Roope Kivioja

Puheen ongelmista kärsiville tarkoitetun kommunikointisovelluksen toteuttaminen Ionic-kehyksellä

Tietotekniikan pro gradu -tutkielma

19. huhtikuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Roope Kivioja

Yhteystiedot: roope.kivioja@gmail.com

Ohjaajat: Jonne Itkonen ja Jukka-Pekka Santanen

Työn nimi: Puheen ongelmista kärsiville tarkoitetun kommunikointisovelluksen toteuttaminen Ionic-kehyksellä

Title in English: Puheen ongelmista kärsiville tarkoitetun kommunikointisovelluksen toteuttaminen Ionic-kehyksellä

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 33+1

Tiivistelmä: Tutkielman tiivistelmä on tyypillisesti lyhyt esitys, jossa kerrotaan tutkielman taustoista, tavoitteesta, tutkimusmenetelmistä, saavutetuista tuloksista, tulosten tulkinnasta ja johtopäätöksistä. Tiivistelmän tulee olla niin lyhyt, että se, englanninkielinen abstrakti ja muut metatiedot mahtuvat kaikki samalle sivulle.

Avainsanat: Ionic, AAC, WWW-sovellukset, käytettävyys

Abstract: Tiivistelmä englanniksi.

Keywords: Ionic, AAC, web applications, usability

Termiluettelo

Ionic	Ohjelmistokehys.

Kuviot

Sisältö

1 Johdanto

// Ohjepituus 5 sivua. (Varmaan oikeasti 1-3 sivua.)

// TODO: Tähän tulee johdanto.

2 Kommunikointisovelluksen rakentamiseen tarvittavia taustatietoja ja menetelmiä

// Ohjepituus 10-15 sivua.

// TODO: Lyhyt siltaava kappale johdannosta teoriakappaleeseen

// TODO: kirjoita omaksi kappaleeksi:

- Taustat, tarpeet ja tavoitteet:

- progressiivisten WWW-sovellusten tutkiminen

- progressiivisten WWW-sovellusten sopiminen avusteisen kommunikaation mobiilisovelluksien tuottamiseen

-> tavoite: Ionic-kehyksen toiminnan tutkiminen

-> tavoite: Ionic-kehyksen avulla tehdyn avusteisen kommunikaation sovelluksen vertaaminen ohjelmistoteknisiin ja käytettävyyspohjaisiin suosituksiin kehittäjän näkökulmasta

-> tavoite: Ionic-kehyksen avulla tehdyn avusteisen kommunikaation sovelluksen soveltuminen autistien kätettävyystarpeisiin suosituksien pohjalta

2.1 Avusteinen kommunikaatio

On olemassa useita eri sairauksia ja kehityshäiriöitä, joiden johdosta henkilön kyky muodostaa puhetta voi hetkellisesti tai pysyvästi heikentyä. Puheen muodostamisen ongelmista kärsivä henkilö saattaa joutua turvautumaan arkipäivän kommunikaatiossa erilaisiin apuvälineisiin kommunikoidakseen muiden ihmisten kanssa. Yleisesti näitä viestintämenetelmiä kuvaamaan tarkoitettu termi on **puhetta tukeva ja korvaava kommunikaatio** (engl. *Augmentative and Alternative Communication, lyh. AAC*).

Puhetta tukeva ja korvaava kommunikaatio voidaan jakaa kahtia: avustamattomaan ja avusteiseen. **Avustamattomalla puhetta tukevalla ja korvaavalla kommunikaatiolla** tarkoite-

taan kommunikaatiota, jossa ei tarvita apuvälineitä. Viittomakieli on yksi esimerkki avustamattomasta puhetta tukevasta ja korvaavasta kommunikaatiosta, mutta myös ihmisen elekieltä voidaan pitää avustamattomana puhetta tukevana ja korvaavana kommunikaationa.

Avustettu puhetta tukeva ja korvaava kommunikaatio tarkoittaa puolestaan kommunikaatiota, jossa käytetään jotain apuvälinettä. Apuvälineenä voi olla esimerkiksi valokuvia, kommunikaatiotaulu tai elektroninen laite. Tämän perusteella avustettu puhetta tukevaa ja korvaava kommunikaatio voidaan jakaa vielä eteenpäin kahdeksi ryhmäksi: matalan teknologian ja korkean teknologian puhetta tukevaan ja korvaavaan kommunikaatioon. Käyttäjä ei välttämättä käytä vain yhtä edellä mainituista tyypeistä. (Sigafoos ja Drasgow 2001)

Puhetta tukevaa ja korvaavaa kommunikaatiota voidaan tarjota puheen muodostamisen ongelmista kärsiville myös tietoteknisten sovellusten avulla, joiden kautta käyttäjä kirjoittaa joko suoraan tekstiä tai kommunikoi valitsemalla symboleja. Esimerkiksi autistisilla käyttäjillä on tyypillisesti hyvin henkilö- ja yksityiskohtaisia tarpeita puhetta tukevalle ja korvaavalle kommunikaatiolle, joten tietoteknisten sovellusten suhteellisen helppo muokattavuus puoltaa niiden käyttöä perinteisempien puhetta ja kommunikointia korvaavien apuvälineiden sijaan.

// TODO: korttipohjaiset järjestelmät

Puhetta tukevat ja korvaavat kommunikointisovellukset käyttävät yleisimmin symboleja. Autisteille tyypillistä on vahva visuaalis-avaruudellinen hahmotuskyky, joten tutkimuksen mukaan piirroksiin ja valokuviin liitetyt merkitykset ovat tälle ryhmälle luontevin tapa kommunikoida. Vaughnin ja Hornerin tutkimuksessa (Vaughn ja Horner 1995) Karl-nimisen autistisen koehenkilön haastava käytös ja aggressio vähenivät, kun pelkästään verbaalisesti annettujen ruokavaihtoehtojen rinnalle tuotiin kuvat ruoka-annoksista. Symbolipohjaiselle puhetta tukevalle ja korvaavalle kommunikaatiolle on siis sekä tutkimuspohjaista näyttöä että käytännön kokemuksiin perustuvaa kannustetta.

Puhetta tukevaan ja korvaavaan kommunikointisovellukseen voidaan liittää myös symboleita lukeva ääniominaisuus. Ääniominaisuus mahdollistaa kommunikoinnin näköyhteyden ulkopuolelle, vähentää symbolien tulkitsijan läsnäolon pakollisuutta ja helpottaa pidempien viestien rakentamista puhetta tukevassa ja korvaavassa kommunikointisovelluksessa. (Nunes 2008)

// TODO: ankkuroi edellinen kappale paremmin tekstiin

Esimerkiksi ääniominaisuuden lisäämisellä avustetusta puhetta tukevasta ja korvaavasta kommunikaatiosta voidaan tehdä **monimodaalista**. Monimodaalisuus tarkoittaa usean eri kommunikaatiotavan kautta tapahtuvaa kommunikaatiota. Monimodaalinen kommunikaatio voi tapahtua eri tapojen kautta yhtäaikaisesti tai peräkkäin. Puhetta tukevasta ja korvaavasta kommunikaatiosta kannattaa tehdä monimodaalista useista eri syistä. Ensinnäkin, suurin osa kaikesta kommunikaatiosta on monimodaalista, sillä toiselle ihmiselle puhuttaessa on tavallista selkeyttää sanomaa ilmein ja elein. Toiseksi, puhetta tukevaa ja korvaavaa kommunikaatiota käyttävä henkilön tulee useasti kommunikoida muiden puhetta tukevaa ja korvaavaa kommunikaatiota käyttävien henkilöiden kanssa, jolloin vaihtoehdoista on hyötyä. Kolmas merkittävä syy on se, että eri puhetta tukevissa ja korvaavissa kommuunikaatiotavoissa on vahvuuksia ja heikkouksia, joten eri kommunikaatiotapoja sekoittamalla voidaan korvata yksittäisen kommunikaatiotavan heikkouksia. (Sigafoos ja Drasgow 2001)

// TODO: mitä ongelmia

2.2 Progressiiviset WWW-sovellukset

Progressiiviset sovellukset (engl. *Progressive Web Applications, lyh. PWAs*) ovat selaimessa ajettavia WWW-sovelluksia, joiden ulkoasu määrittyy alustakohtaisesti niin, että niiden ulkoasu on mahdollisimman yhdenmukainen laitteen natiivien sovellusten kanssa. Selainpohjaisuuden takia progressiiviset sovellukset pystyvät käyttämään tarjolla olevia sovellusympäristön resursseja joustavasti sen sijaan, että ne itse määrittäisivät vaatimukset. Termi on verrattain tuore ja vakiintumaton, sillä esimerkiksi Ionic-kehyksen dokumentaatiossa käytetään myös termiä **hybridisovellus** (engl. *hybrid application*).

