Saavutettavuuden kehittäminen sovelluksessa

Aleksandr Seppenen

Haaga-Helia ammattikorkeakoulu

Amk-opinnäytetyö

2022

Tradenomin tutkinto

Tiivistelmä

Tekijä(t)
Seppenen Aleksandr

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Saavutettavuuden kehittäminen sovelluksessa

Sivu- ja liitesivumäärä
30 + 0

Digitaalisista verkkopalveluista on tullut erottamaton osa jokaisen elämää. Kuitenkin useat verkkopalvelut eivät ole saavutettavissa kaikille käyttäjille. Tässä työssä tarkastellaan saavutettavuuden merkitystä käyttöliittymässä ja sen teknistä toteutusta verkkosovelluksessa.

Tämän työn tarkoituksena oli määritellä, mikä on saavutettavuus ja sen sisällönrakenne, tutkia menetelmiä saavutettavuuden kehittämiseen soveltamalla niitä käytännössä verkkosovelluksessa.

Tämä työ on toteutettu todellisella sovelluksella, jossa sivujen virheiden tunnistaminen suoritettiin analysoimalla jokainen applikaation käyttäjälle näkyvä elementti. Tässä työssä kuvataan myös työkalut ja menetelmät, joita tulee käyttää virheanalyysissä. Teoreettiset ja tekniset ratkaisut on liitetty jokaiselle yksittäiselle elementille, jotta ymmärretään, mitkä tekniset ratkaisut tulisi toteuttaa saavutettavan sovelluksen aikaansaamiseksi.

Tämän työn lopussa tehtiin myös pohdinta tehdystä työstä, jotta saatiin selville lähestymistavan edut, sekä haitat saavutettavan sovelluksen ohjelmointia varten.

Asiasanat
Saavutettavuuden kehittäminen sovelluksessa, WCAG, verkkopalvelu

Sisällys

1 Johdanto 1
2 Saavutettavuus 2
2.1 Mitä on saavutettavuus 2
2.2 WCAG 3
2.3 Tekninen saavutettavuus 5
2.4 Yhteenveto 7
3 Verkkosivuprojekti 8
3.1 Tavoite ja projektisuunnitelma 8
3.2 Käytetyt teknologiat 10
3.3 Toteutus 11
3.3.1 Menetelmät 11
3.3.2 Yleiset havainnot 11
3.3.3 Välilehdet 12
3.3.4 Sopimustiedot 14
3.3.5 Muokkaa yhteyshenkilöä 17
3.3.6 Vakuutusturva 20
3.3.7 Vakuutetut 21
3.3.8 Lisää vakuutettu 22
3.3.9 Maksuerittely 24
3.3.10 Roolit 25
3.4 Tuotos 25
4 Pohdinta 26
Lähteet 28

1 Johdanto

Tämän opinnäytetyön tavoite on selvittää mitä on saavutettavuus ja siihen liittyvät seikat. Sen lisäksi tavoitteena on näyttää, miten saavutettavuutta toteutetaan käyttöliittymissä teknisesti. Digitalisaation aikana käytämme päivittäin digitaalisia palveluita. Tämä helpottaa ja tehostaa yritysten asiakaspalvelua ja säästää asiakkaiden aikaa. Verkkosovellusten ansiosta ei tarvitse seisoa jonossa pankissa, tai vakuutusyhtiössä asiakirjoja täyttääkseen. Lähes jokaisella yrityksellä, varsinkin julkisella palvelulla, on verkkosivusto tai verkossa toimiva sovellus. Verkkopalveluista on nykyään tullut osa jokaisen elämää ja niiden tulee olla kaikkien ihmisen saatavilla fyysisistä rajoituksista riippumatta.

Verkkopalvelu on ensisijaisesti käyttöliittymä. Kaikki esineet, joiden kanssa ihminen on vuorovaikutuksessa, jopa ovenkahva tai vesihana ovat myös käyttöliittymiä. Nykyään tosielämässä esteet ovat vähäiset ihmisille kenellä on esimerkiksi motoriikka tai liikkumisongelmia ja niitä kehitetään jatkuvasti, mutta verkkopalveluissa on edelleen monia esteitä.

Tässä työssä käsitellään kansainvälistä saavutettavuusstandardia (WCAG), sen sisältöä ja sen soveltamista käytännössä Java -ohjelmointikielellä toteutettuun sovellukseen. Lisäksi tarkastellaan työkaluja saavutettavuuden analysointiin ja kehittämiseen. Pääpaino tulee olemaan virheiden etsinnässä, niiden ratkaisuissa ja teknisessä toteutuksessa todellisella sovelluksen käyttöliittymällä. Työ sisältää myös tutkintaa mahdollisuuksista toteuttaa saavutettavuutta käytännössä vanhentunutta teknologiaa sisältävissä sovelluksissa. Tämän toteutuksen ansiota sovelluksen saavutettavuuden tason tulisi olla merkittävästi vahvempi ja sovelluksen saavutettavuuden tulee olla kansainvälisten kehitysstandardien mukainen. Lopussa tehdään johtopäätökset tehdystä työstä, käytetyistä teknologioista.

Tämän työn keskeiset käsitteet ovat saavutettavuuden kehittäminen sovelluksessa, WCAG, ja verkkopalvelun saavutettavuus. Saavutettavuudella tarkoitetaan kykyä käyttää jotakin olemassa olevaa verkkopalvelua käyttäjän fyysisistä rajoitteista, esimerkiksi näkökyvyttömyydestä huolimatta. WCAG on kansainvälinen standardi, jota hyödynnetään saavutettavuuden toteuttamista varten. Verkkopalvelulla tarkoitetaan palvelua internetissä. Monet suomalaiset käyttävät esimerkiksi omakela- tai omavero- verkkopalveluita.

2 Saavutettavuus

2.1 Mitä on saavutettavuus

Kun puhumme saavutettavuudesta, sillä tarkoitetaan käyttöliittymän saatettavuutta kaikille käyttäjille fyysisistä rajoitteista huolimatta. Nyt maailmassa on nyt noin 1 000 000 000 ihmistä eli noin 15 % maailman väestöstä, joilla on jokin vamma. Heistä 190 miljoonaa ovat vähintään 15-vuotiaita, joilla on hengenvaikeuksia eli vamma. Sanalla "vammainen", viittaa ihmisiin, joilla on fyysistä tai kognitiivisia -vammoja ja yleensä kuitenkin käytetään sanaa invalidi. Kaikki nämä vammojen muodot ovat hyvin erilaisia. Ja tällaisten ihmisten määrä kasvaa jatkuvasti, joka johtuu erityisesti väestökehityksestä ja ihmisten eliniän pidentymisestä. Melkein jokainen kokee jossain vaiheessa elämäänsä jonkinlaisen vamman. Tällaiset ihmiset kokevat rajallista tai heikkolaatuista digitaalisten palveluiden käyttöä jopa sellaisissa peruspalveluissa kuin terveydenhuolto, finanssiala, näitä ovat pankit ja vakuutuslaitokset. (WHO 2021).

Kaikki ymmärtävät sanan esteettömyys liikkumiseksi ympäristössä ilman esteitä, esimerkiksi kuka tahansa pääsee elokuviin ilman esteitä, vaikka liikkuu pyörätuolissa. Samaan tarkoitukseen käytetään sanaa saavutettavuus tilanteissa, joissa puhutaan digitaalisista palveluista digitaalisessa ympäristössä. Saavutettavuus pyrkii siihen, että kaikki ihmiset terveydentilastaan riippumatta kykenevät käyttämään verkkosivuja ja mobiilisovelluksia samalla tavalla ja samoin tuloksin. (Aluehallintovirasto).

Peruskäyttäjä ei välttämättä ymmärrä, onko sivusto saavutettava vai ei, mutta vammaisen käyttäjän osalta tilanne voi olla hyvin erilainen. Tosiasia on, että näkövammaisilla vuorovaikutus ympäristön kanssa tapahtuu eri tavalla kuin tavallisilla ihmisillä. Samalla tavalla he näkevät verkkosisällön eri tavalla kuin peruskäyttäjät ja he käyttävät verkkosivustoja meille tuntemattomilla tavoilla. Näköongelmista kärsivät turvautuvat useimmiten näytönlukuohjelmiin, mutta tämä ratkaisu ei ole tehokas, jos itse käyttöliittymä ei ole saavutettava. Tilastojen mukaan näkövammaiset ovat Suomessa suurin vammaisten ryhmä, noin 5 % vammaisten kokonaismäärästä. Vain 22 % tästä määrästä on täysin sokeita. Olemassa on myös muita näkövammoja, kuin sokeus. Näitä ovat esimerkiksi värisokeus, silmänpohjan rappeumat, verkkokalvon perinnölliset rappeumat ja muut ongelmat, joissa ihminen ei näe kunnolla. (Näkövammaisten liitto 2022).

On olemassa myös muita saavutettavuutta hankaloittavia vammoja, kuten esimerkiksi lukihäiriö, kuulovammat ja kuurous fyysiset ja motoriset rajoitteet, kuten CP-vamma, lihasheikkous, vapina tai halvaantuminen ja tilapäiset haasteet, kuten meluisa ympäristö, kirkas auringonpaiste, stressi tai kipsissä oleva käsi. Tästä voidaan päätellä ongelman

suuruus. Vaikka vammat voivat olla hyvin erilaisia, kaikille ne saattavat tuottaa vuorovaikutusongelmia käyttöliittymän kanssa ja itse palveluiden käyttö voi olla haasteellista.

Haasteet digitaalisten palveluiden käytössä voivat olla hyvin erilaisia. Esimerkiksi sokea käyttäjä lukee verkkosivustoa näytönlukuohjelman avulla tai käyttää vain näppäimistöä hiiren sijaan, mutta koodissa olevan virheen vuoksi sivusto on osittain lukukelvoton näytönlukuohjelmille. Siksi käyttäjä ei voi havaita kaikkia sivuston tietoja tai ne eivät ole täydellisiä. (Aluehallintovirasto).

Tutkimukset osoittavat, että suurin osa Internet-resursseista eivät vieläkään ole saavutettavia. Voittoa tavoittelematon järjestö WebAIM (Web Accessibility In Mind) on tarjonnut savutettavuus ratkaisuja vuodesta 1999 ja on maailmanlaajuinen saavutettavuuden toimittaja. Heidän tutkimuksensa osoittaa, että hyvin harvat verkkosivustot ovat saavutettavissa. Helmikuussa 2019, 2020 ja 2021 WebAIM testasi 1 000 000 000 suosituinta sivustoa saavutettavuuden puitteissa. Tutkimukset ovat osoittaneet, että 98 prosentilla verkkosivustoista on jonkinlaisia saavutettavuusongelmia. Noin 51,4 virhettä per verkkosivu. Vaikka sivuston virheiden määrä väheni 15 prosentilla helmikuun 2020 ja helmikuun 2021 välisenä aikana, nämä sivustot eivät siltikään täytä WCAG:n vähimmäisvaatimuksia. (WebAIM 2022).

Saavutettavuus antaa monia etuja kaiken tyyppisille esteisille käyttäjille. Esimerkiksi tekstitys auttaa paikassa, jossa palvelun käyttö tapahtuu liian äänekkäässä- tai liian hiljaisessa paikassa. Oikea kontrasti tekee sisällöstä luettavampaa kirkkaassa valossa ja ratkaisee näkövammaisten ongelmia. Joillekin ihmisille tämä mahdollisuus on erittäin tärkeä käytön kannalta. Siksi Euroopan parlamentti julkaisi vuonna 2016 saavutettavuusdirektiivin, jonka tavoitteena on saada julkisen sektorin palvelut saavutettaviksi. (Saavutettavuusdirektiivi). Tämän lisäksi Suomen kansallinen laki 306/2019 velvoittaa laajentamaan saavutettavuutta puitteita, mikä käytännössä tarkoittaa sitä että, Suomen laki edellyttää tietyn tason saavutettavuutta myös kaikilta yksityisiltä organisaatioilta, jotka tuottavat julkisia palveluita. (Aluehallintovirasto).

