä*. https://urly.fi/2Ulu

Kearney, D. (2015, syyskuuta 17). *Mobile design 101: Pixels, points and resolutions*. https://blog.fluidui.com/designing-for-mobile-101-pixels-points-and-resolutions/

Kehitysvammaliitto ry. (n.d. -b). *Fyysiset ja motoriset rajoitteet*. Papunet. https://papunet.net/saavutettavuus/fyysiset-ja-motoriset-rajoitteet

Kehitysvammaliitto ry. (n.d. -c). *Kognitiiviset ja kielelliset vaikeudet*. Papunet. https://papunet.net/saavutettavuus/kognitiiviset-ja-kielelliset-vaikeudet

Kehitysvammaliitto ry. (n.d. -a). *Kuka hyötyy saavutettavuudesta?* Papunet. https://papunet.net/saavutettavuus/kuka-hyotyy-saavutettavuudesta

Kuuloliitto ry. (n.d.). *Kuulovammaisena digituessa*. Kuuloliitto. https://www.kuuloliitto.fi/toiminta/kuulovammaisena-digiopastuksessa/

Laki digitaalisten palvelujen tarjoamisesta 306/2019. (2019, maaliskuuta 15). Oikeusministeriö. https://www.finlex.fi/fi/laki/alkup/2019/20190306#Lidm45949344664080

Lamminen, V. (2020, elokuuta 4). *Hyvä tietää ruudunlukuohjelmista*. Saavutettavasti.fi. https://www.saavutettavasti.fi/hyva-tietaa-ruudunlukuohjelmista/

Lizama, E. (2021, heinäkuuta 24). Flutter accessibility: Getting started. *Medium*. https://medium.com/@enzoftware/flutter-accessibility-getting-started-52286746314e

Mazzarolo, M. (2023). *React-native-modal-datetime-picker* [JavaScript]. https://github.com/mmazzarolo/react-native-modal-datetime-picker (Original work published 2016)

Meta Platforms, Inc. (2022a, syyskuuta 5). *AccessibilityInfo · React Native*. https://reactnative.dev/docs/accessibilityinfo

Meta Platforms Inc. (2022, syyskuuta 6). *Core Components and Native Components · React Native*. https://reactnative.dev/docs/intro-react-native-components

Meta Platforms, Inc. (2022b, marraskuuta 7). *Using Libraries · React Native*. https://reactnative.dev/docs/libraries

Meta Platforms, Inc. (2023, tammikuuta 12). *Accessibility · React Native*. https://reactnative.dev/docs/accessibility

Meta Platforms Inc. (2023a, tammikuuta 12). *AccessibilityLabel—Accessibility—React Native*. https://reactnative.dev/docs/accessibility

Meta Platforms Inc. (2023b, tammikuuta 12). *Core Components and APIs · React Native*. https://reactnative.dev/docs/components-and-apis

Meta Platforms Inc. (2023c, tammikuuta 12). *Testing · React Native*. https://reactnative.dev/docs/testing-overview

Meta Platforms Inc. (2023d, tammikuuta 13). *Setting up the development environment · React Native*. https://reactnative.dev/docs/environment-setup

Muistiliitto ry. (n.d.). *Hyvä työ aivoille | Muistiliitto*. www.muistiliitto.fi. https://www.muistiliitto.fi/fi/aivot-ja-muisti/aivoterveys/hyva-tyo-aivoille

Näkövammaisten liitto ry. (2019a, syyskuuta 23). *Saavutettavuus*. https://www.nkl.fi/fi/saavutettavuus

Näkövammaisten liitto ry. (2019b, syyskuuta 23). *TalkBack-ruudunlukuohjelmalla testaaminen*. https://www.nkl.fi/fi/talkback-ruudunlukuohjelmalla-testaaminen

Näkövammaisten liitto ry. (2019c, syyskuuta 23). *VoiceOverilla testaaminen | Näkövammaisten liitto*. https://www.nkl.fi/fi/voiceoverilla-testaaminen

Näkövammaisten liitto ry. (2021, syyskuuta 23). *Suurin osa näkövammaisista kokee digipalvelut haastaviksi käyttää*. nkl.fi. https://www.nkl.fi/fi/artikkeli/suurin-osa-nakovammaisista-kokee-digipalvelut-haastaviksi-kayttaa