Progressiivisten sovellusten ilmeisin hyöty on alustariippumattomuus. Samaa sovellusta voidaan käyttää kaikissa ympäristöissä, jotka tukevat WWW-selaimia. Eri versioita samasta sovelluksesta ei tarvitse kehittää ja ylläpitää erikseen, joten progressiivisten sovellusten avulla voidaan säästää kalliita työresursseja. Hyvä esimerkki progressiivisten sovellusten käyttöä tukevasta markkinatilanteesta on nykyisten älytelevisioiden kirjava tarjonta, sillä valmistajien omien käyttöjärjestelmien lisäksi muunmuassa Apple, Amazon ja Roku kehittävät

televisioon liitettäviä laitteita, joiden avulla käyttäjä voi ajaa erilaisia sovelluksia. Näiden kaikkien laitteiden tukeminen olisi hyvin vaikeaa perinteisten sovellusten avulla. (Frankston 2018)

Tällä hetkellä erityisesti Google panostaa progressiivisten sovellusten kehittämiseen Chrome-selaimen kehitysympäristön yhteydessä. Googlen (Google 2018) mukaan progressiiviset sovellukset tarjoavat perinteisiin WWW-sovelluksiin verrattuna enemmän luotettavuutta, käytettävyyttä ja monipuolisempaa sisältöä. Yrityksen mukaan luotettavuus paranee, sillä progressiiviset sovellukset pystyvät tarjoamaan sisältöä myös ilman verkkoyhteyttä. Google myös väittää, että progressiivisten sovellusten kohdalla käytettävyyttä parantaa nopeammin käyttäjän komentoihin vastaava käyttöliittymä ja sovellusmaisuus puolestaan parantaa käyttäjän immersiota.

Googlen Chrome-selain on rakennettu avoimen lähdekoodin Chromium-selaimen päälle. Yksi merkittävimmistä Chromium-selaimeen pohjautuvista progressiivisten WWW-sovellusten kehitykseen tarkoitetuista kehyksistä on Electron. Electron on Microsoftin nykyisin omistaman GitHubin kehittämä ja ylläpitämä. Esimerkkejä Electron-sovelluksista ovat Discord, Slack ja Visual Studio Code. On mahdollista, että Microsoftin päätös muuttaa Edge-selain Chromium-pohjaiseksi johtui ainakin osittain progressiivisten WWW-sovellusten vaatimista ominaisuuksista kuten tehokkaammasta muistinhallinnasta.

Twitter, AliExpress ja Lancôme ottivat vuonna 2017 käyttöön progressiiviset sovellukset ja julkaistut tulokset ovat olleet positiivisia. Twitter onnistui uuden progressiivisen sovelluksensa ansiosta lisäämään sivupäivityksiä 65% käyttäjäsessiota kohden, lisäämään lähetettyjen Twitter-viestien määrää 75% ja vähentämään käytön lopettamista 20%. Lisäksi uusi sovellus käytti vähemmän kuin 3% natiivin sovelluksen vaatimasta muistitilasta ja vähensi datan käyttöä 70%. Datan käytön määrän vähentyminen on erityisen merkittävää, koska Twitter arvioi, että vuonna 2017 45% sen sisällöstä ladattiin 2G-verkon läpi. (AppInstitute 2017)

AliExpress ja Lancôme ovat molemmat verkkokauppoja, joilla on ollut ongelmia mobiiliverkkokauppojensa tehokkuuden kanssa. Progressiiviseen sovellukseen siirtymällä AliExpress lisäsi uusien asiakkaiden myyntitapahtumien määrää 104%:lla ja Lancôme 17%:lla. Lisäksi

AliExpress tuplasi sivunlatausten määrän sekä kasvatti sessioiden pituutta 74%:lla. Lancôme puolestaan onnistui lisäämään iOS-sessioiden määrää 53%:lla. Edellä mainitut tulokset eivät kuitenkaan ole suoraan yleistettävissä yleisemmin progressiivisten sovellusten vaikutuksiin, sillä niiden otoskoko on erittäin pieni. (AppInstitute 2017)

Progressiivisten sovellusten käyttö ei kuitenkaan ole ongelmatonta. Ensinnäkin, koska progressiivisten WWW-sovellusten hyödyntämät teknologiset kehitysaskeleet ja niiden käyttöön kannustavat taloudelliset tekijät ovat verrattain tuoreita, vaaditaan kehittäjiltä paljon uusien asioiden opettelua sekä vakiintumattomien työkalujen ja kehysten käyttämistä kehitystyössä. Toinen merkittävä haaste voi olla tiedon tallentaminen, sillä selaimen toimintaan pohjautuvalle sovellukselle ei ole varattu tietoturvan takia samoja oikeuksia kuin natiiveille sovelluksille. Kolmas ongelma voi tulla eteen käytettävyydessä, sillä selaimessa toimivan sovelluksen vaatimat resurssit saattavat hidastaa sovelluksen toimintaa. (Divante 2018)

Myös progressiivisten sovellusten keskusmuistinkäyttöä on kritisoitu. Progressiiviset sovellukset vaativat perinteisiä sovelluksia enemmän muistia, koska niitä ajetaan selaimen päällä. On vaikea arvioida, kuinka suuri osa muistinkäytöstä johtuu sovelluksen toteutuksesta ja kuinka suuri osa itse ohjelmistokehyksestä, mutta hyvin yksinkertaisissakin testeissä on saatu tuloksia, jotka viittaavat moninkertaiseen keskusmuistinkäyttöön natiiveihin sovelluksiin verrattuna (O'Kelly 2017).

// - TODO: tähän omaa kriittistä pohdintaa

Progressiiviset WWW-sovellukset sopivat siis parhaiten tilanteisiin, joissa vähän muistia vaativan sovelluksen halutaan tukevan useita eri alustoja, mutta joissa tavallinen WWW-sivu ei riitä. Avusteisen kommunikaation sovelluksien vaatimukset osuvat hyvin yhteen progressiivisten WWW-sovellusten vaatimusten kanssa, sillä kuvakorttien näyttäminen ei ole muistiintensiivistä, mutta toisaalta taas avusteisen kommunikaation sovellusten halutaan olevan helposti muovattavissa henkilökohtaisiin tarpeisiin sopiviksi sekä toimivan useilla eri alustoilla.