2.2 WCAG

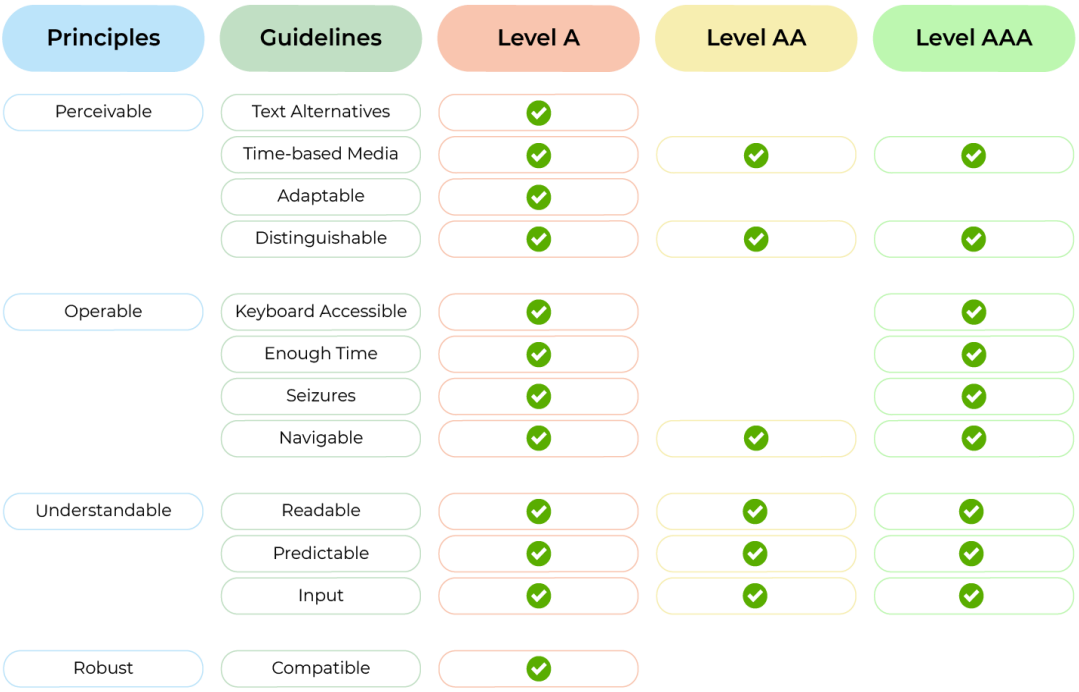
Saavutettavuudella on olemassa myös kansainvälinen standardi ISO 40500 WCAG (eng. Web Content Accessibility Guidelines). WGAC ensimmäinen versio kehitettiin vuonna 1999 osana W3C:n aloitetta yhteistyössä muiden organisaatioiden kanssa ympäri maailmaa yhden yhtenäisen saavutettavuusstandardin luomiseksi. Vuonna 2018 julkaistiin uusi standardiversio WCAG 2.0, jolla on täysin uusi rakenne, koska ensimmäinen versio ei enää pystynyt teknisesti tukemaan tätä standardia. (W3C 2022).

Käytännössä WCAG on joukko kuvauksia ja ohjeistuksia, joita sovelluksen on noudatettava. Itse saavuttavuuden toteutuksen suorittaa kehitystiimi saman WCAG ohjeistuksen mukaisesti.
Tämän ohjeistuksen noudattaminen tekee digitaalisesta palvelusta entistä saavutettavamman sekä vammaisille että monivammaisisille. Lisäksi näiden ohjeistuksen toteuttaminen helpottaa digipalvelun käyttöä riippumatta siitä, onko käyttäjällä rajoituksia vai ei ole. (W3C 2008).

WCAG tarkoituksena on varmista että:

- Sisältö voidaan havaita aputekniikoilla, esimerkiksi ruudunlukijalla
- Kaikilla käyttäjillä on pääsy digipalvelun sisältöön, sekä sen ominaisuuksiin rajoituksista riippumatta
- Sisältö toistuu samanlaisena eri laitteilla

WCAG-ohjeistuksella on hierarkkinen kerrosrakenne, jossa on kolme eri tasoa. Ensimmäinen taso on periaatteet. Nämä ovat perusperiaatteet, jotka jaetaan ohjeistuksiin ja luokkiin. Toinen taso on ohjeistukset. Ne ovat tarkat ohjeet sivun elementeistä ja niiden ominaisuuksista. Kolmas taso ovat luokat tai onnistumiskriteerit, jotka koostuvat tasoista A, AA, ja AAA.



Kuva 1. Hierarkia WCAG. (Openclassrooms 2022).

Kuvassa 1 on näkyvissä WCAG:n hierarkkinen rakenne. Periaatteet ovat päätekijät, jotka perustuvat neljään periaatteeseen: havaittava, hallittava, ymmärrettävä ja toimintavarma. Havaittavalla tarkoitetaan, että sisältö ja rakenne ovat käyttäjälle havaitsemiskelpoisia

millä tahansa aistielimellä, silmillä tai kuulolla. Eli kuurojen täytyy nähdä ja sokeiden kuulla. Hallittavalla tarkoitetaan, että käyttäjän tulee pystyä suorittamaan toimintoja ilman esteitä. Käyttöliittymällä ei pidä olla sellaisia toimintoja, joita käyttäjä ei pystyisi suorittamaan. Ymmärrettävällä tarkoitetaan, että sisällön ja käyttöliittymän hallittavuus tulee olla käyttäjälle selkeää ja loogista. Toimintavarma tarkoittaa, että käyttöliittymän sisältö tulee olla tulkittavissa useilla apuvälineillä, kuten erilaisilla ruudunlukijoilla.

Jokaisella periaatteella on ohjeistukset ja onnistumiskriteerit, jotka on jaettu 3 tasoon A, AA, AAA. Jokaisella tasolla on omat vaatimuksensa alimmasta tasosta vaativimpaan. A on perustaso, joka parantaa saavutettavuutta osalle käyttäjistä, joilla on erityisiä haasteita digipalveluiden käytössä. AA tason kriteeri parantavat entistää laajemmin. Tämä on kansainvälinen käytäntö ja kuuluu yleensä lainsäädännön piiriin. Myös Suomen laki velvoittaa noudattamaan vähintään AA-tasoa. AAA taso parantaa savutettavuutta entistä paremmin useammalle käyttäjälle, esimerkiksi äänen ja videon kääntäminen viittomakielelle.
WCAG ei kuitenkaan voi taata, että näiden ohjeistuksien noudattaminen tekee käyttöliittymästä täydellisesti saavutettavan, koska WCAG kriteerit ovat enneminkin ohjeita eivätkä valmiita ratkaisuja. Saavutettavuuden saavuttamiseksi sovelluksen koodin on oltava virheetöntä. Lisäksi nämä ohjeistukset ovat usein tulkinnanvaraisia, tällöin kehittäjät tai arvioijat eivät välttämättä pääse yksinmielisyyteen esimerkiksi mahdollisista teknistä ratkaisusta. (Aluehallintovirasto).

2.3 Tekninen saavutettavuus

Onnistumiskriteerin saavuttamiseksi sovelluksen ohjelmistokoodin on oltava semanttisesti oikein kirjoitettua, virheetöntä ja suunniteltu HTML-standardien ja WCAG-ohjeiden mukaisesti. Tämän kaiken tekee sovelluksen kehitystiimi. Tällaisia ratkaisuja ei kuitenkaan kehitetä tyhjästä, vaan tähän on olemassa erillinen ARIA-teknologia.
ARIA on merkintäkieleen lisätty erityinen attribuuttisarja (kuva 2), jota tukevat kaikki nykyaikaiset selaimet ja ruudunlukijat. Ensimmäinen versio ARIA-attribuuttisarjasta julkaistiin vuonna 2014. Näiden attribuuttien avulla merkintäkielen tiedot siirretään käyttöjärjestelmään alkuperäisten rajapintojen kautta ruudunlukijalle, jolloin saavutetaan sovellus, jonka kuurot näkevät ja sokeat kuulevat. ARIA:n avulla voidaan lisätä kuvauksia objekteihin, työkaluvihjeitä, navigointimerkkejä, virheilmoituksia ja paljon muuta.

```
<a role="button" href="#">Button</a>
```

Kuva 2. ARIA:n käyttö merkintäkielessä.

Tämä yksinkertainen esimerkki osoittaa, että sinänsä elementti on linkki koodissa, mutta sen visuaalinen toteutus käyttöliittymässä voikin hyvin olla painike. Tämä saattaa olla hämmentävää sokealle käyttäjälle. Mutta lisäämällä erityisen "role"-attribuuti ja sen arvoksi "button" ruudunlukijat lukevat elementin "painikkeena". (MDN web docs 2022).

ARIA-attribuuttien käytöstä koodissa on apua sokeiden käyttäjien lisäksi niille, joilla on motoriikkaa ongelmia kehon liikkuvuuden kanssa. Jotkut käyttäjät eivät pystyy hallita hiirtä fyysisesti ja luottavat vain näppäimistöön sivustolla navigoidessa. Täten käyttöliittymän sopeutuminen näppäimistökäyttöön on yksi tärkeimmistä näkökulmista saavutettavuudessa. Standardina tapana on liikkua sivulla TAB-näppäimellä. Kun TAB-näppäintä painetaan, kohdistus siirtyy aina seuraavaan interaktiiviseen elementtiin ja ruudunlukija lukee elementin sisällön siitä, mikä elementti on kulloinkin kohdistuksessa. Samalla näkevän käyttäjän pitäisi pystyä näkemään, missä kohdistus on. Yleensä tämä on musta tai sininen kehys elementin ympärillä. Yleisimmät selaimet on jo sopeutettu tähän toimintoon ja selaimet tarjoavat automaattisesti tarkennusilmaisimen, mutta selainversiosta riippuen kehys voi vaihdella. Siitä huolimatta tällainen toiminnallisuus riippuu kuitenkin merkintäkielestä ja edellä mainituista role-määritteistä. Selaimet lukevat merkinnän sellaisenaan, eivät sen visuaalisia elementtejä. Jotkut sivustojen elementeistä on toteutettu staattisina, mutta ovat todellisuudessa interaktiivisia. Tämä näkyy näkevälle käyttäjälle, mutta sokea käyttäjä ei tiedä siitä, ellei erityisiä attribuutteja lisätä elementtiin. Siksi on erittäin tärkeää suunnitella koodi oikein, jotta näppäimistön käyttö on mahdollista kaikissa interaktiivisissa elementeissä. (Web AIM 2020).

Siitä huolimatta ARIA ei kuitenkaan ratkaise kaikkia saavutettavuusongelmia. ARIA-attribuutit ovat erittäin tehokas työkalu, joka auttaa välittämään tietoa loppukäyttäjälle, mutta on olemassa muitakin ongelmia, kuten värisokeus. Värisokea ihminen näkee, mutta ei pystyy erottaa punaista vihreästä. Tällaisissa tapauksissa voi syntyä käyttökokemukseen liittyviä ongelmia, joissa esimerkiksi vihreä painike tarkoittaa "kyllä" ja punainen painike "ei". Tai katsoessa kaaviota "piirakkadiagrammi" muodossa, käyttäjä näkee vain yksitoikkoisen kaavion. Tämän voidaan korjata käyttämällä erilaista kontrastia viereisten elementtien kanssa. Kontrastin ja värien käyttö on saavutettavuuden kannalta elintärkeää. Huonokontrastiset elementit voivat olla vaikeita nähdä myös käyttäjille, joilla ei ole mitään rajoitteita. Peruskäyttäjien lisäksi myös näkövammaisten käyttäjien on pystyttävä havaitsemaan sivun sisältö. Kontrastin tarkistamiseksi on olemassa erilaisia työkaluja, kuten kontrastin tarkistin "Contrast Checker". (WebAIM).

Kuva 3. Kontrasti elementin ja taustan välillä.

Kuvassa 3 on esitetty kolmen tekstielementin kontrastitasot. Vain elementeissä 2 ja 3 kontrastitaso täyttää AA- onnistumiskriteerit. Taso A ei vaadi kontrastitasoa ollenkaan. WCAG ohjeistuksen mukaisesti kontrastitaso määräytyy taustavärin ja elementin värin eron perusteella. Esimerkiksi valkoinen teksti valkoisella taustalla olisi 1:1 suhteessa ja musta teksti valkoisella taustalla 21:1 suhteessa teksti- ja tausta- väri. Taso AA vaatii kontrastisuhteen vähintään 4,5:1 pienelle tekstille ja 3:1 suurelle tekstille. WCAG 2.1 vaatii vähintään 3:1 kontrastisuhteen grafiikoille ja käyttöliittymäkomponenteille (kuten lomakkeiden syöttökentille). WCAG:n kolmas taso AAA vaatii kontrastisuhteen vähintään 7:1 normaalille tekstille ja 4,5:1 suurelle tekstille.
(WebAIM 2021).

