

React Native kehittäjän arkea

Miika Kilponen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018



Tekijä(t) Miika Kilponen	
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma	
Opinnäytetyön otsikko React Native kehittäjän arkea	Sivu- ja liite- sivumäärä 8 + 1
Opinnäytetyön otsikko englanniksi The day to day operations of a React Native developer	
<p>Tässä päiväkirjaopinnäytetyössä opiskelija kuvaa päivittäistä oppimista, oppimisen ja osaamisen kehittymistä ja työarkea mobiilikehittäjänä GBC Networks Oy:llä. Päiväkirja sijoittuu ajalle 30.4.2018-27.7.2018 ja se sisältää aloitusviikon, kymmenen raportointiviikkoa ja lopetusviikon. Raportointiviikkoihin sisältyy päivittäiset merkinnät työpäivän tavoitteista, kulusta ja lopputuloksista, sekä jokaisen työviikon päättävä analyysi.</p> <p>GBC Networks on 8 työntekijää työllistävä yritys, jonka työntekijöillä on kirjava määrä erilaisia tehtäviä yrityksessä. Opiskelija otettiin ensin palkalliseen työharjoitteluun kehittämään GBC:een VAIHDE-sovellusta, ja myöhemmin työharjoittelua jatkettiin kahdellatoista viikolla tämän opinnäytetyön vuoksi. Opiskelija työskentelee pääasiassa VAIHDE-sovelluksen parissa sovelluskehittäjänä.</p> <p>Lopputuloksena opiskelija koodasi ja päivitti GBC Networksin VAIHDE-sovellukseen ja muihin web-palveluihin useita tärkeitä päivityksiä ja ominaisuuksia. Tehtävien kautta opiskelija syvensi osaamistaan JavaScript ohjelmoinnista, React Nativesta ja web- sekä mobiilikehitysprosesseista.</p>	
Asiasanat React Native JavaScript Firebase	

Sisällys

1	Johdanto	1
2	Lähtötilanteen kuvaus	3
2.1	Oman nykyisen työn analyysi	3
2.2	Sidosryhmät työpaikalla	4
2.3	Vuorovaikutustaidot työpaikalla	5
3	Päiväkirjaraportointi	7
3.1	Seurantaviikko 1	7
3.2	Seurantaviikko 2	14
3.3	Seurantaviikko 3	19
3.4	Seurantaviikko 4	23
3.5	Seurantaviikko 5	27
3.6	Seurantaviikko 6	31
3.7	Seurantaviikko 7	35
3.8	Seurantaviikko 8	38
3.9	Seurantaviikko 9	41
3.10	Seurantaviikko 10	45
4	Pohdinta ja päätelmät	49
	Lähteet	53
	Liitteet	54
	Liite 1. Käsitteet	54

1 Johdanto

GBC (Global Business Calls) Networks on vuonna 2001 perustettu puhelinjärjestelmiä ja VoIP -palveluita tuottava palveluoperaattori, joka myy palveluitaan räätälöidysti yritysasiakkaille.

Tekstissä esiintyvät ammattisanat ja lyhenteet on selitetty liitteessä 1: käsitteet.

Ennen tämän päiväkirjan kirjoittamisen aloittamista suoritin 100-päiväisen työharjoitteluopintojaksoni GBC:eellä. Harjoittelua aloittaessani olin kehittänyt muutamia web-projekteja koulukursseja varten (HTML5, CSS3 ja JavaScript) ja jonkin verran itseopiskelua vapaa-ajallani, erityisesti keskittyen JavaScriptin perusteisiin ja ohjelmoinnin peruskäsitteisiin. Tuolloin minulla ei ollut vielä kokemusta laajoista kehitysprojekteista. Harjoittelun alkaessa pääsin ensiksi kehittämään GBC Connect web-sovellusta. Kun laajemman web-sovelluksen kanssa työskentely alkoi sujua, siirryin kehittämään GBC:een VAIHDE-sovellusta.