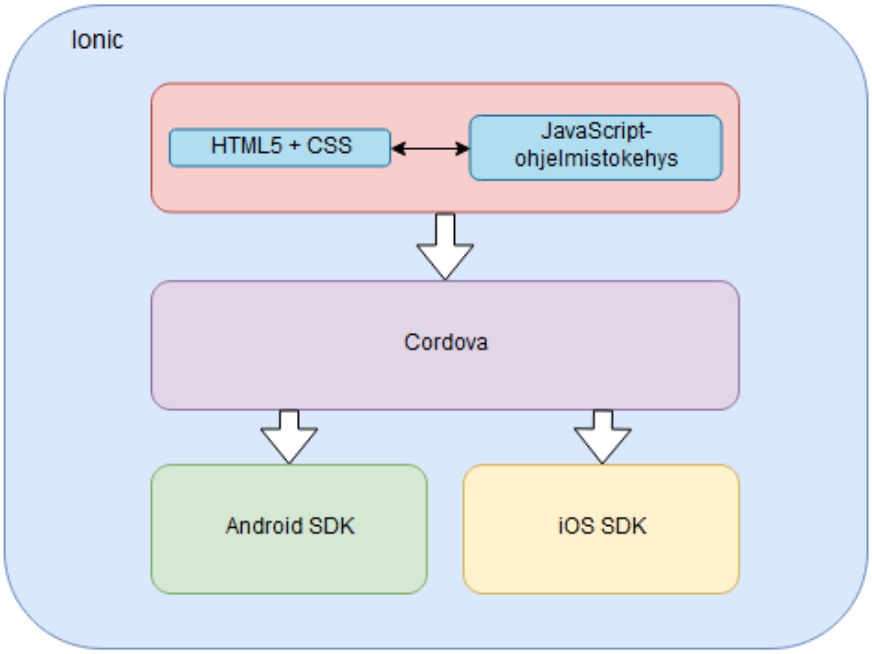
2.3 Ionic-ohjelmistokehyksen rakenne ja ominaisuudet

Ionic on yksi suosituimpia progressiivisten sovellusten kehittämiseen tarkoitetuista ohjelmistokehyksistä. Electronin ollessa työasemille suunnattu kehys, Ionic keskittyy mobiililaitteiden vaatimuksiin. Ionic on avointa lähdekoodia ja hyödyntää Apache Cordova -ympäristöä sekä suosituimpia frontend-ohjelmistokehyksiä. Aiemmin Ionic tuki vain Angular-ohjelmistokehystä, mutta Ionicin versiosta 4 lähtien kehittäjällä on mahdollisuus valita itse käyttämänsä frontend-kehys. Ionic on MIT-lisenssin alainen ja täten avointa lähdekoodia. Angular-ohjelmistokehys perustuu TypeScript-ohjelmointikieleen. Ionicin perusrakenne on kuvattuna kaaviossa 1. Kehyksenä Ionicin pääpaino on tarjota kehittäjälle oikealta näyttävä ja visuaalisesti toimiva sovellus. Sen ei ole tarkoitus korvata tyypillisiä JavaScript-kirjastoja vaan se toimii niiden tukena. (Ionic 2019)

Ionicia ylläpitää ja kehittää Ben Sperryn ja Max Lynch perustama samanniminen yritys. Avoimen lähdekoodin projektina Ionic-kehyksen kehitykseen ja ylläpitoon pääsee osallistumaan GitHub-sivuston välityksellä kuka tahansa. Ionicin viimeisin pääversio on 4.0.0. Viime aikoina Ionicin kehitystyössä painoarvoa on annettu erityisesti Ionic-kehyksen ja Angularin välisten riippuvuuksien vähentämiseen, jotta Ionicia voitaisiin käyttää muiden kehyksien kuten Reactin ja Vue.js:n yhteydessä.

Ionic koostuu komponenteista. Ionicin komponentit ovat uudelleenkäytettäviä käyttöliittymäelementtejä, jotka toimivat sovelluksen käyttöliittymän rakennusosina. Niiden avulla sovellus näyttää yhtenäiseltä ja käyttöliittymän kehitystyö nopeutuu. Komponentit koostuvat HTML:stä, CSS:stä ja JavaScriptistä. Esimerkkejä tällaisista komponenteista ovat muunmuassa painikkeet, välilehdet ja erilaiset listat. Ionicin toteutuksessa on pyritty tukemaan näiden komponenttien mahdollisimman joustavaa räätälöintiä (Ionic 2019).

Ionicin ulkoasu perustuu teemoihin. Myös teemat lisäävät sovelluksen ulkoasun yhtenäisyyttä. Teemat myös mukautuvat eri alustojen ulkoasustandardien mukaisiksi kehittäjän niin halutessa. Esimerkiksi Android- ja iOS-mobiilikäyttöjärjestelmille suunnattujen sovellusten ulkoasu poikkeaa toisistaan, jos käytetään Ionicin oletusteemoja. Teemojen käyttäminen parantaa sovelluksen käytettävyyttä tekemällä sen ulkoasusta ennustettavamman ja tutumman loppukäyttäjälle. Ionicin oletusteemat noudattavat Applen iOS-designperiaatteita sekä Googlen



Kuvio 1. Ionic-sovelluksen perusrakenne.

Material Design -määrityksiä.

2.3.1 Apache Cordova

Apache Cordova (aikaisemmin PhoneGap) on alunperin Nitobin kehittämä sovelluskehitysympäristö mobiililaitteille. Adobe osti Nitobin vuonna 2011 ja myöhemmin uudelleenjulkaisi Apache Cordovan avoimena lähdekoodina. Apache Cordova toimii WWW-sovelluskehyksien ja natiivien sovelluskehyksien välisenä linkityskerroksena (Waranashiwar ja Ukey 2018). Apache Cordovan avulla Ionic ja muut Apache Cordovan päälle rakennetut ohjelmistokehykset voivat toimia syvemmällä tasolla kuin tyypillinen WWW-sovellus, sillä niille tarjoutuu rajapinta suoraan natiiveihin sovelluskehyksiin. Näin progressiivisille WWW-sovelluksille tarjoutuu mahdollisuus käyttää muunmuassa mobiililaitteen kameraa ja GPS-paikannusta.

2.3.2 Angular

Angular on Googlen Angular Teamin ylläpitämä TypeScript-pohjainen käyttöliittymäkehys. Se on jatkoa aiemmin ilmestyneelle AngularJS-kehykselle. Alkuperäinen AngularJS-

kehys julkistettiin vuonna 2010, ja se oli ensimmäinen suosittu ohjelmistokehys dynaamisten HTML-sivujen kehittämiseen mahdollistaen tehokkaamman yhden sivun WWW-sovellusten rakentamisen. Angular julkaistiin vuonna 2016, ja se on kokonaan uudelleenohjelmoitu versio AngularJS:stä.

AngularJS perustuu MVC-arkkitehtuuriin, kun taas uudempi Angular on komponenttipohjainen. Komponenttipohjainen arkkitehtuuri pyrkii tarjoamaan paremman uudelleenkäytettävyyden, luettavuuden, testattavuuden ja ylläpidettävyyden. Ohjelma jaetaan itsenäisiin komponentteihin, joita voidaan käyttää useasti ja niiden itsenäisyys helpottaa yksikkötestaamista. Itsenäiset komponentit ovat huomattavasti helpommin ymmärrettävissä ja niiden korvaaminen on joustavampaa, mikä parantaa ylläpidettävyyttä. (AltexSoft 2018)

Angularin komponenttipohjainen rakenne perustuu kolmeen hiljattain ilmestyneeseen teknologiaan: Web-komponentteihin (engl. *Web Components*), JavaScriptin ES2015-standardiin ja TypeScript-ohjelmointikieleen.

Web-komponentit on laaja-alainen termi, jolla tarkoitetaan neljää WWW-selaimissa yleistyvää standardia: **kustomoitavat elementit** (engl. *custom elements*), **varjo-DOM** (engl. *shadow DOM*), **mallit** (engl. *templates*) ja **HTML-tuonti** (engl. *HTML imports*).