2.4 Yhteenveto

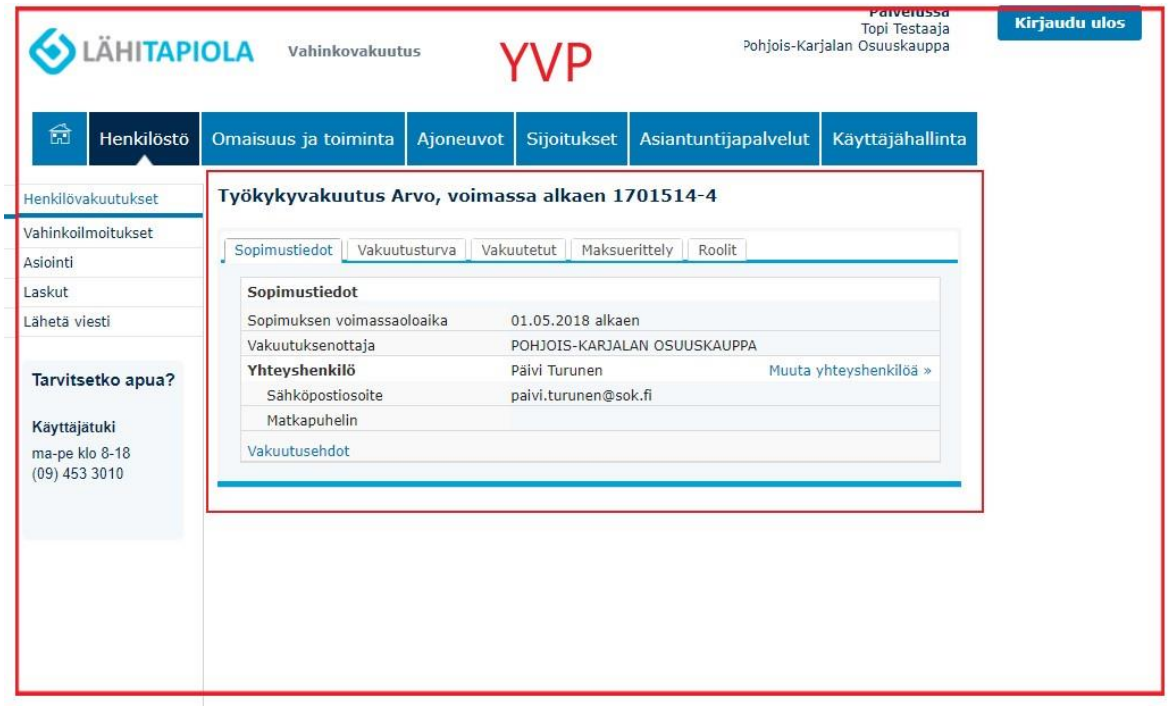
Saavutettavuusdirektiivi on melko tuore, joten suurin osa käyttöliittymistä ei edelleenkään täytä WCAG-tason kriteerejä vuonna 2022. On olemassa erityisiä ARIA-työkaluja elementtien mukauttamiseen tähän standardiin. Näiden onnistumiskriteerien saavuttamiseksi on välttämätöntä suunnitella koodi semanttisesti oikein, lisätä tarvittavat ARIA-attribuutit WCAG ohjeistuksen mukaisesti ja asettaa tekstielementtien kontrasti taso vähintään 4,5:1.

3 Verkkosivuprojekti

3.1 Tavoite ja projektisuunnitelma

Tämän tutkimuksen puitteissa teen saavutettavuus muutokset vanhalle Lähitapiolan sovellukselle nimeltään "YVP". Tämä sovellus on peräisin 90-luvun puolivälistä ja se sisältää muita erillisiä itsenäisiä sovelluksia. Koska kyseessä on finanssiorganisaation sovellus, kuuluu Lähitapiolan web ympäristö saavutettavuuslain piiriin.

YVP yhtenä sovelluksena on kooltaan valtava. Se koostuu kolmestakymmenestä muusta sovelluksesta, ja sen saavutettavuuden sopeuttaminen voi kestää yli vuoden. Siksi käsittelen tämän tutkimuksen puitteissa vain yhden sen alisovelluksista, "Työkykyvakuutus Arvon" (lyhennettynä työkyky), joka toimii itsenäisenä osana upotettuna YVP-sovellukseen.



Kuva 4. Työkyky -alisovellus upotettuna YVP-sovellukseen.

Tässä tutkimuksessa tunnistan saavutettavuuspuutteita ja sopeutan käyttöliittymän WCAG 2.1 A – AA standardiin. Tulen arvioimaan ja kuvailemaan sovelluksessa käytettyjä teoreettisia ja teknisiä ratkaisuita sovelluksen jokaiseen yksittäiseen komponenttiin tapauskohtaisesti, jotta tutkimuksen rakenne olisi loogisempi. Arvioimiseksi ja saavutettavuus puutteiden tunnistamiseksi tulen käyttämään seuraavia Windows 10 -käyttöjärjestelmän sovelluksia:

- NVDA-ruudunlukuohjelma (v. 2020.4)
- Google Chrome-selain (v. 80.0)
- Firefox-selain (v. 74.0)
- Eclipse Luna-ohjelmointiympäristö

Muut ohjelmat ja työkalut:

- WebAIM – kontrastin tarkistus työkalu
- Wave – saavutettavuustarkastelu työkalu

NVDA on erikoistyökalu, joka lukee tiedot ruudusta. Tätä työkalua käyttävät myös näkövammaiset käyttäjät sovelluksia käyttäessään.

Wave on Chrome laajennus, jonka avulla voidaan arvioida sivun saavutettavuutta selaimessa. Tämä auttaa minua tunnistamaan sivuvirheet suoraan selaimen kehittäjätyökaluista.

WebAim on työkalu, jonka avulla voidaan tarkistaa kontrastin taso.

Tässä tutkimuksessa ei käydä läpi, miten käytetään Eclipse-, Wave-, NVDA- ja kehittäjätyökaluja selaimessa. Kaikki huomio keskittyy vain saavutettavuuden puutteiden tunnistamiseen ja niiden ratkaisujen toteutukseen.

Ensisijaisesti tarkistan sovelluksessa käytetyt tekniikat. Tämä on tarpeellista käytettyjen teknologioiden arvioimiseksi, niiden mahdollisuuksien ja rajoitusten tunnistamiseksi WCAG:ssa määriteltyjen kriteerien suhteen. Tämän jälkeen siirryn varsinaiseen toteuttamiseen.

Tämän tehtävän tarkoituksena on tutkia sovelluksen sivut, tunnistaa WCAG-standardiin liittyvät havainnot ja antaa teoreettiset ohjeet korjaamiseen ja toteuttaa niitä. Komponentteja on yhteensä viisi. Puutteiden tunnistaminen edellyttää NVDA:n käyttöä, jotta voidaan tunnistaa, kuinka tietty elementti luetaan sokealle käyttäjälle, sekä "Wave"-laajennusta, joka mahdollista sivun tutkimisen mahdollisten ristiriitaisuuksien varalta WCAG-periaatteiden kanssa. Toteuttamista ja kehittämistä varten tulen käyttämään Eclipse- kehitysympäristöä.

Seuraavassa luvussa pystytän työympäristön paikallisella tietokoneella ja aloitan käytettyjen tekniikoiden tutkimisen, sen jälkeen analysoin käyttöliittymän saavutettavuuden tason ja pyrin löytämään ratkaisuja.

3.2 Käytetyt teknologiat

Työkyky-alisovelluksen analysoimiseksi käytetään seuraavia tekniikoita:

- jQuery (v. 1.11)
- PrimeFaces (v. 6.0)
- Java Servlet Faces (v. 2.1)
- Java 6

JSF on Java -ohjelmistokehys. Sen tärkein merkitys on yksinkertaistaa ja nopeuttaa oliopohjaisesti suuntautuneiden verkkosovellusten kehittämistä Javalla. Tässä sovelluksessa JSF käyttää XHTML-tekniikan tietojen esittämiseen käyttöliittymälle. Tämä on suuri etu, koska sen syntaksi on hyvin samanlainen kuin HTML:n syntaksi, ja siksi helppo ylläpitää. JSF 2.1 on kuitenkin melko vanha tekniikka, jonka kyseinen versio kehitettiin vuonna 2010. ARIA-attribuuttien tuki versiossa 2.1 on todella heikko, enimmäkseen olematon. (Oracle).

ARIA-attribuuttien ongelman ratkaisee osittain PrimeFaces-kirjasto niissä paikoissa, joissa kyseessä olevaa komponenttia käytetään. PrimeFaces 6.0 julkaistiin vuonna 2016, ja se tukee jo ARIA-attribuutteja. Primefaces-kirjasto itsessään on JSF-lisäkirjasto ja sen komponentteja voidaan kutsua suoraan XHTML-tiedostoista. (PrimeFaces User Guide 2016).

jQuery on JavaScript-kirjasto, jonka avulla voidaan helposti manipuloida DOM-elementtejä.

Jotta ARIA-tuki saavutetaan JSF:ssä, on ylikirjoitettava render- luokaat tai päivitettävä ohjelmointikehys uudempaan versioon, jossa tuki näille attribuuteille on jo saatavilla. Ohjelmointikehyksen päivittäminen edellyttää muiden riippuvuuksien päivittämistä. Tämä voi vaikuttaa muutoksiin myös muissa YVP sovelluksissa, mikä tekee sen toteutuksesta melko työlästä. Kirjastoa on kuitenkin mahdollista ylikirjoittaa luomalla sovelluksen konfiguraatiotiedostoihin uuden tag-tunnisteen, jonka Java käsittelee ja renderöi haluttuun muotoon. Tämän lisäksi on tarpeellista luoda uusia Java luokkia, jotka peritään vakiokirjastosta, ylikirjoittaa metodit ja lisätä metodeihin tarvittavat lisäattribuutit. On kuitenkin syytä olla tekemättä tätä tarpeettomasti, koska myös tämä prosessi on työläs. (Oracle 2013).

Toinen vaihtoehto on tehdä muutokset varsinaiseen DOM-elementtiin. DOM on ohjelmointirajapinta. Silloin kun selain on tehnyt pyynnön palvelimelle, se vastaanottaa palvelimesta HTML-koodin ja luo sen perusteella DOM-puun jonka jälkeen selain renderöi sivun. JavaScriptin avulla voidaan tehdä muutoksia jo renderöityyn DOM-elementtiin. Tämä olisi helpoin ja turvallisin vaihtoehto siinä mielessä että vaikutus tule olemaan renderöidyyn

DOM-puun mutta ei back-end koodistoon ja sen mahdolliset haitat ovat vähäiset sovelluksen ympäristöön.

3.3 Toteutus

3.3.1 Menetelmät

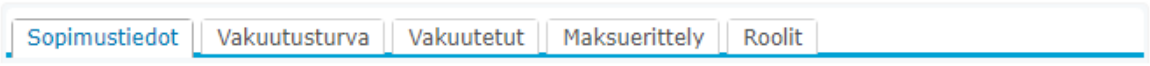
Ensisijaisesti jokainen sivu on tarkistettava Wave- työkalulla. Wave mahdollistaa mahdollisten puutteiden havaitsemisen, kuvaa ja luokittelee tarkemmin mihin WCAG ohjeistuksiin kyseinen havainto kuuluu. Tämä menetelmä helpottaa huomattavasti sivun analysointia, kun käydä läpi jokainen elementti erikseen. Kuitenkaan Wave ei osaa tunnistaa kaikkia havaintoja. Esimerkiksi näppäimistön käyttö, joka vaati käyttäjältä suoraa vuorovaikutusta päätelaitteen kanssa. Täten joissain tilanteessa joudutaan tutkimaan erikseen joitakin elementtejä ja tunnistaa käsin mahdolliset havainnot tutkimalla HTML- koodia.

Korjausehdotukset tehdään WCAG sekä ARIA- määreiden puutteiden perusteella HTML koodissa. Ei ole olemassa ainoaa oikeata teknistä ratkaisua siitä, miten saavutettavuutta voidaan saavuttaa, vaan jokainen ratkaisu on uniikki ja vaihtoehtoja voi olla monta. Tärkeintä on, että ratkaisulla saavutetaan WCAG standardi.