VAIHDE-sovellus on kirjoitettu JavaScriptillä ja itse sovellus toimii React Native JavaScript kehyksellä, syntaksi on JSX:ää ja JavaScriptiä. Sovellus on suunnattu GBC Networkin asiakkaille ja se on yksi GBC:een tarjoamista palveluista. Sen pääfunktio on tallentaa sovellukseen koko yrityksen kaikki sisään tulevat ja ulospäin lähtevät puhelut kaikkien työntekijöiden nähtäville. Näin yritys pysyy ajan tasalla siitä, mihin vastaamattomiin puheluihin on soitettu takaisin, ja mihin ei, jotta vältetään toistuvat puhelut samasta aiheesta takaisin asiakkaille. Tämän lisäksi sovelluksessa käyttäjät voivat määrittää oman läsnäolostatuksensa, lisätä itsensä soittoryhmään ja soittaa puheluita sovelluksesta. Suurin osa työstäni on VAIHDE-sovelluksen kehittämistä, mutta satunnaisia webkehitys tehtäviä voi tulla tehtäväkseni.

Oppimisen tueksi olen valinnut kolme teosta

1. React.js Essentials, Artemij Fedosejev

Teos kattaa pohjustuksessaan syyt React ohjelmistokehyksen kehittämiseksi, sekä sen perusteet ja toimintatavat. Valitsin tämän teoksen, koska React Native on Reactin ohjelmointikehys, eli React Nativen toimintatavat pohjautuvat Reactiin. Alkuperäisen React ohjelmointikirjaston tunteminen ja ymmärtäminen on erittäin olennainen taito React Native kehittämisessä kehittymiselle. Laajoja websovelluksia kehitettäessä on suuri riski sille, että kun kehitysprosessi tuo yhä uutta kompleksisuutta sovellukseen uusien

toimintojen muodossa, monimutkaisuus ”riistäytyy käsistä” jolloin jatkokehittämisestä tulee ajan kuluessa vaikeampaa, ja sovellus ei enää skaalaudu yhtä nopeasti ja helposti. React.js Essentials esittelee React.js ohjelmointikirjaston ratkaisuna tähän ongelmaan.

2. Learning React Native, Bonnie Eisenman

Perusteos React Nativen ymmärtämiselle ja oppimisille. Sisältää kaiken peruskonsepteista haastavampiin pulmiin. Kirja ohjeistaa lukijaa esimerkkien avulla oivaltamaan React Nativen toimintaperiaatteet siten, että tiedon soveltaminen olisi mahdollisimman helppoa. Koska opinnäytetyöni keskiössä on React Native, valitsin tähän keskittyvän teoksen oppimisen tueksi.

Functional Programming in JavaScript, Dan Mantyla

Funktionaalinen ohjelmointi on ohjelmointiparadigma, joka nopeuttaa sovellusten vasteaikaa ja parantaa sekä käytettävyyttä että sovelluksen skaalautuvuutta. Sen toimintaperiaate on nimensä mukainen. Ohjelma pohjautuu funktioihin, joita ohjelmassa kutsutaan yhä uudelleen ja uudelleen tilanteenmukaisilla argumenteilla. Imperatiivinen ohjelmointi, jonka ongelmiin funktionaalinen ohjelmointi vastaa, on tapa kirjoittaa koodia vaiheittain. Tämä johtaa nopeasti keittokirjamaiseen koodiin (”Ota jauhot kaapista, mittaa 2dl jauhoja, laita ne kulhoon...”) joka paisuttaa sovelluksen koodia ja kokoa, jolloin sekä sovelluksen vasteaika kärsii ennen pitkää häiritsevän paljon, sekä tekee koodin kirjoittamisesta ja päivittämisestä erittäin aikaa vievää ja vaivalloista. Funktionaalinen ohjelmointi kertoo ohjelmalle yllä esitetyn esimerkin sijasta ”Tee X taikina”, jolloin sama taikinafunktio voidaan kutsua useita kertoja sijoittaen X:n kohdalle haluttuja argumentteja. Sen lisäksi että funktionaalinen ohjelmointi on erinomainen taito kenen tahansa koodarin työkalupakissa, ”VAIHDE”-sovelluksessa on jo useita eri toimintoja, joiden toteuttamisessa käytetään funktionaalista ohjelmointia, joten tämän hallitseminen on hyvä osata hyvin.