Kustomoitavat HTML-elementit ovat HTML-standardielementtien ulkopuolisia elementtejä, joita voidaan käyttää standardielementtien seassa. Kustomoitava HTML-elementti irroittaa komponentin sivun muista osista, joten se mahdollistaa komponentin eristämisen. Varjo-DOM on piiloitettu osa sivua, jolla on oma eristetty ympäristö skripteille, CSS-tyylitiedostoille ja HTML-elementeille. Varjo-DOM:in elementit ja tyylit eivät vaikuta varjo-DOM:in ulkopuolisiin alueisiin ja vastavuoroisesti muut sivun elementit ja tyylit eivät vaikuta varjo-DOM:in alaisiin osiin. Komponentti voi täten käyttää tätä eristettyä aluetta renderöintiin.

Mallit ovat HTML-palasia, jotka eivät lataudu välittömästi HTML-sivun auetessa vaan ne voidaan aktivoida JavaScriptillä myöhemmin. Malleista on useita toteutuksia eri kehyksissä, mutta Web-komponentit standardisoivat mallien rakentamisen ja tarjoavat suoran tuen niiden hyödyntämiselle selaimessa. Malleja käyttämällä varjo-DOM:iin piilotetusta sisällöstä voidaan tehdä dynaamista. Viimeinen Web-komponenttien osa on HTML-tuonti. HTML-tuonnin avulla HTML-, CSS- ja JavaScript-tiedostoja voidaan ladata yhtenäisinä osina. An-

gular ei käytä HTML-tuontia, vaan se käyttää JavaScriptin moduulilatausta. (Arora ja Hennessy 2018)

2.3.3 TypeScript

TypeScript on avoimen lähdekoodin ohjelmointikieli, jota ylläpitää Microsoft. TypeScript kääntyy suoritusvaiheessa JavaScriptiksi, ja se on tarkoitettu tukemaan JavaScript-ohjelmien kehitystä parantelemalla JavaScriptin ominaisuuksia. TypeScript sisältää ES 2015 -standardin mukaiset ominaisuudet sekä lisäksi se tarjoaa ohjelmoijan käyttöön tyypit ja koristelijat. Angular-ohjelmointikehyksen ohjelmointiin käytetään TypeScript-ohjelmointikieltä.

TypeScriptin on tarkoitus tarjota Microsoftin .NET-ympäristöön tottuneille ohjelmoijille oliolähtöisempi lähestymistapa JavaScript-kehitykseen (Maharry 2013). JavaScript on suosittu ohjelmointikieli, mutta varsinkin ohjelmiston lähdekoodin määrän kasvaessa sen heikkoudet alkavat tulla esiin. TypeScript pyrkii puuttumaan näihin ongelmiin tarjoamalla ohjelmoijalle moduulijärjestelmän, luokat, rajapinnat ja staattisen tyypityksen (Bierman, Abadi ja Torgersen 2014).

Ohjelmointikielen **modulaarisuudella** tarkoitetaan sitä, että muuttujat, funktiot, luokat ja muut vastaavat ohjelmointikielen perusrakenteet ovat olemassa vain moduulien sisällä, ellei niitä erikseen esitellä muille moduuleille (Microsoft 2019c). Tämä vähentää ohjelman eri osien välistä riippuvuutta toisistaan, mikä helpottaa muutosten tekemistä ohjelmistoon.

Perinteisesti JavaScript on käyttänyt uudestikäytettävien komponenttien rakentamiseen funktioita ja prototyyppipohjaista perintää. Iso osa nykyohjelmoijista ei kuitenkaan ole tottunut käyttämään edellä mainittuja keinoja, vaan nykyisin suosituin lähestymistapa uudelleenkäytettävyyteen on luokkien käyttäminen. Myös JavaScriptin oliotukea on pyritty parantelemaan sen viimeisimmissä versioissa. (Microsoft 2019a)

Uudelleenkäytettävyyden lisäksi TypeScriptin luokkien avulla aikaan modulaarisia komponentteja, joita on helpompi ylläpitää ja skaalata. Virheiden löytämistä helpottaa se, että luokkien avulla ohjelmakoodin rajoista saadaan selkeämpiä. Skaalautuvuutta varten täytyy monesti pystyä korvaamaan vanhoja komponentteja uusilla ja luokkia käyttämällä myös tämä on helpompaa.

Rajapintoja käytetään kuvaamaan luokkien ominaisuuksia. TypeScriptissä rajapintoja käytetään tyyppien nimeämiseen ja niiden avulla voidaan tehdä olioiden välisiä sopimuksia ohjelmoijan itse laatiman ohjelmistokoodin sisällä sekä myös ohjelmoijan oman ohjelmistokoodin ja muiden kehittämien kirjastojen välillä. (Microsoft 2019b) Esimerkiksi Swaggerniminen työkalu generoi .NET-palvelinkoodista valmiita rajapintoja, joita TypeScript-pohjaisen WWW-sovelluksen on helpompi käyttää. Rajapinnat ovat luonnollinen osa luokkiin perustuvaa ohjelmointia. Niiden avulla luokan toteutus voidaan eroittaa sen ulkopuolelle näkyvistä osista ja jakaa luokkia ryhmiin. Kuten luokkienkin kohdalla, rajapintojen käyttäminen voi parantaa ohjelman ylläpidettävyyttä, skaalautuvuutta ja osien uudelleenkäytettävyyttä.

// TODO: kirjoita tyypitykseen liittyvät kappaleet uudelleen selkeämmiksi

TypeScript on dynaamisesti tyypitetty kieli samoin kuin JavaScript, mutta se noudattaa **ankkatyypitystä** (engl. *duck typing*). Ankkatyypitys tarkoittaa sitä, että luokan käyttömahdollisuudet eivät määrity sen tyypin perusteella, vaan luokkaa voidaan käyttää sen ominaisuuksien ja metodien perusteella.

JavaScriptistä poiketen TypeScriptin tyypitys on myös vahvaa. Vahvalla tyypityksellä tarkoitetaan sitä, että tyyppien täytyy olla operaatioissa toisiaan vastaavia. Vahvan tyypityksen avulla ohjelmoijan on helpompi nähdä suoraan, että ohjelman tieto on oikean tyyppistä sitä käsitellessä. Iso osa tyyppivirheistä jää tällöin kiinni jo ennen kääntämistä kun taas JavaScriptilla ohjelmoidun ohjelman kohdalla näin ei käy. Toisaalta vahva tyypitys aiheuttaa lisätyötä tyyppimuunnosten takia.

2.4 Käytettävyyden perusteet

Kaikki ihmisen tuottamat esineet ja asiat täytyy suunnitella. Vaikka olemme suorittaneet aktiivista suunnittelua esihistoriallisista ajoista saakka, on tietoisten suunnitteluprosessien tutkimus verrattain tuoretta. Nykyinen suunnittelun kenttä voidaan jakaa karkeasti kolmeen eri osaan: teollinen suunnittelu, käytettävyyssuunnittelu ja kokemussuunnittelu (Norman 2013). Kommunikaatiossa avustavan sovelluksen suunnittelussa tulee olla erityisen kiinnostuneita käytettävyyssuunnitelusta.

// TODO: kappaleen toinen virke uusiksi Hyvin suunniteltu sovellus on miellyttävä käyttää ja ohjaa käyttäjää käyttämään sovellusta oikealla tavalla. Tämä on erityisen tärkeää sovelluksissa, joita on tarkoitus käyttää useasti päivittäin. Avustettua kommunikaatiota käyttävillä ryhmillä on myös omia käytettävyystarpeita, joiden huomioimista varten täytyy ymmärtää käytettävyyttä tavallista laajemmassa kontekstissa.

Ohjelman käyttöliittymän **käytettävyys** (*engl. usability*) on laadullinen määre, joka voidaan jakaa viiteen laadulliseen osa-alueeseen: **opittavuus** (*engl. learnability*), **tehokkuus** (*engl. efficiency*), **muistettavuus** (*engl. memorability*), **virhealttius** (*engl. errors*) ja **tyydyttävyys** (*engl. satisfaction*).