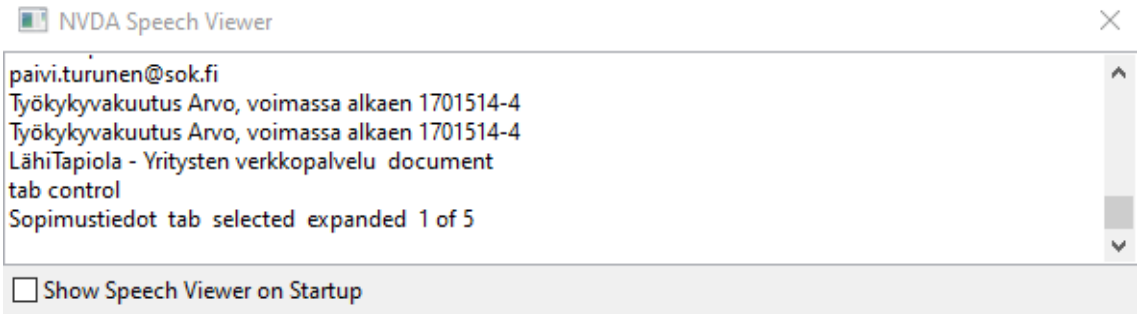
3.3.2 Yleiset havainnot

Yleisessä katselmoinnissa käy ilmi, että saavutettavuuden taso on melko heikko. Jotkut perustoiminnot puuttuvat kokonaan. Osa toiminnoista on kuitenkin toteutettu hyvällä tasolla, esimerkiksi näppäimistön käyttö ja näppämistöfokukset sekä taulukot ovat enimmäkseen toteutettu saavutettavaksi, mutta suurimmaksi osaksi löytyy paljon puutteita. Osa virheistä on havaittu YVP-kehyksessä, ”työkyky”- sovelluksen ulkopuolella. Esimerkiksi välilehtiä vaihdettaessa sivun otsikko ei vaihdu eikä käyttäjälle ilmoiteta, että välilehti on vaihtunut (2.4.2, Taso A, Sivuotsikot), pääsisältöön ei ole hyppylinkkiä (2.4.1, Taso A, Ohita lohkot), ja HTML koodista puuttuu ”<lang>” tunniste, joka määritä sivun kielen (3.1.1, Taso A, Sivun kieli). (W3C 2016).

3.3.3 Välilehdet



Kuva 5. Välilehdet.



Kuva 6. Välilehti NVDA-ruudunlukijalla luettuna.

Välilehdet eivät ole kovin loogisia ruudunlukijalla luettuna. Kuvassa 6 on esitetty välilehti-rakenne ruudunlukijalla luettuna, jossa luetaan kaksi attribuuttia. Ensimmäinen arvo (selected) kertoo, että välilehti on valittu, ja toinen arvo (expanded) onko välilehti auki vai kiinni. Auki/kiinni- ilmaisu viittaa haitarirakenteisiin, joten tämän attribuutin käyttö voi antaa väärä kuvan sivun rakenteesta. Kun kyseessä on välilehdet, tulisi tässä kohdassa käyttää ainoastaan ”selected” attribuuttia (4.1.2 Taso A Nimi ja rooli). (W3C 2016).

```
▼<li class="ui-state-default ui-tabs-selected ui-state-active ui-corner-top" role="tab" aria-
expanded="true" aria-selected="true" aria-label tabindex="0" style>
```

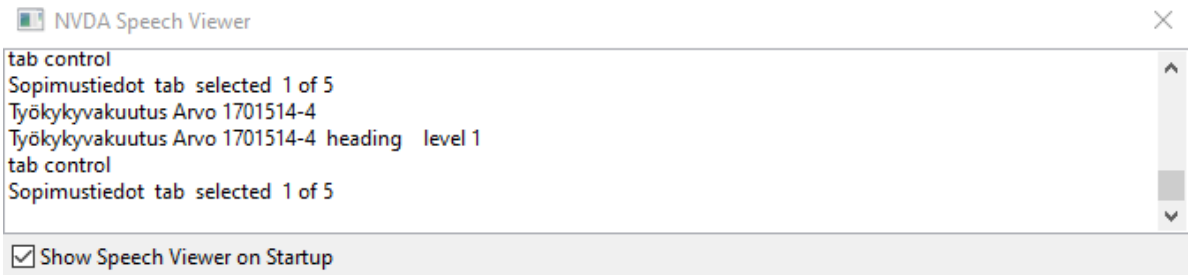
Kuva 7. Välilehti. Elementin lähdekoodi.

Kuten kuvassa 7 on esitetty, koodissa on ylimääräinen "aria-expanded" -attribuutti, joka on poistettava kaikista tämän komponentin elementeistä. Tämän on PrimeFaces -kirjaston valmis komponentti, joten jouduin ylikirjoittamaan kyseisen koodin tavoitteen saavuttamiseksi.

```
show: function () {
  this.alignPanel();
  this.panel.show();
  this.visible = true;
  this.trigger.attr("aria-expanded", true);
  this.closer.trigger("focus");
},
```

Kuva 8. PrimeFaces kirjaston funktio.

Kuvassa 8 näkyy, että ”show”-funktio luo uuden attribuutin ”aria-expanded” aina, kun sitä kutsutaan. Tämä ominaisuus on poistettava.



Kuva 9. Välilehti NVDA luettuna. Toteutuksen tulos.

Sivun välilehtivalikko ei toimi WCAG:n mukaisesti. Käyttäjä pääse liikkumaan tabulaattorilla-näppäimellä välilehtien välillä. Tämä toimintalogiikka on väärä. Välilehtivalikkoon pitäisi päästä vain kerran sisään tai ulos TAB:n avulla, mutta ei liikkua niiden välillä. Välilehtien välillä tulisi liikkua nuolinäppäimellä (2.1.1, Taso A, Näppäimistökäyttö). (W3C 2016) (W3C 2019).

Ratkaisu tähän olisi JavaScript funktio, joka kuuntele näppäinpainalluksia ja suorittaa tarvitsemamme toiminnot vain kyseisen elementin puitteissa.

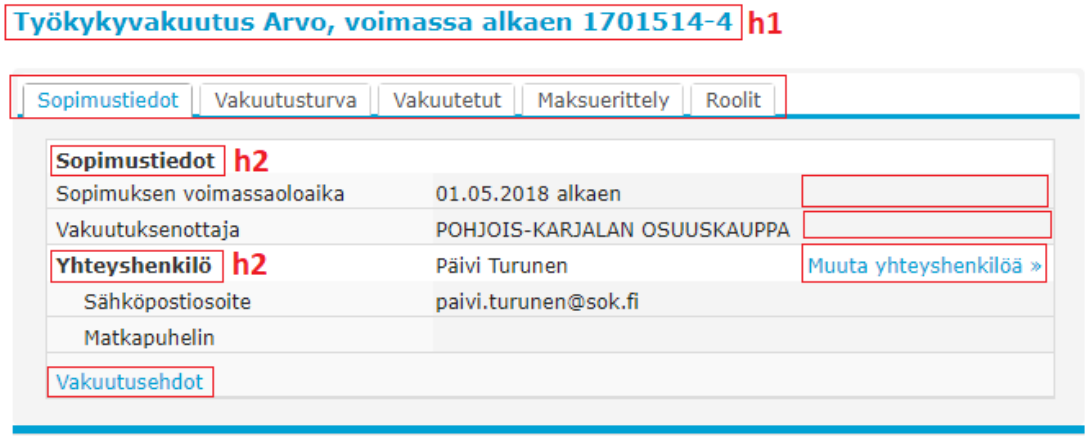
Kuvassa 10 on esitetty WCAG mukainen näppäimistökäytön toteutus käyttäen JavaScript funktiota.

```
$('li.ui-corner-top').keydown(function(e) {
  switch (e.keyCode)
  {
  case 13: $(this).click(); break;
  case 37: e.preventDefault(); $(this).prev("li").focus(); break;
  case 39: e.preventDefault(); $(this).next("li").focus(); break;
  }

  if (e.which === 9 && !e.shiftKey) {
   e.preventDefault();
   $(".ui-tabs-panels").find(':focusable').first().focus();
  }
  if (e.which === 9 && e.shiftKey) {
      e.preventDefault();
      $("#sideLinksForm").find(':focusable').last().focus();
    }
})
```

Kuva 10. Välilehdet. Näppäimistökäyttö funktio.

3.3.4 Sopimustiedot

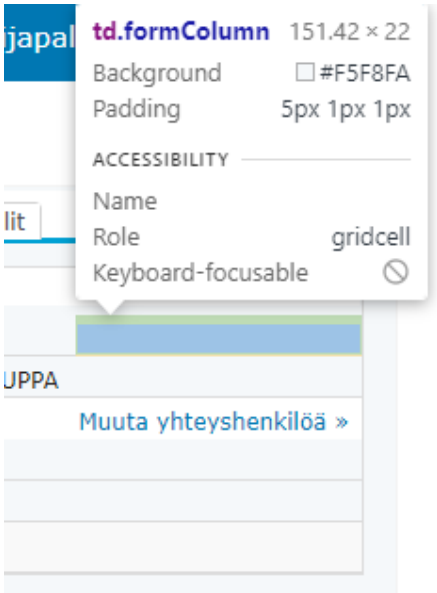


Kuva 11. Sopimustiedot välilehti. Havainnot.

Vaaleasinisten tekstien kontrastin taso ei ole riittävä suhteessa taustaan. Taso 3.26:1. WCAG edellyttää vähimmäistason arvoon 4.5:1 (1.4.3, Taso AA, Kontrasti). (W3C 2016).

Sinitekstien väristä saadaan riittävää, jos tekstin väri asetetaan #0077B3. Tämä muutos on globaali ja vaikuttaa koko sovellukseen.

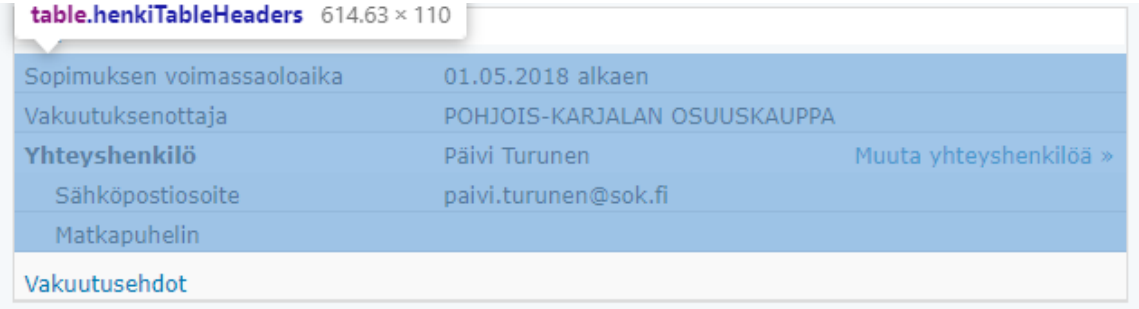
”Muuta yhteyshenkilöä” -linkin ylläpuolelle on tarpeettomasti laitettu tyhjät sarakkeet, jotka eivät ole informatiivisia ruudunlukijan käyttäjälle. Tyhjät sarakkeet on poistettava tai piilotettava ruudunlukijalta koska ruudunlukija lukee myös nämä tyhjät sarakkeet (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).



Kuva 12. Sopimustiedot välilehti. Tyhjä sarake.

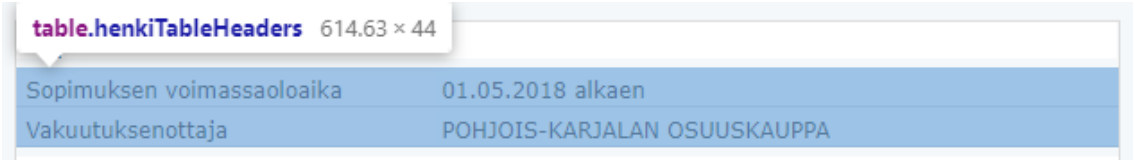
Tyhjät sarakkeet on tehty sivun muotoilua varten <td> tägillä. Kuten kuvassa 12 näkyy, ”Muuta yhteyshenkilö” -linkki myös käyttää kolmannetta saraketta. Tässä tapauksessa ”Muuta yhteyshenkilöä” -linkki pitää viedä taulukon ulkopuolelle, jotta taulukon kolmas rivi vapautuu sisällöstä, jonka jälkeen voidaan poistaa koko rivi.

”Sopimustiedot” ja ”Yhteystiedot” -sarakkeiden alla oleva sisältö on syytä laittaa erilliseen taulukkoon. Tällä hetkellä ”Yhteyshenkilö” osio sijaitsee ”Sopimustiedot” taulukon sisällä, joka on semanttisesti väärin (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

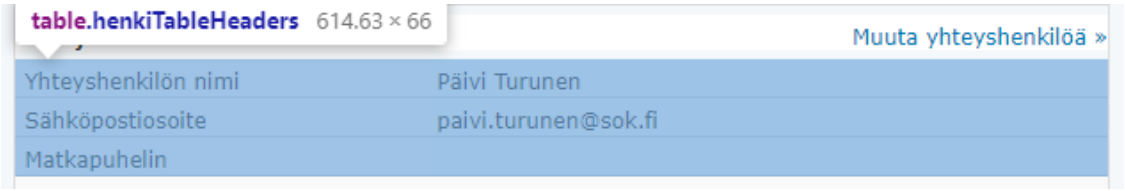


Kuva 13. Sopimustiedot välilehti. Taulukkorakenne.

