

Roope Kivioja

**Puheen ongelmista kärsiville tarkoitettun
kommunikointisovelluksen toteuttaminen Ionic-kehyksellä**

Tietotekniikan pro gradu -tutkielma

17. maaliskuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Roope Kivioja

Yhteystiedot: roope.kivioja@gmail.com

Ohjaajat: Jukka-Pekka Santanen ja Jonne Itkonen

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: 25+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

Kuvio 1. Ionic-sovelluksen perusrakenne	7
---	---

Sisältö

1	JOHDANTO	1
2	KOMMUNIKOINTISOVELLUKSEN RAKENTAMISEEN TARVITTAVIA TAUSTATIETOJA JA MENETELMIÄ	2
2.1	Avusteinen kommunikaatio	2
2.2	Progressiiviset WWW-sovellukset	4
2.3	Ionic-ohjelmistokehityksen rakenne ja ominaisuudet	6
2.3.1	Apache Cordova	8
2.3.2	Angular	8
2.3.3	TypeScript	9
2.4	Käytettävyyden perusteet	11
2.4.1	Käytettävyys ja avusteinen kommunikaatio	13
3	KOMMUNIKOINTISOVELLUKSEN SUUNNITTELU JA TOTEUTUS	14
3.1	Ohjelmistotekninen näkökulma	14
3.1.1	Valikot ja navigaatio	14
3.1.2	Korttien luontinäkökulma	14
3.1.3	Kommunikaationäkökulma	14
3.2	Käytettävyysnäkökulma	14
4	TOTEUTUNEEN SOVELLUKSEN ARVIOINTI	15
5	POHDINTA	16
6	YHTEENVETO	17
	LÄHTEET	18
	LIITTEET	21

1 Johdanto

// Ohjepituus 5 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-kehiksen toiminnan tutkiminen
- > tavoite: Ionic-kehiksen avulla tehdyn avusteisen kommunikaation sovelluksen vertaaminen ohjelmistoteknisiin ja käytettävyyssopijaisiin suosituksiin kehittäjän näkökulmasta
- > tavoite: Ionic-kehiksen avulla tehdyn avusteisen kommunikaation sovelluksen soveltumisen autistien käytettävyyssopijaisiin 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 tukevaa 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.

Puhetta tukevat ja korvaavat kommunikointisovellukset käyttävät yleisimmin symboleja. Autisteille tyypillistä on vahva visuaalis-avaruudellinen hahmotuskyky, joten tutkimuksen mukaan piirroksiin ja valokuviiin 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 yhtäaikaaisesti tai peräkkäin. Puhetta tukevasta ja korvaavasta kommunikaatiosta kannattaa tehdä monimodaalista useista eri syistä. Ensinnäkin, suurin osa kaikesta kommunikaatiosta on monimodaalista: toiselle ihmiselle puhuttaessa on tavallista selkeyttää sanomaa ilmeillä ja eleillä. Toiseksi, puhetta tukevaa ja korvaavaa kommunikaatiota käyttävä henkilön tarvitsee useasti kommunikoida muiden puhetta tukevaa ja korvaavaa kommunikaatiota käyttävien henkilöiden kanssa, jolloin vaihtoehtoja on hyötyä. Kolmas merkittävä syy on se, että eri puhetta tukevissa ja korvaavissa kommunikaatiotavoissa on vahvuuksia ja heikkouksia, joten eri kommunikaatiotapoja sekoittamalla voidaan korvata yksittäisen kommunikaatiotavan heikkouksia. (Sigafos 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-kehiksen 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: 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% per sessio, 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%. Datat käyttö 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 mobiili-verkkokauppojensa 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. (AppInstitute 2017) Edellä mainitut tulokset eivät kuitenkaan ole suoraan yleistettävissä yleisemmin progressiivisten

sovellusten vaikutuksiin, sillä niiden otoskoko on erittäin pieni.

Progressiivisten sovellusten käyttö ei kuitenkaan ole ongelmaton. Ensinnäkin, koska progressiivisten WWW-sovellusten hyödyntämät teknologiset kehitysaskleet ja niiden käyttöön kannustavat taloudelliset tekijät ovat verrattain tuoreita, vaaditaan kehittäjiltä paljon uusien asioiden opettelua sekä vakiintumattomien työkalujen sekä 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 muistinkäyttöä 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 muistinkäyttöön natiiveihin sovelluksiin verrattuna (O’Kelly 2017).