Opittavuus tarkoittaa sitä miten helppo käyttöliittymää on oppia käyttämään. Käyttöliittymän tehokkuus määrittyy siitä miten nopeasti käyttäjät voivat suorittaa ohjelman toimintoja sen jälkeen kun ohjelman käyttöliittymää on opittu käyttämään. Muistettavuus on määrite, jonka avulla arvioidaan miten nopeasti käyttäjä muistaa toiminnot ohjelman käyttämisen lopettamisen ja uudelleen aloittamisen jälkeen. Virhealttiudella tarkoitetaan käyttäjien tekemien virheiden määrää ja niiden laatua sekä kuinka helppoa niistä selviäminen on. Tyydyttävyys kertoo, miten tyytyväinen käyttäjä on käyttöliittymään. On olemassa muitakin laadullisia määreitä kuten **käyttökelpoisuus** (*engl. utility*), jolla määritellään ohjelman tarjoamien eri ominaisuuksien määrää ja vertaamista haluttuihin ominaisuuksiin. Käytettävyyttä ja käyttökelpoisuutta yhtä aikaa tarkistelemalla saadaan aikaan kuva ohjelman **hyödyllisyydestä** (*engl. usefulness*). (Nielsen 2012)

// TODO: opittavuus Lyhyesti ilmaistuna, opittavuus tarkoittaa ohjelmiston kykyä opettaa käyttäjälleen oikea tapa käyttää ohjelmistoa. Opittavuutta vastaava termi on **löydettävyys** (*engl. discoverability*). Opittavuutta lisääviä ominaisuuksia ovat muunmuassa muistettavuus, loogisuus, toistettavuus ja yhdenmukaisuus. Opittavuudeltaan hyvä ohjelma ilmaisee käyttäjälleen tehokkaasti mitä toiminnallisuuksia se sisältää, mitä eri toiminnot tarkoittavat ja kuinka eri toiminnallisuuksia käytetään. (Shamsuddin ym. 2012)

// TODO: tehokkuus Tehokkuus on määritelmä tai mitta siitä, miten helposti ja nopeasti haluttu toiminto voidaan suorittaa tuttua käyttöliittymää käyttämällä. Tehokkuutta voidaan suoraan mitata esimerkiksi kulunutta aikaa tai välivaiheiden määrää mittaamalla.

// TODO: muistettavuus Muistettavuus mittaa sitä miten helppo käyttäjän on käyttää ohjelmistoa sen jälkeen kun ohjelmiston edellisestä käyttökerrasta on kulunut aikaa. Muistettavuutta on hankala mitata suoraan tyypillisten käyttäjätutkimusmetodien avulla, mutta esimerkiksi Affordable Usability -sivuston (Affordable Usability 2011) mukaan sitä voidaan WWW-sovellusten yhteydessä tutkia erilaisten verkkosivuanalytiikkatyökalujen avulla.

// TODO: virhealttius Käyttäjien tekemien virheiden määrän minimointi on yksi käyttettävyyssuunnittelun tärkeimmistä tavoitteista. Don Normanin mukaan (Norman 2013) käyttäjää ei juuri koskaan pitäisi syyttää tekemistään virheistä vaan suurin osa käyttövirheistä johtuu huonosta suunnittelusta.

// TODO: tyydyttävyys Tyydyttävyyttä ymmärtääkseen täytyy tietää, että käyttöliittymä ja käyttäjäkokemus ovat kaksi eri asiaa. Käyttäjäkokemus ja sen tyydyttävyys tai tyydyttämyys on seurausta useasta eri tekijästä. Visuaalinen ulkoasu on osatekijä käyttöliittymän tyydyttävyyttä arvioidessa, mutta jos käyttäjä ei löydä haluamiaan toimintoja, osa halutuista ominaisuuksista puuttuu tai ne on hankalta löytyää, ohjelmiston tyydyttävyys on matala. Tyydyttävyys on viidestä edellä listatusta käytettävyystekijästä kaikista subjektiivisin.

2.4.1 Käytettävyys ja avusteinen kommunikaatio

Avusteiseen kommunikaatioon liittyy omia käytettävyyshaasteita. Tarve avusteiseen kommunikaatioon voi johtua useista eri syistä, joten käyttäjäkirjo ja käyttäjien henkilökohtaisten tarpeiden määrä on suuri. Jo pelkästään tämän havainnon perusteella voidaan olettaa ohjelmiston muokattavuuden olevan tärkeää.

CP-vammaisten avusteista kommunikaatiota tutkineessa tutkimuksessa (Clarke ja Wilkinson 2005) yksi merkittävä käytettävyystekijä on avusteista kommunikaatiota käyttävän henkilön hidas toiminta. Avusteista kommunikaatiota käyttävä henkilö ei välttämättä pysty reagoimaan nopeasti eri keskusteluaiheisiin tai muodostamaan riittävän paljon kommunikaatiota, jolloin keskustelukumppani joutuu arvaamaan, mitä avusteista kommunikaatiota käyttävä henkilö yrittää sanoa. Käytettävyyden näkökulmasta on myös huomioitava, että käyttäjällä on riittävästi aikaa suorittaa valintoja käyttöliittymässä ilman, että näkymä vaihtuu tai käyttöliittymäelementit muuttuvat.

Comunicador-nimistä espanjankielisille avusteisen kommunikaation käyttäjille tarkoitettua ohjelmistoa käsittelevässä tutkimuksessa (Cagigas, Olalla ja Sanchez 2005) nostetaan esiin kontekstiriippuvaisten valintojen tärkeys. Koska kommunikointi on usein vaivalloista avusteisten kommunikaation sovellusten kautta, avusteisen kommunikaation sovelluksessa on hyvä olla mahdollisuus ryhmitellä esivalmistellut sanat tai kuvat tilanteiden mukaan. Esimerkiksi työ-, harrastus- ja arkitilanteissa tarvitaan usein hyvin erilaisia sanavarastoja. Kontekstin valitsemisen lisäksi sanoja tai kuvia täytyy myös pystyä ryhmittelemään ja järjestelemään kontekstiryhmien sisällä.

Autismi vaikuttaa monella tapaa ihmisen kykyyn havainnoida ympäristöä ja sen myötä kykyyn käyttää sovelluksia. Iso-Britannian The National Autistic Society listaa verkkosivuillaan (The National Autistic Society 2018) autistisille ihmisille sopivien verkkosivujen visuaalisesta toteutuksen päävaatimukset. Listauksessa painotetaan erityisesti visuaalisen ilmeen selkeyttä, staattisuutta ja yksiselitteisyyttä. Lisäksi koekäyttäjien roolia korostetaan.

Ilmaiseksi tarjolla olevia avusteisen kommunikaation mobiilisovelluksia tutkineen tutkimuksen (Khan, Tahir ja Raza 2014) mukaan autistisilla käyttäjillä on ominaisuus- ja käytettävyystarpeita, joita tarjolla olleissa sovelluksissa ei ole. Tutkimuksen mukaan mahdollisuus kuvien ottamiseen sekä äänen tallentamiseen on erittäin hyödyllinen ominaisuus avusteisia kommunikaatiosovelluksia käyttäville autisteille. Kuvat eri sijainneista ja ihmisistä auttavat päivittäisessä kommunikaatiossa huomattavasti.

Kommunikaatiosovelluksessa tulisi myös olla mahdollisuus säätää ohjelman asetuksia hallintapaneelin kautta. Hallintapaneelin tulisi olla salasanasuojattu, jotta käyttäjä ei pääse vahingossa poistamaan tärkeitä asetuksia tai kuvia sovelluksesta. Yhteen näyttöön ei saa mahduttaa liian paljon informaatiota, sillä autismiin liittyy usein aistiyliherkkyyttä, mikä heikentää autistin kykyä ymmärtää isoja määriä informaatiota kerrallaan. Tämän takia esimerkiksi kommunikaatiosovelluksen korteilla olevien kuvien määrää tulisi pystyä säätämään sovelluksen hallintapaneelista.