Sivun koko sisältö on laitettu yhteen ”table” elementin sisälle, joka on XHTML:n natiivi elementti. Korjaus onnistuu yksinkertaisesti jakamalla taulukkoa kahteen erilliseen taulukkoon.

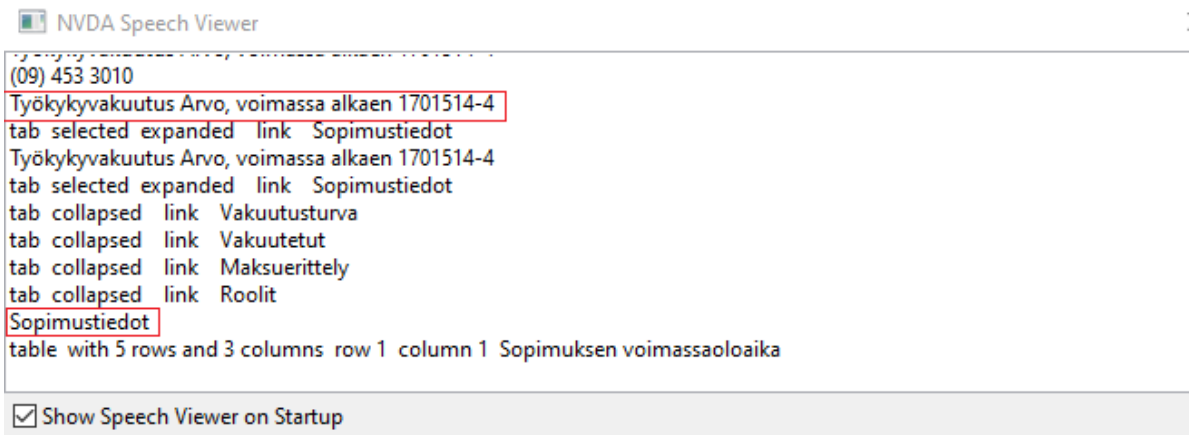


Kuva 14. Sopimustiedot välilehti. Taulukkorakenne 1 korjattuna.



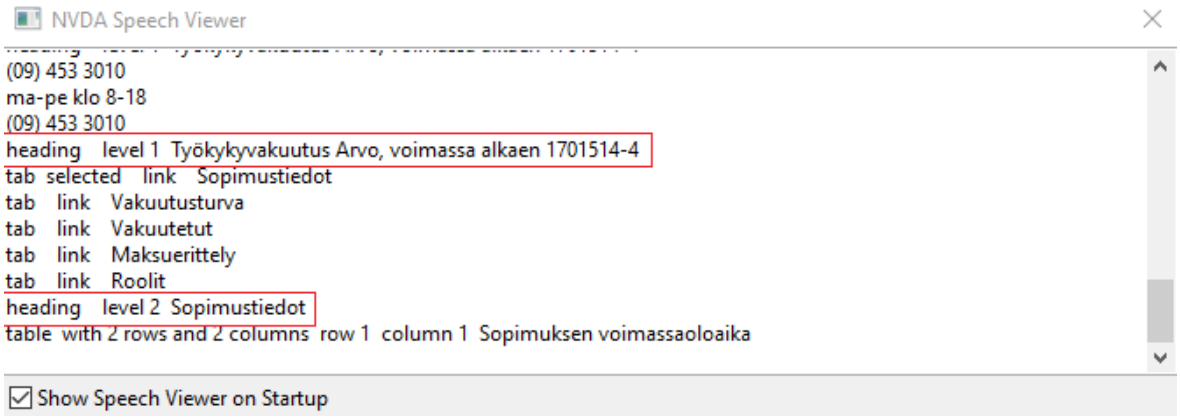
Kuva 15. Sopimustiedot välilehti. Taulukkorakenne 2 korjattuna.

Sivulla puuttuu H1-H6 otsikot. Korostetut tekstit kuvassa 11 viittaavat otsikkoihin. NVDA lukee nämä tekstit normaalitekstinä eikä tiedotta käyttäjää, että kyseessä on otsikko (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).



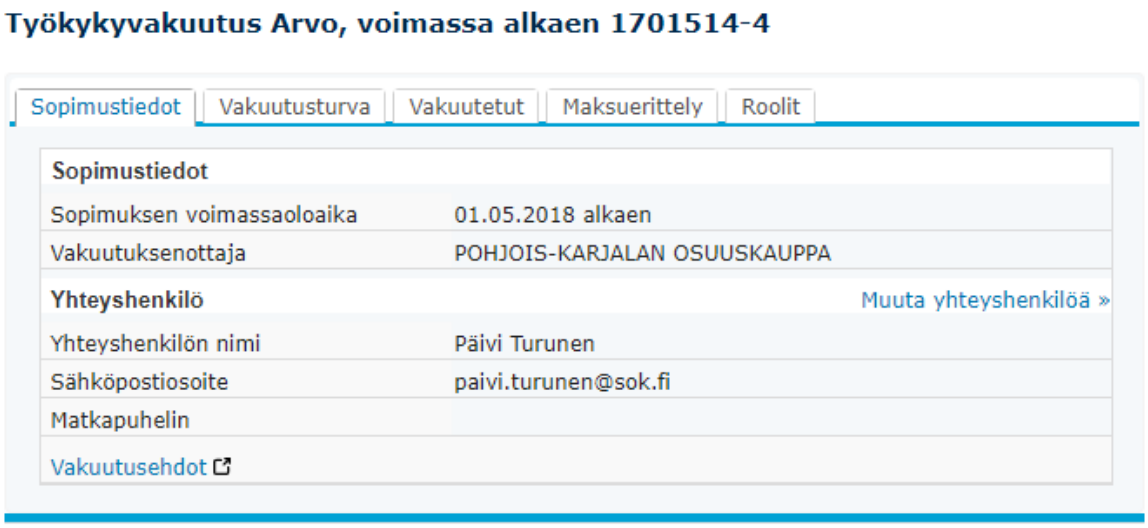
Kuva 16. Otsikot NVDA:lla luettuna.

Tämä onnistuu muuttamalla tekstit h1, h2 -elementiksi. H -otsikoissa on oma hierarkia, joka erittäin tärkeä ottaa huomioon, täten pääotsikko kuulu h1 tasolle. ”Sopimustiedot” ja ”Yhteyshenkilö” kuuluvat h2 tasolle koska molemmat ovat samantasoisia alaotsikoita.



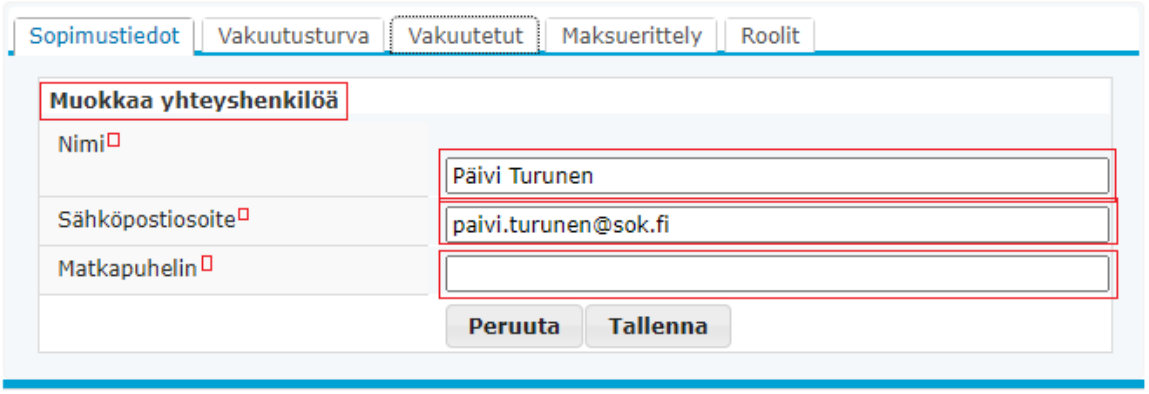
Kuva 17. Sopimustiedot välilehti. Otsikkotasot luetaan WCAG mukaisesti.

Sivulla on ulosvievä linkki ”Vakuutusehdot", joka avautuu uudelle välilehdelle. Ruudunlukijaa käyttävälle saattaa olla vaikea hahmottaa linkin merkitys sen osalta ja päätellä, että mennäkö siihen vai ei. Linkillä tulisi olla ikoni, jonka tarkoitus olisi kertoa käyttäjälle, että linkki avautuu uudelle välilehdelle. Samalla ikonille tulisi lisätä alt teksti, joka kertoo ruudunlukijaa käyttävälle sen merkityksestä (2.4.4, Taso A, Linkin tarkoitus kontekstissa). (W3C 2016).



Kuva 18. Sopimustiedot. Lopputulos.

3.3.5 Muokkaa yhteyshenkilöä



Kuva 19. Muokkaa yhteyshenkilöä.

Syöttökenttiä ei ole merkitty saavutettavuuden kannalta pakollisiksi, vaikka ovat ohjelmallisesti pakollisia. Nimilapuista puuttuu asteriski, joka indikoi pakollisuuden (3.3.2, Taso A, Nimilaput tai ohjeet). (W3C 2016).

Kenttiin on lisättävä ”aria-required” attribuutti, joka kertoo käyttäjälle NVDA:n välityksellä kentän pakollisuudesta. Nimilapun viereen laitetaan asteriski. Asteriski piilotetaan ruudunlukijalta sen informatiivisuuden puuteen vuoksi ruudunlukijakäyttäjälle.

Syöttökentillä voi olla useita ARIA-attribuutteja. Aria-label, aria-required, aria-labelledby ja aria-describedby joita jatkossa tulen käyttämään. Täten on syytä luoda tuki kaikille attribuuteille kerralla kaikissa syöttökentissä, joka täten toimisi koko sovelluksen puitteissa.

Uusien ARIA-attribuuttien tukemiseksi ensin pitää lisätä uusi tagi- tunniste JSP konfiguraatioon ja viittaus uuteen Java luokkaan. Seuraavassa vaiheessa täytyy ylikirjoittaa syöttökentän render- luokka ja määritä uudet attribuutit. Kutsumalla uuden tagi- tunnisteen XHTML koodista, sovellus renderöi uuden komponentin, jossa on mukana uudet ARIA-attribuutit.

Kuvassa 20 on esitetty ylikirjoitettu luokka, joka renderöi uusi syöttökenttä ARIA attribuuttien tuella.

```
public class InputRenderer extends HtmlTextRendererBase {

    @Override
    protected void renderInputEnd(FacesContext facesContext, UIComponent component) throws IOException {
        Map<String,String> additionalAttributes = new HashMap<String, String>();

        String placeholder = (String) component.getAttributes().get("placeholder");
        String maxDate = (String) component.getAttributes().get("maxDate");
        String minDate = (String) component.getAttributes().get("minDate");
        String ariaLabel = (String)component.getAttributes().get("aria-label");
        String ariaLabelledby = (String)component.getAttributes().get("aria-labelledby");
        String ariaDescribedby = (String)component.getAttributes().get("aria-describedby");
        Boolean required = (Boolean)component.getAttributes().get("required");

        if(placeholder != null) { additionalAttributes.put("placeholder", placeholder); }
        if(maxDate != null) { additionalAttributes.put("data-maxdate", maxDate); }
        if(minDate != null) { additionalAttributes.put("data-mindate", minDate); }

        if(ariaLabel != null) { additionalAttributes.put("aria-label", ariaLabel);}
        if(ariaLabelledby != null) { additionalAttributes.put("aria-labelledby", ariaLabelledby);}
        if(ariaDescribedby != null) { additionalAttributes.put("aria-describedby", ariaDescribedby);}
        if(required) { additionalAttributes.put("aria-required", "true");}

        ResponseWriter writer = facesContext.getResponseWriter();

        for(Map.Entry<String, String> entry : additionalAttributes.entrySet()){
            writer.writeAttribute(entry.getKey(), entry.getValue(), null);
        }
        super.renderInputEnd(facesContext, component);
    }
```

Kuva 20. Muokkaa yhteyshenkilöä. Syöttökentissä käytettävä InputRender -Java luokka.