Pernice, K., & Nielsen, J. (2001). *Usability Guidelines for Accessible Web Design. Based on Usability Studies with People Using Assistive Technology*. www.nngroup.com/reports/usability-guidelines-accessible-web-design

Pesonen, T. (2021, huhtikuuta 3). *Testausohje—Saavutettavuusmalli—Helsingin kaupunki*. Saavutettavuusmalli. https://saavutettavuusmalli.hel.fi/testaaminen/

Sapega, M. (2021, lokakuuta 18). *What is a screen reader, and why are they important?* TPGi. https://www.tpgi.com/what-is-a-screen-reader-and-why-are-they-important/

Stack Overflow. (2022, toukokuuta). *Stack Overflow Developer Survey 2022*. Stack Overflow. https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022

Suomi.fi Design System. (n.d.). *Komponenttien käyttöohje*. https://designsystem.suomi.fi/components/

W3C. (2015, helmikuuta 26). *Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile*. https://www.w3.org/TR/mobile-accessibility-mapping/

W3C. (2022, heinäkuuta 6). *WCAG 3 Introduction*. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/standards-guidelines/wcag/wcag3-intro/

W3C. (n.d.a). *Documents published at W3C*. https://www.w3.org/standards/types#CRY

W3C. (n.d.b). *Understanding Success Criterion 2.4.6: Headings and Labels*. http://www.w3.org/TR/UNDERSTANDING-WCAG/

Watson, L. (n.d.). What is a screen reader? *Nomensa*. https://www.nomensa.com/blog/what-screen-reader

WCAG 2.1. (2018a, kesäkuuta 5). *WCAG 2.1 at a Glance*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/standards-guidelines/wcag/glance/

WCAG 2.1. (2018b, kesäkuuta 5). *Web Content Accessibility Guidelines (WCAG) 2.1*. https://www.w3.org/TR/WCAG21/

WCAG 2.1. (2019, lokakuuta 4). *How to Meet WCAG (Quickref Reference)*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/WCAG21/quickref/

WCAG 2.2. (2023, tammikuuta 25). *Web Content Accessibility Guidelines (WCAG) 2.2*. https://www.w3.org/TR/WCAG22/#new-features-in-wcag-2-2

WebAIM. (2021). *Screen Reader User Survey #9 Results*. https://webaim.org/projects/screenreadersurvey9/#mobiledevice

World Health Organization (WHO). (2021, marraskuuta 24). *Disability and health*. https://www.who.int/news-room/fact-sheets/detail/disability-and-health

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojektin tuloksena syntyneet React Native- ja Flutter-sovellusten lähdekoodi on tekijän tietokoneen C-asemalla, sekä tallennettu GitHub-versiohallintapalveluun tekijän oman käyttäjäprofiilin (SannaKN) alle julkiseen säilytykseen, master-haaraan (master branch). React Native-projektikansio on ReactNativeComponentTests ja Flutter-projektikansio on form_app. Molempien README-tiedostosta löytyy YouTube-linkit sekä Android- että iOS-laitteella tehtyihin ruudunlukijatesteihin. Lisäksi React Nativella testattiin päivämäärään liittyviä toimintoja erillisellä Expossa toimivalla kirjastolla, jonka lähdekoodi löytyy Githubista: https://github.com/mmazzarolo/react-native-modal-datetime-picker. Lisäksi ohjelman lähdekoodi on pakattuna tekijän C-asemalla ja tallennettuna Expo Go-sovellukseen. Päivämäärän testauksen videolinkit löytyvät myös edellä mainitun React Native-projektikansion README-tiedostosta.

Ruudunlukijatestien videoiden perusteella on tehty opinnäytetyössä esitetyt taulukot erilaisten käyttöliittymäkomponenttien toiminnoista ruudunlukijalla. Projektien lähdekoodia säilytetään tekijän C-asemalla sekä GitHub-palvelussa ainakin vuosi opinnäytetyön valmistumisesta. Videoita säilytetään samoin tekijän tietokoneen C-asemalla sekä YouTube-videopalvelussa vähintään vuosi opinnäytetyön valmistumisesta. Projekti ei sisällä toimeksiantajan luottamuksellisia tietoja, eikä materiaalia ole tarpeen anonymisoida tai salata.