Kommunikaatiosovellusten ääniominaisuus voi auttaa käyttäjää oppimaan sovelluksen käyttöä nopeammin. Äänisyntetisaattoreita Martin-nimisen pojan avulla tutkineessa tutkimuksessa (Schlosser 1999) havaittiin, että Martin oppi muodostamaan uusia lauseita helpom-

min, kun avusteisen kommunikaation sovellus luki kirjoitetun tekstin ääneen. Tutkimuksen luotettavuutta kuitenkin heikentää se, että tutkimuksessa tutkittiin vain yhden koehenkilön oppimistuloksia.

Symbolien käyttöä sovelluksien käyttöliittymissä tekstin korvikkeena pidetään yleisesti hyvänä tapana säästää tilaa, vähentää lukemisen aiheuttamaa kognitiivista kuormaa sekä kiertää tarvetta tekstien kääntämiseen useille eri kielille. Autistisilla henkilöillä esiintyy kuitenkin useasti hahmotusongelmia, jotka haittaavat symbolien merkityksen ymmärtämistä. Autististen henkilöiden on keskimäärin haastavaa ymmärtää abstraktien ja vähän ikonisuutta sisältävien symbolien merkitystä. Esimerkiksi palloa esittävä symboli on autisteille helppo ymmärtää, mutta abstraktien asioiden kuten "mene-verbin yhdistäminen nuoli-symboliin voi olla haastavaa. (Kozleski 1991)

Yksi jonkin verran tutkittu seikka on autismin vaikutus ihmisen suosikkiväreihin. Grandgeorgen ja Masatakan (Grandgeorge ja Masataka 2016) mukaan autistiset lapset vaikuttavat pitävän vihreästä väristä. Keltaista ja ruskeaa tulisi välttää. Edellä mainitun tutkimuksen otoskoko oli kuitenkin pieni, joten tuloksia ei voi yleistää. Grandgeorgen ja Masatakan mukaan lapset pitävät erityisesti pääväreistä.

3 Kommunikointisovelluksen suunnittelu ja toteutus

// Ohjepituus 10-15 sivua.

// TODO: Siltaava kappale, jossa alustetaan aliotsikot

// TODO: Mikä on FreeAAC?

Tutkielmaa varten toteutettiin FreeAAC-niminen avusteisen kommunikaation sovellus.

3.1 Ohjelmistotekninen näkökulma

- Ominaisuusvaatimukset:

* korttipohjainen järjestelmä ”linkki”

* kuvien valitseminen kortilta viestiin ”linkki”

* viestin äänisynteesi ”linkki”

* korttien luominen ja poistaminen ”linkki”

* korttien koon määrittäminen ”linkki”

* kortin vaihtaminen ”linkki”

* kuvien valitseminen korteille valmiista kirjastosta ”linkki”

* kuvien valitseminen korteille kuvia ottamalla ”linkki”

* hallintapaneeli ja asetusten tallentaminen ”linkki”

3.2 Kehitysympäristö ja -työkalut

// TODO: Ionic CLI, Ionic Lab, npm, VS Code, ESLint

// TODO: Kirjoita tämä kappale yhtenäiseksi

Ionic CLI (Ionic Command Line Interface, *suom. Ionic-komentorivikehoite*) on Ionicin tarjoama konsolisovellus, jolla voidaan nopeasti generoida, asentaa ja räätälöidä Ionic-sovellukseen liittyviä tiedostoja.

Ionic-sovellusta voidaan ajaa paikallisena testiversiona neljällä eri tapaa. Selaimessa, iOS- tai Android-simulaattorilla, mobiililaitteen selaimessa tai itsenäisenä sovelluksena puhelimessa.

Node.js on alustariippumaton runtime-ympäristö palvelinpuolen JavaScript-koodin suorittamiseen. Lähes kaikki JavaScript-pohjaiset kirjastot noajaavat Node.js:ään ja myös Ionic vaatii Node.js:n asennuksen. Node.js Package Manager eli **npm** on paketinhallintajärjestelmä JavaScript-kirjastoille.

Visual Studio Code on Microsoftin kehittämä ja ylläpitämä avoimen lähdekoodin Electron-pohjainen ohjelmistoympäristö. Angularin käyttämä TypeScript-ohjelmointikieli on myös Microsoftin kehittämä, joten Visual Studio Coden TypeScript-tuki on tällä hetkellä erittäin hyvällä tasolla.

ESLint on koodianalyysityökalu, joka käy läpi ohjelmistoympäristössä olevaa ohjelmakoodia ja auttaa ohjelmoijaa ohjelmoimaan koodianalyysityökaluun valittujen asetusten mukaista ohjelmakoodia.

3.3 Rakenne

FreeAAC noudattaa Ionic-sovelluksien oletusrakennetta. Ionic-sovelluksissa kansiorakenteen ylimmälle tasolle sijoitetaan projektin asetuksia sisältävät tiedostot. Projektin asetustiedostoista ohjelmistokehitysprosessin kannalta tärkein on `package.json`. Tiedostossa listataan kaikki ohjelmiston käyttämät ulkopuoliset kirjastot ja Node.js:n paketinhallintajärjestelmä npm lataa sen perusteella oikeat versiot jokaisesta kirjastosta Ionic-sovelluksen käytettäväksi.

Seuraavalla kansiotasolla Ionicin oletusrakenneteessa projekti jakautuu kahteen pääkansioon: `Resources` ja `Src`. Ohjelmiston pikakuvakkeiden ja käynnistysruudun tarvitsemat mediatiedostot ovat `Resources`-kansiossa.

`Src`-kansiossa sijatsevat ohjelmiston lähdekoodia sisältävät tiedostot. `Src`-kansion päätasolla sijaitsevat `index.html`, `manifest.json` ja `service-worker.js`-tiedostot. `Index.html` on ensimmäinen HTML-tiedosto, jonka käynnistyvä Ionic-sovellus lukee. Se sisältää Ionic-sovelluksen juurikomponentin ja linkkejä sovelluksen yleisesti vaatimiin resursseihin kuten tyylitiedostoihin.

`Src`-kansiota alemmalla tasolla on oletusprojektissa viisi eri kansiota: `Assets`, `Classes`, `Pages`, `Providers` ja `Theme`.

`Assets`-kansioon ohjelmiston kehittäjä voi sijoittaa kuva- ja äänitiedostoja, joita ohjelman toiminnallisuudet vaativat. FreeAAC-sovelluksen kuvakirjasto sijaitsee kansiossa.

`Classes`-kansiossa sijaitsevat ohjelmiston käyttämät luokat. FreeAAC-ohjelmistossa kaksi käytössä olevaa luokkaa ovat `Card` ja `WordSymbol`. `Card`-luokka vastaa kommunikointisovelluksissa käytettäviä kortteja. Se sisältää tiedon kortin nimestä, sen koosta ja kortin sisältämistä kuvista ja symboleista. `WordSymbol`-luokka taas on representaatio korteilla olevien kuvien ja symbolien tietomallista. Se sisältää kuvan tai symbolin nimen sekä viittauksen kuvatiedostoon.

Ohjelmiston näkymät sijaitsevat `Pages`-kansiossa. FreeAAC-ohjelmistossa on seitsemän eri näkymää: Home, Main, CardCreate, CardDelete, Info, Options ja SelectSymbolModal. Angular-ohjelmistokehystä käyttävässä Ionic-sovelluksessa näkymät ovat HTML-tiedostoja, joihin on **sidottu** (*engl. bind*) komponenttitiedostossa olevia muuttujia ja aliohjelmia. Lisäksi kansioon sijoitetaan yleensä moduulitiedosto, jossa sijaitsevat viittaukset tarvittaviin ulkopuolisiin kirjastoihin sekä muihin moduuleihin ja tyylitiedosto, jolla ohjelmiston ulkoasua voidaan säätää näkymäkohtaisesti.