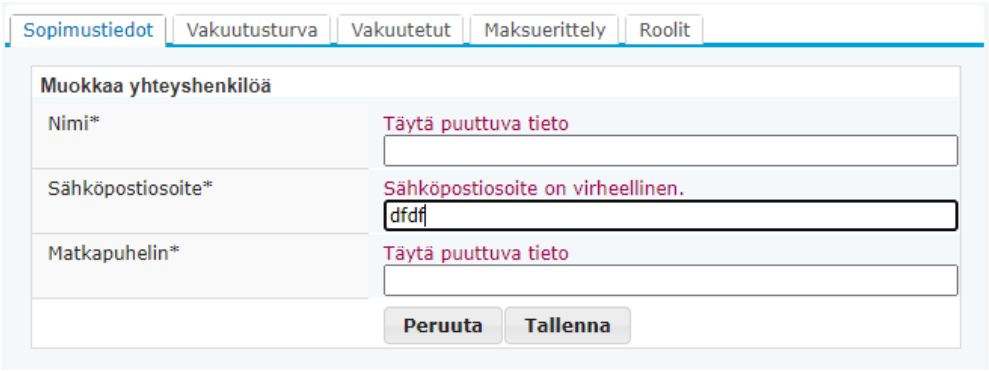
Asteriskit voidaan yksinkertaisesti lisätä XHTML tiedostoon ja laittaa piiloon ruudunlukijalta, koska niiden tarkoitus on kertoa näkevälle käyttäjälle syöttökentän pakollisuudesta, mutta eivät sinänsä ole informatiivisia ruudunlukijakäyttäjälle. Piilottaminen tehdään lisäämällä elementtiin ”aria-hidden” attribuutti.

Ruudunlukija ei osaa lukea syöttökenttien nimilappuja. Nimilaput tulee kiinnittää syöttökenttiin (2.5.3, Taso A, Nimilappu nimessä). (W3C 2016).

Ratkaisussa voidaan käyttää joko ”aria-label”- tai aria-labelledby- attribuutteja, jotta nimilaput saadaan luettavaksi ruudunlukijalle. Näiden kahden attribuutin välinen ero on se, että aria-label välittää ruudunlukijalle kyseisen attribuutin arvon, kun taas aria-labelledby välittää sen elementin arvon, jonka id on asetettu aria-labelledby-attribuutin arvoksi. Var-

mempi tapa olisi käyttää aria-labelledby-attribuuttia, sillä jos nimilapun arvo muuttuu tulevaisuudessa, se muuttuu automaattisesti myös näytönlukijalle. Eli lisätään id-attribuutti nimilappuelementtiin ja aria-labelledby syöttökenttiin.

Virhetilanteessa puuttuu aria-invalid määre elementistä (3.3.1, Taso A, Virheen tunnistaminen). (W3C 2016).



Kuva 21. Muokkaa yhteyshenkilöä. Virhetilanne.

Aria-invalid tulisi kertoa ruudunlukijalle, että kyseinen elementti on viallinen. Tässä tapauksessa voidaan lisätä ehdollinen aria-invalid määre JavaScript:llä tapauskohtaisesti.

Tämän funktion kirjoittaminen vaatii yksinkertaista logiikkaa. Jokaiseen virhetekstiin on lisättävä errorTextData-luokka indikoimaan elementtiä. Sen jälkeen tulee kirjoittaa funktio, joka etsii kaikki ne elementit, joihin tämä luokka kuuluu. Kun elementti löytyy, funktio lisää aria-invalid määreen sen lähimpään syöttökenttään.

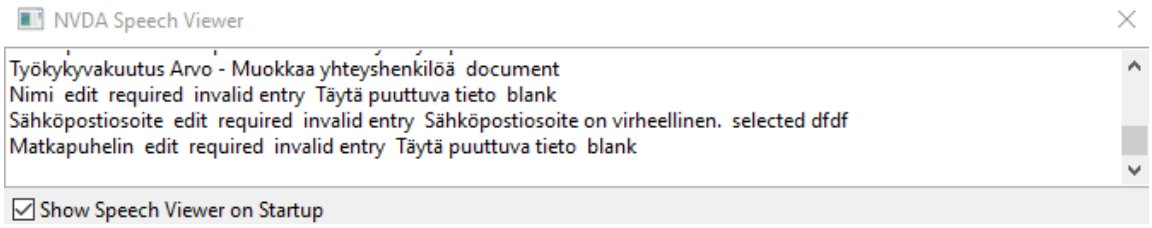
Tämä JavaScript funktio, kuten kaikki muutkin, on kirjoitettava erilliseen tiedostoon, jotta ne olisivat saatavilla koko sovelluksessa ja vaikutus olisi globaali sovelluksen puitteissa. Jatkossa, jos samat puuteet löytyvät, ei tarvitse tehdä, muuta kuin lisätä errorTextData oikeaan paikkaan.

```
$("td").each(function() {
  var thisE = $(this);
  var span = thisE.find("span");
  if(span.hasClass("errorTextData")){
      thisE.find("input").attr("aria-invalid", "true");
  }
});
```

Kuva 22. Muokkaa yhteyshenkilöä. Aria-invalid lisäys.

Ruudunlukija ei myöskään lue itse virhetekstiä, koodista puuttuu automaattinen tarkennus virheelliseen elementtiin (3.3.3, Taso AA, Virhe ehdotus). (W3C 2016).

Jotta ruudunlukijalle kuuluisi myös yksilölliset virhetekstit pitää lisätä aria-described-määre syöttökenttäelementtiin ja asettaa sen arvoksi virhetekstin id, jolla identifioidaan tietty syöttökenttäelementti.



Kuva 23. Muokkaa yhteyshenkilöä. Virhelilanne NVDA:lla luettuna.

Automaattista tarkennusta voidaan saavuttaa lisäämällä koodia, joka etsii ja fokusoi ensimmäiseen elementtiin, jolla on aria-invalid arvo. Tämä ratkaisu toimii jatkossa koko sovelluksen puitteissa.

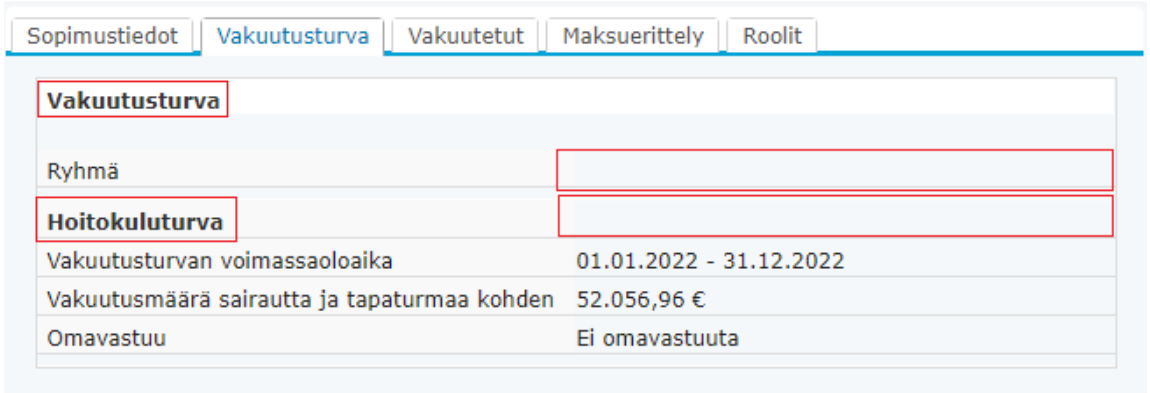
```
$('.yvpTemplateBody').find('*[aria-invalid="true"]:eq(0)').focus();
```

Kuva 24. Muokkaa yhteyshenkilöä. Automaattinen tarkennus.

Sivulla korostetut ”Muokkaa yhteyshenkilöä”- teksti viittaa otsikkoon. NVDA lukee tämän tekstin normaalitekstinä eikä tiedota käyttäjää, että kyseessä on otsikko (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Korostettu teksti muokataan h2 elementiksi laittamalla se <h2> tagiin sisään.

3.3.6 Vakuutusturva



Kuva 25. Vakuutusturva.

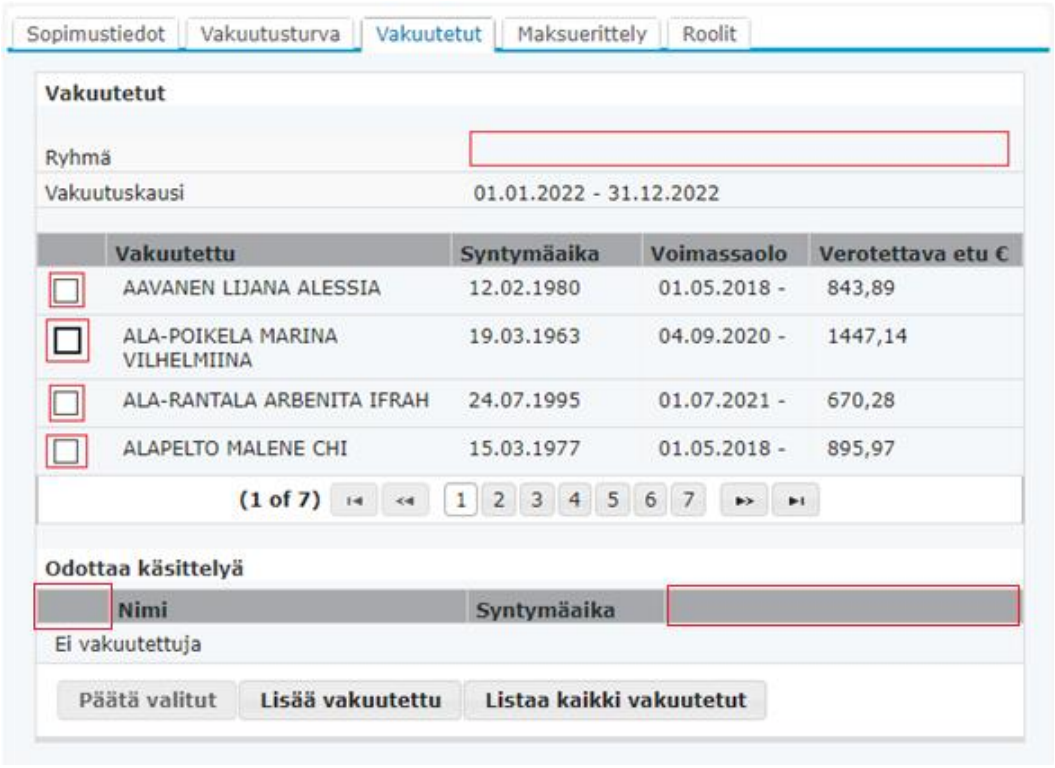
Sivulla korostetut ”Vakuutusturva” ja ”Hoitoturva”- teksti viittaa otsikkoon. NVDA lukee tämän teksti normaalitekstinä eikä tiedota käyttäjää, että kyseessä on otsikko (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Korostettu teksti muokataan h2 elementiksi laittamalla sen <h2> elementtiin sisään.

Sivulla löytyy tyhjät sarakkeet, joita luetaan ruudunlukijalla, joita pitää poistaa (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Toteuttaminen onnistuu muokkaamalla XHTML tiedoston natiivi elementtejä.

3.3.7 Vakuutetut



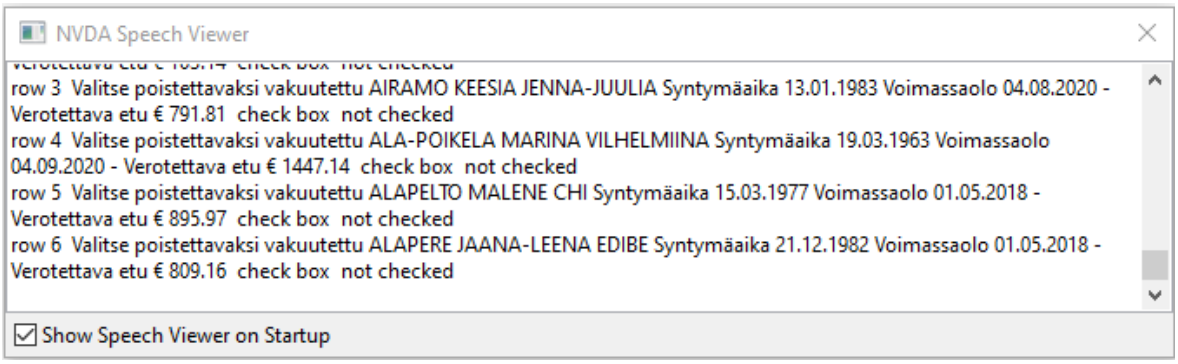
Kuva 26. Vakuutetut välilehti.

Taulukossa on tarpeettomia tyhjiä sarakkeita (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Sarakkeen poistaminen onnistuu yksinkertaisesti poistamalla td elementti XHTML tiedostosta.