// - TODO: tähän omaa kriittistä pohdintaa

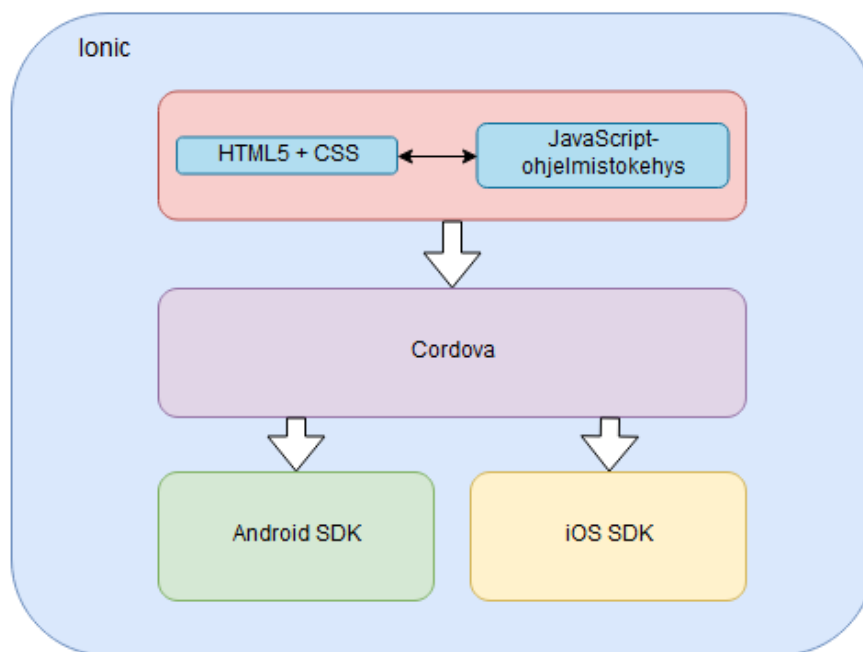
Progressiiviset WWW-sovellukset sopivat siis parhaiten tilanteisiin, joissa vähän muistia vaativan sovelluksen halutaan tukevan useita eri alustoja, mutta joissa tavallinen WWW-sivu ei riitä. Avusteisen kommunikaation sovelluksien vaatimukset osuvat hyvin yhteen progressiivisten WWW-sovellusten vaatimusten 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 tuottamiseen suunnitelluista ohjelmistokehyksistä. Electronin ollessa kiinteille päätteille suunnattu kehys, Ionic keskittyy mobiililaitteiden vaatimuksiin. Ionic on avointa lähdekoodia ja hyödyntää Apache Cordova -

ympäristöä sekä Angular-ohjelmistokehystä. 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 Sperry 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.



Kuvio 1. Ionic-sovelluksen perusrakenne.

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 kustomointia (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 lopputulokseen käyttäjälle. Ionicin oletusteemat noudattavat Applen iOS-designperiaatteita sekä Googlen Material Design -määrittelyjä.

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-sovelluskehityksen ja natiivien sovelluskehityksen 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 sovelluskehityksiin. 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 julkaistiin vuonna 2010, ja se oli ensimmäinen suosittu ohjelmistokehys dynaamisten HTML-sivujen tuottamiseen 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 kom-

ponentteihin, 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 piilotettu osa sivua, jolla on oma eristetty ympäristö skripteille, CSS-tyylitiedostoille ja HTML-elementeille. Varjo-DOM:n elementit ja tyylit eivät vaikuta varjo-DOM:n ulkopuolisiin alueisiin ja vastavuoroisesti muut sivun elementit ja tyylit eivät vaikuta varjo-DOM:n 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-DOMiin piilotetusta sisällöstä voidaan tehdä dynaamista. Viimeinen web-komponenttien osa on HTML-tuonti. HTML-tuonnin avulla HTML-, CSS- ja JavaScript-tiedostoja voidaan ladata yhtenäisinä osina. Angular 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 suoritussvaiheessa 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 tyyppit ja koristelijat.

Angular-ohjelmointikehyksen ohjelmointiin käytetään TypeScript-ohjelmointikieltä.