`Theme`-kansiossa sijaitsevat ohjelmiston ulkoasuun vaikuttavat SCSS-tiedostot. Ionicin oletusprojektissa se sisältää `variables.scss`-tiedoston, joka asettaa projektille Ionicin oletusteeman. Kehittäjä voi itse vapaasti säätää ohjelmiston päätason ulkoasua tätä kautta.

Riippuvuusinjektio (engl. *dependency injection*) on suunnittelumalli, jossa luokalle annetaan toinen luokka tai staattinen aliohjelma, joka tarjoaa luokalle uusia ominaisuuksia käyttöön. Riippuvuusinjektiossa luokka tai staattinen aliohjelma annetaan toiselle luokalle sen si-

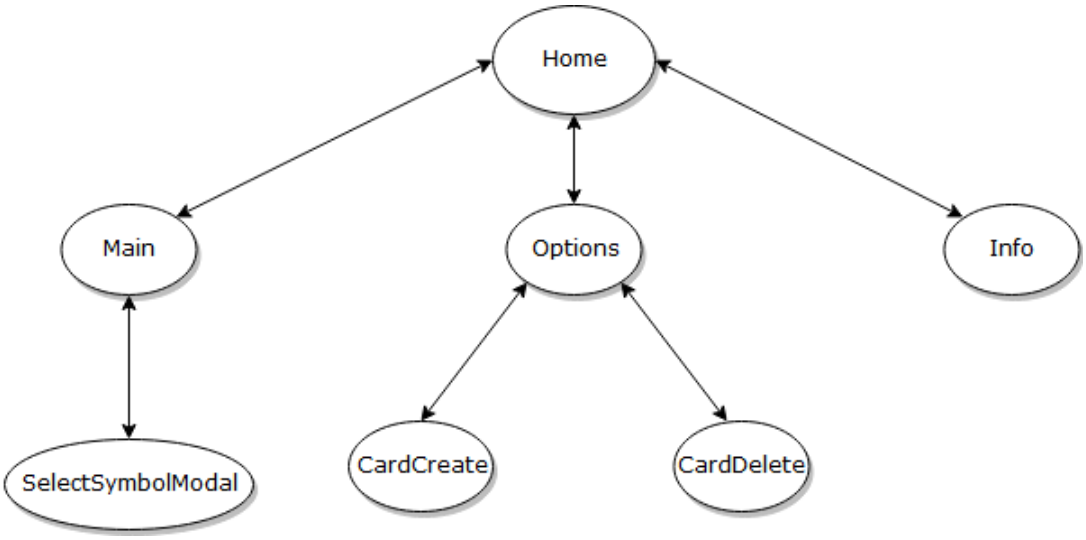
jaan, että luokka itse hakisi luokan tai staattisen aliohjelman. Angular-ohjelmistokehyksessä riippuvuusinjektio toteutetaan tuomalla toinen luokka import-lauseella mukaan luokan lähdekoodiin ja julistamalla riippuvuusinjektio luokan konstruktorissa.

`Providers`-kansioon sijoitetaan ohjelmiston palveluluokat. **Palvelumalli** (engl. *service pattern*) on suunnittelumalli, jossa useiden eri luokkien käyttämiä palveluita sijoitetaan samaan luokkaan yhdeksi palveluksi. Näin keskeisten palveluiden toteutus voidaan irroittaa itse luokkien toteutuksesta.

// TODO: DI:stä ja palvelumallista lisää?

3.4 Ohjelmiston näkymät

Ionicin näkymiä käsitellään pinona. Ionic-sovelluksessa näkymästä toiseen siirrytään lisäämällä uusi näkymä pinon päällimmäiseksi käyttämällä Ionicin `NavController`-kontrolleriluokkaa. Näin näkymäsiirtymien historia säilyy ja käyttäjä voi palata edellisiin näkymiin helposti painamalla mobiililaitteen edellinen-painiketta. Tällöin viimeksi pinoon lisätty näkymä poistetaan pinosta.



Kuvio 2. FreeAAC-sovelluksen näkymäpuu.

FreeAAC:n Ionic-näkymissä sisältö on jaettu kahden eri päätason HTML-elementin sisälle: `ion-header` ja `ion-content`. `Ion-header`-lohkossa sijaitsevat näkymän ylälaitaan

tuleva teksti, joka kertoo missä näkymässä ollaan tällä hetkellä. Lisäksi sinne voidaan sijoittaa valikkotoimintoja. `Ion-content`-lohkossa sijaitsee näkymän varsinainen pääsisältö.

Home-näkymä on FreeAAC-ohjelmiston ensimmäinen näkymä ja se sisältää ohjelmiston päävalikon. Päävalikossa sijaitsevat painikkeet, joita klikkaamalla käyttäjä pääsee keskustelunäkymään (Main), asetusnäkymään (Options) tai infonäkymään (Info). Home-näkymän komponenttitiedostossa ei ole muita kuin näkymän vaihtamiseen liittyviä aliohjelmia.

Ohjelmiston tärkein näkymä on Main-niminen näkymä, jossa sijaitsevat ohjelmiston varsinaiset kommunikointityökalut. Näkymä koostuu kolmesta eri komponentista: viestilaatikko, korttivalinta ja symbolivalinta. Viestilaatikkossa näkyvät käyttäjän valitsemat symbolit järjestyksessä. Käyttäjä voi poistaa symboleita yksi kerrallaan viimeisestä symbolista alkaen. Symbolikortti valitaan pudotusvalikosta, josta löytyvät käyttäjän kortinluonti-näkymässä tekemät kortit. Sivun alalaidassa näkyvät kortin symbolit, joista käyttäjä voi valita haluamansa symbolit viestilaatikkoon.

Main-näkymän komponenttitiedosto sisältää viestilaatikon päivittämiseen sekä korttien tietojen lataamiseen tarvittavat aliohjelmat. Korttien lataamiseen käytetään `CardDataProvider`-luokkaa, joka on injektoitu mukaan komponenttiin sen konstruktorissa.

Options-näkymä on kokoomanäkymä ohjelmiston hallintaa varten. Options-näkymän kautta käyttäjä pääsee siirtymään korttien luontinäkymään (CardCreate) ja korttien poistonäkymään (CardDelete).

Kortteja luodaan ohjelmiston CardCreate-näkymässä. CardCreate-näkymässä kortille voidaan valita nimi, koko ja liittää siihen kuvia kuvakirjastosta. Kaikki edellä mainitut ovat pakollisia ja käyttäjälle näytetään toast-virheilmoitus, jos jokin tieto puuttuu. Toast-virheilmoitukset käyttävät Ionicin `ToastController`-luokkaa. Kuvia valitaan erillisen modaalin kautta ja Ionic-sovelluksessa modaalien näyttämisestä vastaa `ModalController`-luokka. Korttien tallentaminen tapahtuu `CardDataProvider`-luokan kautta.

Kuvien valintaa varten aukeava modaali sisältää SelectSymbolModal-näkymän. Näkymässä kuvan voi valita eri kategorioista. Kategoria valitaan pudotusvalikosta. SelectSymbolModal-näkymän komponenttitiedosto sisältää kuvan tallentamiseen ja modaalin sulkemiseen liit-

tyvät aliohjelmat. FreeAAC-ohjelmiston kuvakirjasto koostuu Papunet-sivuston tarjoamasta Creative Commons -lisenssin alaisesta kuvapankista.