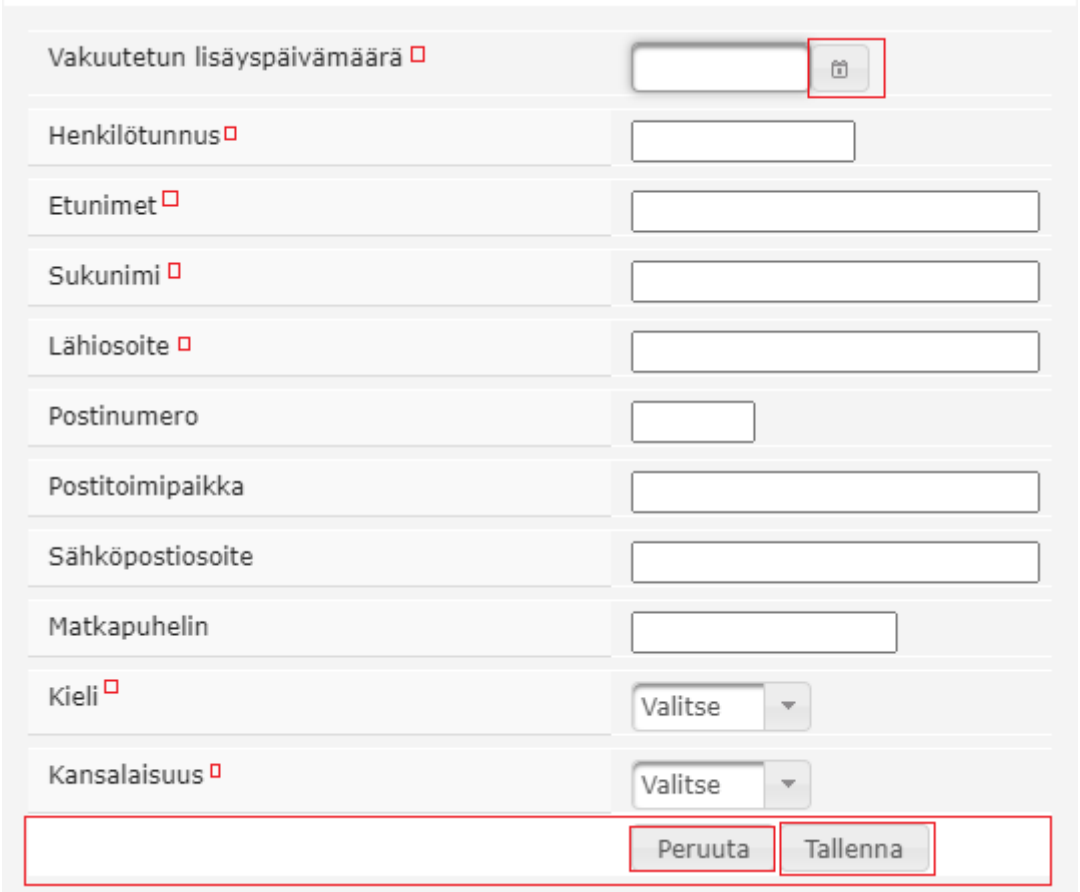
Valintaruudussa puuttuu nimilaput. Ruudunlukijakäyttäjä ei pystyy hahmottaa mihin valittu valintaruutu kuuluu (3.3.2, Taso A, Nimilaput tai ohjeet). (W3C 2016).

Tämä on konfiguroitava erikoistaulukko joka renderöi tarpeen mukaisesti myös ”checkbox” ruudut sekä navigaatio. Tämän taulukon renderöi PrimeFaces-kirjasto, jolla on jo ARIA-tuki. Taulukolla on erikoisattribuutti ariaRowLabel, joka ottaa vastaan mitä tahansa tietoa ja toimii aria-label periaatteella. Jotta WCAG edellytys täyttyy, ariaRowLabel- attribuuttiin tulee laittaa vihjeteksti; ”taulukon nimilaput ja vakuutetun tiedot”.



Kuva 27. Vakuutetut. Taulukko NVDA luettuna.

3.3.8 Lisää vakuutettu



Kuva 28. Lisää vakuutettu modaali ikkuna.

Painikkeet on laitettu tarpeettomasti taulukkoon (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Painikkeet voidaan irrottaa pois taulukosta ja muotoilla CSS avulla.

Painikkeiden tekstien kontrastin taso ei ole riittävä suhteessa taustaan (1.4.3, Taso AA, Kontrasti). (W3C 2016).

Painikkeiden tekstiväri on muutettava mustaksi CSS:n avulla.

Nimilaput eivät ole luettavissa ruudunlukijalla. Syöttökentin nimilaput tulisi kuulua ruudunlukijalla luettuna (3.3.2, Taso A, Nimilaput tai ohjeet). (W3C 2016).

Luvussa 4.3.4 olen esittänyt miten ylikirjoitetaan syöttökentän render komponentti, jonka vaikutus on globaali. Täten tämän havainnon ratkaisemiseksi riittää vain lisätä aria-labelledby- määre ja sen arvoksi asettaa nimilapun id.

Lomakkeesta löytyy pakolliset syöttökentät mutta tätä ei ole indikoitu millään tavalla (3.3.2, Taso A, Nimilaput tai ohjeet). (W3C 2016).

Tämä voidaan korjata lisäämällä asteriski jokaiseen nimilapun viereen ja piilottamalla se ruudunlukijalta ”aria-hidden” määreellä. Syöttökenttäelementtiin lisätään ”required”-määre. Täten sovellus renderöi sen kuten "aria-required" elementin.

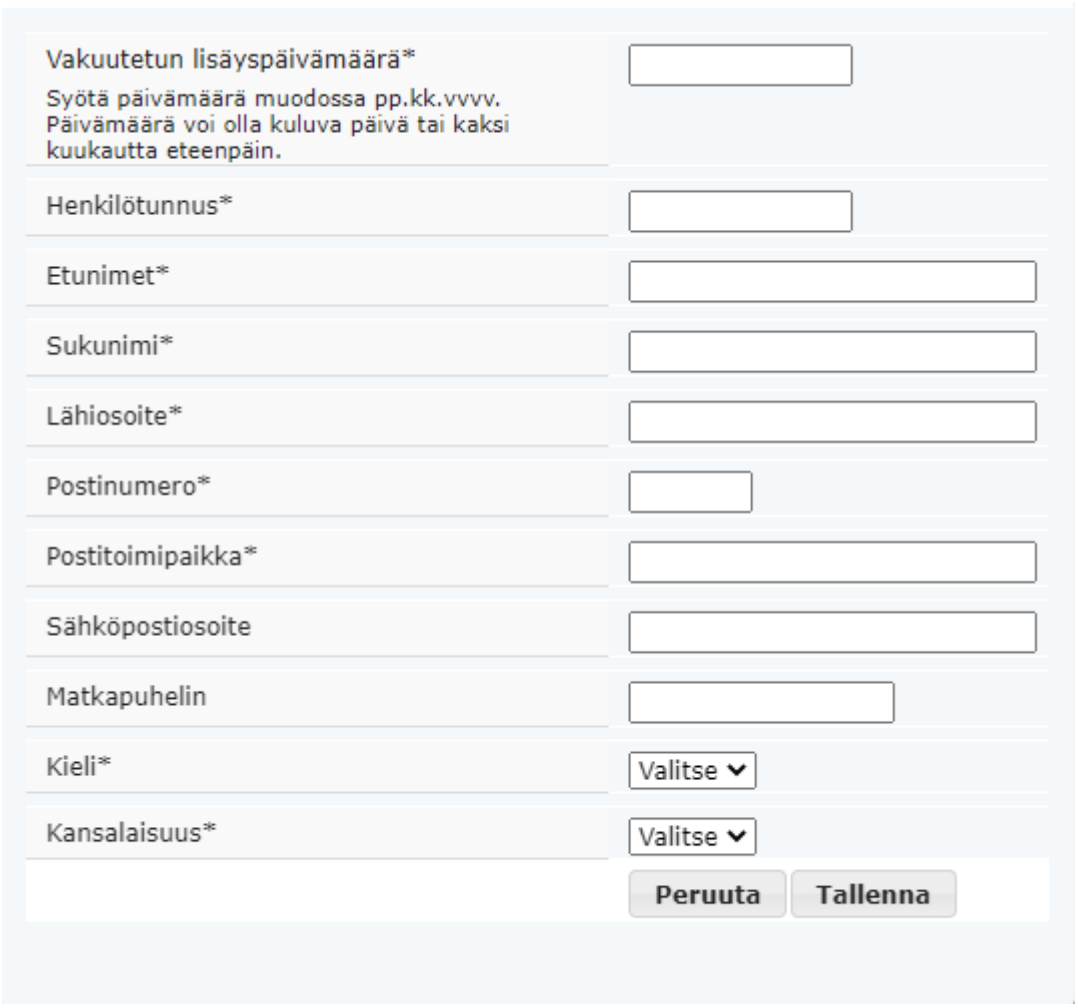
Virhetilanteessa puuttuu ”aria-invalid” määre elementissä (3.3.1, Taso A, Virheen tunnistaminen). Ruudunlukija ei lue virhetekstiä, sovelluksesta puuttuu automaattinen tarkennus virheelliseen elementtiin (3.3.3, Taso AA, Virhe ehdotus). (W3C 2016).

Luvussa 4.3.4 on jo käyty läpi virheitä käsittelevä funktio. Täten näiden ongelmien ratkaisemiseksi riittää vain errorTextData-luokan lisääminen virhetekstiin, ja aria-described-määre syöttökenttään asettamalla sen arvoksi virhetekstin id.

Kalenterin käyttö on täysin mahdoton näppäimistöllä (2.1.1 Taso A, Näppäimistökäyttö). (W3C 2016).

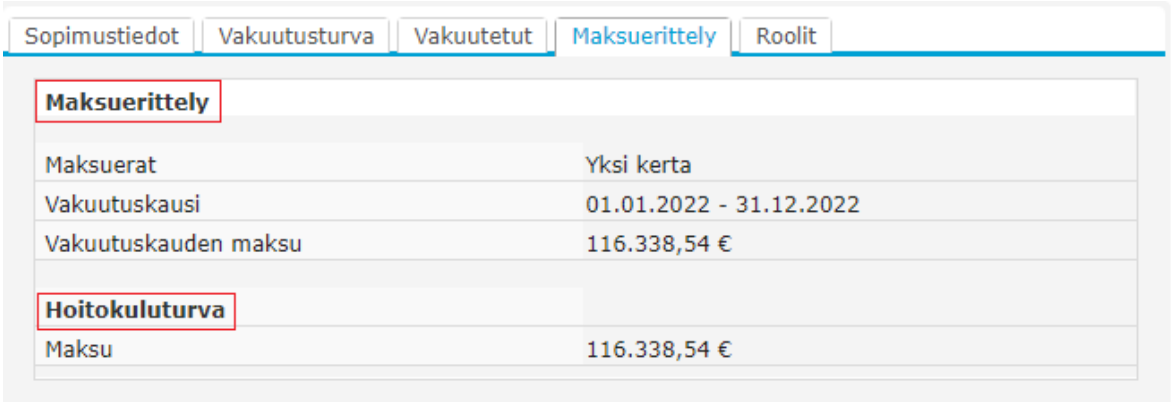
Kalenteri kuuluu PrimeFaces kirjastoon, joka on sinänsä monimutkainen elementti, ettei sitä kannata yrittää sellaisenaan korjata. Helpoin tapa olisi integroida sovellukseen joku muu saavutettava kalenteri. Mutta tässä tapauksessa en lähde tekemään sitä johtuen siitä seuraavasta työmäärästä. Sen sijaan harkitsen toista vaihtoehtoa.

Ratkaisuvaihtoehtona kalenteri elementti voidaan irrottaa ja jättää sovellukseen pelkästään syöttökenttä, mihin lisätään ohjeet syöttömuodosta. Tämä ratkaisu täyttää WCAG edellytykset (3.3.2, Taso A, Nimilaput tai ohjeet). (W3C 2016).



Kuva 29. Lisää vakuutettu. Toteutuksen tulos.

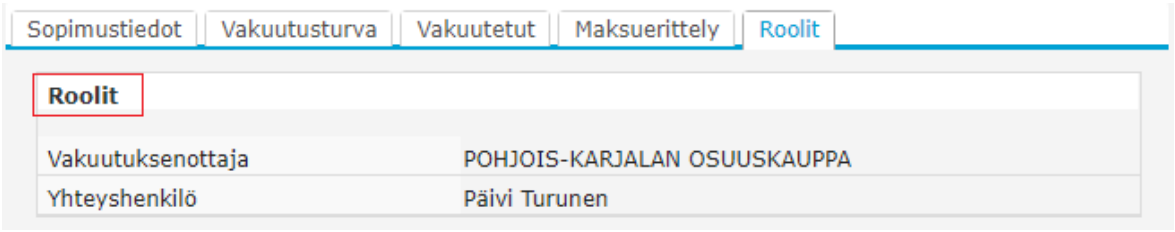
3.3.9 Maksuerittely



Sivulla korostetut ”Maksuerittely” ja ”Hoitoturva”- tekstit viittaavat otsikoihin. NVDA lukee nämä tekstit normaalitekstinä eikä tiedottaa käyttäjää, että kyseessä on otsikko (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Korostettu teksti muokataan h2 elementiksi laittamalla sen <h2> elementtiin sisään.