TypeScriptin on tarkoitus tarjota Microsoftin .NET-ympäristöön tottuneille ohjelmoijille olionlähtöisempi 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 tyyppityksen (Gavin Bierman ja Torgeresen 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. (Microsoft 2019a)

Uudelleenkäytettävyyden lisäksi TypeScriptin luokkien avulla saadaan JavaScriptiin verrattuna helpommin 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 tuottamien kirjastojen välillä. (Microsoft 2019b) Esimerkiksi Swagger-niminen työkalu generoi .NET-palvelinkoodista valmiita rajapintoja, joita TypeScript-pohjaisen WWW-sovelluksen on helpompi käyttää. Rajapinnat ovat luonnollinen jatke luokille. Niiden avulla luokan toteutus voidaan erottaa sen ulkopuolelle näkyvistä osista ja jakaa luokkia ryhmiin. Kuten luokkienkin kohdalla, rajapintojen käyttäminen voi parantaa ohjelman ylläpidettävyyttä, skaalautuvuutta ja osien uudelleenkäytettävyyttä.

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 JavaScriptillä 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 käytettävyyssuunnittelusta.

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 **tyytyttävävyys** (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. Muis-

tettavuus on määrite, jonka avulla arvioidaan miten nopeasti käyttäjä muistaa toiminnot ohjelman käyttämisen lopettamisen ja uudelleen aloittamisen jälkeen. Virhealttiudella tarkoitetaan käyttäjien tekemien virheiden määrää, niiden laatua ja kuinka helppoa niistä selviäminen on. Tyydyttävyyys 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. (Nurul Afiqah Shamsuddin ja Zamli 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 (Usability 2011) mukaan sitä voidaan WWW-sovellusten yhteydessä tutkia erilaisten verkkosivuanalytiikkatyökalujen avulla.

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

// TODO: tyydyttävyyys Tyydyttävyyttä ymmärtääkseen täytyy tietää, että käyttöliittymä ja käyttäjäkokemus ovat kaksi eri asiaa. Käyttäjäkokemus ja sen tyydyttävyyys tai tyydyttämyys on seurausta useasta eri tekijästä. Visuaalinen ulkoasu on osatekijä käyttöliittymän tyydyttä-

vyyttä arvioidessa, mutta jos käyttäjä ei löydä haluamiaan toimintoja, osa halutuista ominaisuuksista puuttuu tai ne on hankalta löytyä, ohjelmiston tyydyttävyyden on matala. Tyydyttävyyden 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 (**classmate-aac-study**) 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, joten keskustelukumppani joutuu arvaamaan mitä avusteista kommunikaatiota käyttävä henkilö yrittää sanoa. Käytettävyyden näkökulmasta on myös huomiotava, että käyttäjällä on riittävästi aikaa suorittaa valintoja käyttöliittymässä ilman, että näkymä vaihtuu tai käyttöliittymäelementit muuttuvat.

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

Symbolien käyttöä tulisi välttää // TODO: lähde

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

// TODO: etsi lisää lähteitä

3 Kommunikointisovelluksen suunnittelu ja toteutus

// Ohjepituus 10-15 sivua.

// TODO: Siltaava kappale, jossa alustetaan aliotsikot

3.1 Ohjelmistotekninen näkökulma

// TODO: Tekninen kuvaus ohjelmasta.

Sovellus koostuu valikkorakenteesta, korttien luontinäkymistä ja kommunikaationäkymästä.

3.1.1 Valikot ja navigaatio

Valikot Ionicin standardielementeistä luotu.

3.1.2 Korttien luontinäkö

Korttien luontinäkö Voidaan tehdä erikokoisia kortteja. Käytetään Papu.net-sivuston vapaata kuvapankkia.

3.1.3 Kommunikaationäkö

Kommunikaationäkö Valitaan kortti. Kirjataan viesti symboleja painamalla.

3.2 Käytettävyysnäkökulma

// TODO: Kerrotaan käytettävyyden näkökulmasta. Erityisesti erikoisryhmien käytettävyys nostetaan esille.

Tutkimuksessa toteutettu sovellus on tarkoitettu erityisesti autistisille käyttäjille.

4 Toteutuneen sovelluksen arviointi

// Ohjepituus 15-20 sivua.

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 esitetyjä näkem

6 Yhteenveto

// Ohjepituus 3-5 sivua.

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

Lähteet

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.

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.

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

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>.

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>.

- Microsoft. 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.
- Nurul Afiqah Shamsuddin, Sharifah Mashita Syed-Mohamad, Shahida Sulaiman, 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.
- 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/>.
- 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.
- Society, The National Autistic. 2018. "Designing Autism-friendly Websites". Viitattu 28. kesäkuuta 2018. <https://www.autism.org.uk/professionals/others/website-design.aspx>.
- Usability, Affordable. 2011. "Usability Goals: Memorability of a Website". Viitattu 16. maaliskuuta 2019. <http://www.affordableusability.com/usability/memorability.html>.

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