Kortteja voidaan poistaa erillisessä CardDelete-näkymässä. CardDelete-näkymä koostuu pudotusvalikosta, josta voidaan valita mikä kortti halutaan poistaa sekä vahvistuspainikkeesta. Korttilistaus ladataan `CardDataProvider`-luokan avulla ja myös poistokomento kulkee samaa reittiä pitkin.

Info-näkymässä listataan FreeAAC-sovellukseen liittyviä tietoja. Info-näkymä on staattinen eikä siinä ole toiminallisuuksia.

FreeAAC:lla on kaksi palveluluokkaa: `CardDataProvider` ja `ImageDataProvider`. `CardDataProvider`-luokan tehtävä on tarjota näkymäkomponenteille tietoa olemassa olevista korteista. `ImageDataProvider`-luokka huolehtii ohjelmiston kuvakirjaston eri kuvien toimittamisesta näkymäkomponenteille.

// TODO: tallennustila

3.5 Käytettävyysnäkökulma

- Ominaisuusvaatimukset:

* yleiset käytettävyysvaatimukset ”linkki”

* selkeiden päävärien käyttö ja riittävästi kontrastia ”linkki”

* abstraktien symbolien välttäminen ”linkki”

* mahdollisuus kontekstiin sopivien asetusten säätämiseen ”linkki”

* näkymien staattisuus ja yksinkertaisuus ”linkki”

// TODO: Vaatimuslista.

// TODO: Yleiset käytettävyysratkaisut.

// TODO: Erikoisryhmien käytettävyysnäkökulma.

Tutkimuksessa toteutettu sovellus on tarkoitettu erityisesti autistisille käyttäjille.

// TODO: Käytettävyys näkymittäin.

4 Toteutuneen sovelluksen arviointi

// Ohjepituus 15-20 sivua.

// TODO: Ohjelmakoodin määrä (rivit).

// TODO: Muistinkäyttö.

Kuvien käsittely ja tallentaminen haastavaa. Progressiiviset sovellukset kohtalaisen hyviä tähän käyttötarkoitukseen eli yksinkertaisiin mobiilikeskeisiin sovelluksiin.

5 Pohdinta

// Ohjepituus 5-10 sivua.

– Mitä aineistoanalyysi kertoo tutkimushypoteesista? Tukevatko tulokset teorialuvussa esitettyjä näkem

6 Yhteenveto

// Ohjepituus 3-5 sivua.

– Tutkielman viimeinen luku on Yhteenveto. Sen on hyvä olla lyhyt; siinä todetaan, mitä tutkielmassa e

Lähteet

Affordable Usability. 2011. “Usability Goals: Memorability of a Website”. Viitattu 16. maaliskuuta 2019. http://www.affordableusability.com/usability/memorability.html.

AltexSoft. 2018. “The Good and the Bad of Angular Development”. Viitattu 29. joulukuuta 2018. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/.

AppInstitute. 2017. “A Beginner’s Guide to Progressive Web Apps”. Viitattu 3. heinäkuuta 2018. https://appinstitute.com/a-beginners-guide-to-progressive-web-apps/.

Arora, Chandermani, ja Kevin Hennessy. 2018. *Angular 6 by Example.* 3. painos. Livery Place, 35 Livery Street, Birmingham, UK: Packt Publishing.

Bierman, Gavin, Martin Abadi ja Mads Torgersen. 2014. “Understanding TypeScript”. *Lecture Notes in Computer Science* 8586:257–281. doi:978-3-662-44202-9_11.

Cagigas, Sira E. Palazuelos, Marisa M. Dominguez Olalla ja Jose L. Martinez Sanchez. 2005. “Graphic Communicator with Optimum Message Access for Switch Users”. *Assistive Technology: From Virtuality to Reality : AAATE 2005:* 207–211.

Clarke, Michael, ja Ray Wilkinson. 2005. “The Usability of AAC in Conversation: A Case Study of Interaction between a Child using AAC and her Classmate”. *Assistive Technology: From Virtuality to Reality : AAATE 2005:* 3–7.

Divante. 2018. “PWA Design Challenges”. Viitattu 30. kesäkuuta 2018. https://divante.co/blog/pwa-design-challenges/.

Frankston, Bob. 2018. “Progressive Web Apps”. *IEEE Consumer Electronics Magazine* 16 (3): 106–107. doi:10.1109/MCE.2017.2776463.

Google. 2018. “Progressive Web Apps”. Viitattu 26. kesäkuuta 2018. https://developers.google.com/web/progressive-web-apps/.

Grandgeorge, Marine, ja Nobuo Masataka. 2016. “Atypical Color Preference in Children with Autism Spectrum Disorder”. *Frontiers in Psychology* 7. doi:10.3389/fpsyg.2016.01976.

Ionic. 2019. “Ionic Documentation”. Viitattu 4. tammikuuta 2019. https://ionicframework.com/docs.

Khan, Sehrish, Mutahira Naseem Tahir ja Arif Raza. 2014. “Usability issues for smartphone users with special needs — Autism”. *IEEE Xplore.* doi:10.1109/ICOSST.2013.6720615.

Kozleski, Elizabeth B. 1991. “Visual symbol acquisition by students with autism”. *Exceptionality: A Special Education Journal* 2 (4): 173–194. doi:10.1080/09362839109524782.

Maharry, Dan. 2013. *TypeScript Revealed.* 1. painos. New York City, NY: Apress.

Microsoft. 2019a. “TypeScript Documentation - Classes”. Viitattu 17. tammikuuta 2019. https://www.typescriptlang.org/docs/handbook/classes.html.

———. 2019b. “TypeScript Documentation - Interfaces”. Viitattu 20. tammikuuta 2019. https://www.typescriptlang.org/docs/handbook/interfaces.html.

———. 2019c. “TypeScript Documentation - Modules”. Viitattu 17. tammikuuta 2019. https://www.typescriptlang.org/docs/handbook/modules.html.

Nielsen, Jakob. 2012. “Usability 101: Introduction to Usability”. Viitattu 2. maaliskuuta 2019. https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

Norman, Don. 2013. *The Design of Everyday Things.* 3. painos. New York City, NY: Basic Books.

Nunes, Débora R. P. 2008. “AAC Interventions for Autism: a Research Summary”. *International Journal of Special Education* 23 (2): 17–26.

O'Kelly, Rory. 2017. "Electron Memory Usage Compared to Other Cross-Platform Frameworks". Viitattu 21. helmikuuta 2019. http://roryok.com/blog/2017/08/electron-memory-usage-compared-to-other-cross-platform-frameworks/.

Schlosser, Ralf. 1999. "Comparative efficacy of interventions in augmentative and alternative communication". *Augmentative and Alternative Communication* 15:56–68. doi:https://doi.org/10.1080/07434619912331278575.

Shamsuddin, Nurul Afiqah, Shahida Sulaiman, Sharifah Mashita Syed-Mohamad ja Kamal Zuhairi Zamli. 2012. "Improving learnability and understandability of a Web application using an action-based technique". *2011 Malaysian Conference in Software Engineering*. doi:10.1109/MySEC.2011.6140678.

Sigafoos, Jeff, ja Erik Drasgow. 2001. "Conditional Use of Aided and Unaided AAC: A Review and Clinical Case Demonstration". *Focus On Autism And Other Developmental Disabilities* 16 (3): 152–161.

The National Autistic Society. 2018. "Designing Autism-friendly Websites". Viitattu 28. kesäkuuta 2018. https://www.autism.org.uk/professionals/others/website-design.aspx.

Waranashiwar, Juilee, ja Manda Ukey. 2018. "Ionic Framework with Angular for Hybrid App Development". *International Journal of New Technology and Research* 4 (5): 1–2.

Vaughn, Bobbie, ja Robert Horner. 1995. "Effects of Concrete Versus Verbal Choice Systems on Problem Behavior". *Augmentative and Alternative Communication* 11 (2): 89–92. doi:10.1080/07434619512331277179.

Liitteet