3.3.10 Roolit



Korostettu teksti ”Rooli”- teksti viittaa otsikkoon. NVDA lukee tämän tekstin normaalitekstinä eikä tiedota käyttäjää, että kyseessä on otsikko (1.3.1 Taso A, Informaatio ja suhteet). (W3C 2016).

Muokataan teksti h2 elementiksi laittamalla sen <h2> elementtiin sisään.

3.4 Tuotos

Sivuilla oli havaittu toistuvia saavutettavuus puutteita, jotka liittyivät erityisesti syöttökenttiin ja tietojen esittämiseen sekä niiden suhteisiin. Osa puutteista on korjattu käyttämällä alkuperäisiä HTML-tageja staattisissa XHTML-elementeissä, muut ongelmat on korjattu ylikirjoittamalla renderöintikirjastoja tai muokkaamalla DOM elementtejä JavaScript funktioilla.

Sovelluksesta on korjattu otsikot, näppäimistökäyttö, kontrastit, kenttien nimilaput, kenttien pakollisuus, virheiden käsittely ja poistettu tyhjät elementit, jotka eivät olleet informatiivisia ruudunlukijakäyttäjälle.

Merkittävää visuaalista muutosta sovellukseen ei ole seurannut näiden muutoksien myötä, ja niitä on vaikea nähdä paljaalla silmällä. Kuitenkin ruudunlukijakäyttäjälle nämä muutokset ovat suuri parannus. Ruudunlukijakäyttäjät voivat nyt näiden muutosten myötä olla vuorovaikutuksessa entistä tehokkaammin käyttöliittymän kanssa.

4 Pohdinta

Tämän opinnäytetyön tavoite on ollut selvittää, mitä on saavutettavuus ja mitä se pitää sisällään. Sen lisäksi tavoitteena on ollut näyttää miten teknistä saavutettavuutta toteutetaan käyttöliittymissä.

Saavutettavuuden tavoitteena on, että käyttöliittymä on kaikkien käyttäjien käytettävissä rajoituksista huolimatta. Saavutettavuuden saavuttamiseksi on olemassa WCAG-standardi, joka sisältää ohjeistuksen, jota noudattamalla sovellus saatetaan saavutettavaksi. Lainsäädäntö edellyttää ainakin WCAG-tason AA saavutettavuusstandardien noudattamista sovelluksissa, joiden palvelut liittyvät julkiseen sektoriin ja rahoituslaitoksiin. Tämän standardin noudattaminen ohjelmointityössä tekee käyttöliittymästä paitsi rajoitteisille käyttäjille saavutettavan, myös parantaa sen käyttöä tavallisille käyttäjille.

Tässä opinnäytetyössä esiteltiin useampia menetelmistä saavutettavuuden kehittämiseen. Olen oppinut mitä WCAG-standardi on ja miten sitä sovelletaan käytännössä. On huomattava, että WCAG on ensisijaisesti teoria, joka ei sisällä teknisiä ratkaisuja. Se tekee tulkinnasta hyvin henkilökohtaisen, mikä voi johtaa hämmennykseen tilanteessa, kun ohjelmoija arvioi havainnot. Itse toteutus on ohjelmoijan vastuulla, joten on tärkeää, että ohjelmoija tulkitsee WCAG-ohjeistukset oikein. Siitä huolimatta onnistuin tässä työssä saattamaan käyttöliittymän WCAG-standardien mukaiseksi huolimatta sovelluksessa käytetyistä vanhentuneista teknologioista, jotka eivät tue nykyaikaista tekniikkaa.

Saavutettavuuden saavuttamiseksi ei tarvitse luoda paljoa koodia tai uutta ohjelmistoa. Suurin osa ohjelmistoista on jo luotu. Lähinnä sitä voidaan kutsua sopeutumiseksi WCAG-standardeihin eikä uusien ohjelmistojen kehittämiseen. Riittää, kun ymmärretään ja noudetaan ohjeistusta ja lisätään tarvittavat määreet koodiin. Tämä voidaan saavuttaa useilla eri tavoilla millä tahansa ohjelmointikielellä.

Saavutettavuuden tekninen toteutus tässä työssä toteutettiin yksinomaan tämän sovelluksen puitteissa, ei yleismaailmallisina ratkaisuina. Suurin osa muutoksista tehtiin jo renderöityyn DOM-elementtiin JavaScriptin avulla. Pieni osa muutoksista on tehty Java-koodiin. Nämä ratkaisut toimivat kaikissa JSF-pohjaisissa sovelluksissa, jotka käyttävät PrimeFaces-kirjastoa, eikä niiden pitäisi vaatia lisäsäätöjä.

Ohjelmoimalla JavaScript funktioita yritin saavuttaa yleismaailmalliseen ja uudelleen käytettävissä olevan ratkaisun tämän sovelluksen puitteissa. Käytännössä tämä tarkoittaa, että riittää, kun lisää koodiin tarvittavat attribuutit, jotta JavaScript toimii tämän elementin parametrien kanssa ilman tarpeettomia riippuvuuksia tietystä elementistä tai lisäkoodin

kirjoittamista. Tämä lähestymistapa helpottaa saavutettavuuden kehittämistä tuotteen jatkokehityksessä ja vähentää virheiden riskiä, jos jokin tunnus tai elementtijärjestys muuttuu. Tämä JavaScript-avusteinen lähestymistapa ei ole paras, koska se ylikuormittaa selainta, sekä on lisäosa sovellukseen ja toimii asynkronisesti sovelluksen kanssa, mikä voi johtaa virheisiin ainakin teoriassa. JavaScriptin avulla voidaan kuitenkin manipuloida tietoja erittäin joustavasti. Tämän projektin yhteydessä, jossa aika ja resurssit olivat rajattuja, tämä lähestymistapa oli optimaalisin, sillä se oli nopein ja edullisin vaihtoehdoista resurssien suhteen. Paras vaihtoehto olisi päivittää sovelluksessa käytetyn ohjelmointikielen ja kirjastojen versiot uudempiin paremman lopputuloksen saavuttamiseksi, mutta tässä sovelluksessa ja siihen liittyvään ympäristöön tämä olisi melko työläs prosessi. Kyseessä oli erittäin suuri ja vanha sovellus. Teknologioiden päivittäminen uusiin versioihin voitaisiin rinnastaa koodin uudelleen kirjoittamiseen monimutkaisuuden suhteen.

Asteikolla; huono, tyydyttävä tau erinomainen, olen mielestäni saavuttanut tavoitteet tyydyttävällä tasolla. Kuten lopputulokset osoittavat WCAG onnistumiskriteerit täyttyivät ja verkkopalvelun käyttöliittymä on entistä saavutettavampi. Kuitenkin vaihtoehtoja toteuttamiseen olisi ollut useita erilaisia. Toteutukseni oli vain yksi mahdollisista vaihtoehdoista.

Tätä työtä voidaan eskaloida tutkimalla sovelluksessa käytetyitä tekniikoita tarkemmin, ja kehittämällä jälleen uutta ratkaisua Java koodiin. Luomalla uudet Java- ratkaisut, joka korvaavat alkuperäiset luokat voidaan mahdollistaa yleismallisempi ratkaisu Java- sovelluksiin. Täten voitaisiin saavuttaa varmempia tuloksia toteutuksessa, kuin JavaScript funktiot.

Tätä työtä tehdessä olen oppinut saavutettavuuden keskeiset käsitteet ja siihen liittyvät toteutustekniikat. Sen lisäksi olen perehtynyt hyvin JSF- kehyksen toimintalogiikkaan sekä erityisesti elementtien renderöinti luokkiin, JavaScriptiin ja testaus- työkaluihin, joita käytin tässä työssä.

Haasteisin osio tässä työssä oli WCAG kriteerien tulkinta. Usein oli vaikea ymmärtää abstraktinen merkitys WCAG- ohjeistuksissa. WCAG- ohjeistukset voivat olla melko laajoja ja epäselviä. Varsinkin luvussa 3.3.3 näppäimistön toimintalogiikkaan liittyvä ongelma ei käynyt heti ilmi WCAG- ohjeistuksessa. Oikea näppäimistön toimintalogiikka on elementtikohtainen, joka on tutkittava tapauskohtaisesti.

Lähteet

Aluehallintovirasto. Digipalvelulain vaatimukset. Luettavissa: https://www.saavutettavuus-vaatimukset.fi/digipalvelulain-vaatimukset. Luettu 21.11.2021.

Aluehallintovirasto. Kenelle saavutettavuus on tärkeää? Luettavissa: https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/kenelle-saavutettavuus-on-tarkeaa. Luettu. 21.11.2021.

Aluehallintovirasto. Tietoa WCAG-ohjeistuksesta. Luettavissa: https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/tietoa-wcag-kriteereista. Luettu 08.12.2021.

Aluehallintovirasto. Yleistä saavutettavuudesta. Luettavissa: https://www.saavutettavuus-vaatimukset.fi/yleista-saavutettavuudesta. Luettu 20.11.2021.

MDN Web docs 2021. ARIA. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA. Luettu: 15.12.2021.

Näkövammaisten liitto 2020. Näkövammaisuus Suomessa. Luettavissa: https://www.nkl.fi/fi/nakovammaisuus-suomessa. Luettu 21.11.2021.

Openclassrooms 2022. Make Your Web Content Accessible. Luettavissa: https://open-classrooms.com/en/courses/6663451-make-your-web-content-accessible/6912083-get-to-know-the-web-content-accessibility-guidelines-wcag. Luettu 07.12.2021.

Oracle. Tag inputText. Luettavissa: https://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/h/inputText.html. Luettu 07.01.2022.
Oracle 2013. The Java EE 6 Tutorial. Luettavissa: https://docs.oracle.com/javaee/6/tutorial/doc/bnavg.html. Luettu 07.01.2022.

PrimeFaces User Guide 2016. WAI-ARIA. Luettavissa: https://www.primefaces.org/docs/guide/primefaces_user_guide_6_0.pdf#page=628&zoom=100,76,92. Luettu 07.01.2022.

Saavutettavuusdirekriivi. Saavutettavuusdirektiivi edistää yhdenvertaisuutta. Luettavissa: https://saavutettavuusdirektiivi.fi. Luettu: 21.11.2021.

W3C 2008. Web Content Accessibility Guidelines (WCAG) 2.0. Luettavissa: https://www.w3.org/TR/WCAG20/. Luettu 07.12.2021.

W3C 2016. Info and Relationship. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/content-structure-separation-programmatic.html. Luettu 12.01.2022.

W3C 2016. Page Titled. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-title.html. Luettu 12.01.2022.

W3C 2016. Bypass Blocks. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-skip.html. Luettu 12.01.2022.

W3C 2016. Language of Page. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/meaning-doc-lang-id.html. Luettu 14.01.2022.

W3C 2016. Name, Role, Value. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/ensure-compat-rsv.html. Luettu 14.01.2022.

W3C 2016. Keyboard. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation-keyboard-operable.html. Luettu 14.01.2022.

W3C 2016. Contrast (Minimum). Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html. Luettu 14.01.2022.

W3C 2016. Link Purpose (In Context). Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-refs.html Luettu 16.01.2022.

W3C 2016. Error identification. Luettavissa: https://www.w3.org/TR/UNDERSTANDING-WCAG20/minimize-error-identified.html Luettu 18.01.2022.

W3C 2019. Example of Tabs with Automatic Activation. Luettavissa: https://www.w3.org/TR/wai-aria-practices-1.1/examples/tabs/tabs-1/tabs.html. Luettu 18.01.2022.

W3C 2021. WCAG 2 Overview. Luettavissa: https://www.w3.org/WAI/standards-guidelines/wcag. Luettu 25.11.2021.

WebAIM 2020. Keyboard Accessibility. Luettavissa: https://webaim.org/techniques/keyboard. Luettu: 15.12.2021.

WebAIM 2021. The WebAIM Million. Luettavissa: https://webaim.org/projects/million. Luettu 21.11.2022.

WebAIM. Contrast Checker. Luettavissa: https://webaim.org/resources/contrastchecker/. Luettu: 15.12.2021.

WebAIM. Contrast and Color Accessibility. Luettavissa: https://webaim.org/articles/contrast/. Luettu: 15.12.2021.

WHO 2021. Disability and health. Luettavissa: https://www.who.int/en/news-room/fact-sheets/detail/disability-and-health. Luettu 20.11.2021.