2 Lähtötilanteen kuvaus

2.1 Oman nykyisen työn analyysi

Suurin osa työtäni on VAIHDE-sovelluksen kehittämistä seuraavasti

1. Uusien toimintojen koodaus ja sovelluksen tehokkuuden optimointi. Saan esimieheltäni ohjeistuksen uusiin toimintoihin ja skripteihin, joiden implementointiin saan yleensä suhteellisen vapaat kädet. Ohjeistus voi olla hyvinkin lyhyt, saan itse keksiä miten toiminto toteutetaan. Toisinaan ohjeistus on tarkka ja yksityiskohtainen, kun haluttu tulos on myöskin hyvin tarkkaan määritelty. Noin 60% kaikista koodattavista toiminnoista vaatii käyttöliittymän koodaamisen lisäksi myös kommunikointia palvelimen, Google Firebasen kanssa. Kun toiminto liittyy sekä käyttöliittymään että palvelimeen on yleensä kyseessä jonkinlainen toiminto tai skripti, joka tallentaa tai muokkaa käyttäjän tietoja tai puhelulokeja.

Uusien toimintojen lisäksi pidän silmällä sovelluksen toimintanopeutta ja mahdollisia ”pullonkauloja” vasteajassa, joita saattaa ilmetä, jos uuden toiminnon koodissa on jonkinlainen hidastava toimintatapa.

Työssäni käytettävä ohjelmointikieli on JavaScript, joten sen peruseriaatteiden tunteminen on välttämättömyys työssäni. Koodi, virheilmoitukset ja internetistä löytyvät ohjeet ovat englanniksi, joten sujuva englannin luetun ymmärtäminen on ehdoton välttämättömyys. Ymmärtääkseni miten minun täytyy mitään taskia lähestyä, React Nativen perustuntemus on välttämättömyys, tai vähintäänkin kyky hahmottaa millä hakusanalla voin etsiä tehtäviin apua internetistä.

2. Ulkoasullisten seikkojen ideointi ja käytettävyyden parantaminen. Uusiin toimintoihin liittyy usein jonkinlainen käyttöliittymäelementti, jonka ulkoasu ja mahdollisen interaktion suunnittelen, toteutan ja sitten hyväksytän esimiehelläni. Ulkoasullisia seikkoja suunnitellessa ja toteuttaessa on tärkeää olla ajattelematta kuin koodaja, ja pyrkiä näkemään sovellus käyttäjän näkökulmasta. Itselleni sovelluksen toimintatavat, periaatteet ja ulkoasu ovat työtäni, joten on helppo unohtaa, että sovelluksen toiminnallisuus ei ole yhtä itsestäänselvää käyttäjälle. Ulkoasullisesti ja käytettävyydellisesti hyvien ratkaisujen tekeminen edellyttää myös graafista silmää. Ulkoasultaan sovellus on selkeä, se ei sisällä monimutkaisia animaatioita tai grafiikkaa.

3. Sovelluksen päivittäminen Androidin Play kauppaan ja iOS App storeen. Saatuani valmiiksi sen verran uusia ominaisuuksia sovellukseen, että asiakkaat saavat konkreettista arvoa sovelluksen päivittämisestä, buildaan sovelluksen ja lataan sen Play kauppaan tai App storeen, riippuen siitä kumman alustan versio sovelluksesta oli työn alla. Päivitysten lataaminen on minulla sujuvasti hallussa Androidille, koska prosessi on hyvin suoraviivainen. App storeen päivittäminen on hiukan mutkikkaampaa, tässä onnistuakseni tarvitsen tietämystä sekä ymmärrystä Xcode-ohjelmasta ja Applen vaatimuksista sovellukselle.

Omat taitoni ovat kehittyneet monipuolisesti sitten ensimmäisen harjoittelupäiväni. Olen oppinut paljon siitä, miten oikea mobiilisovellus toimii, miten sitä kehitetään, millainen on mobiilisovelluksen kehitysprosessi, sekä mitä yleisiä kompastuskiviä ja ongelmia mobiilikkehittäjä/React Native kehittäjä kohtaa päivittäin. Minulla on syventynyt käsitys, siitä miten JavaScript ohjelmointikielenä toimii, mitkä ovat sen vahvuudet ja sudenkuopat ja miten JavaScriptillä kannattaa lähestyä ohjelmointipulmia.

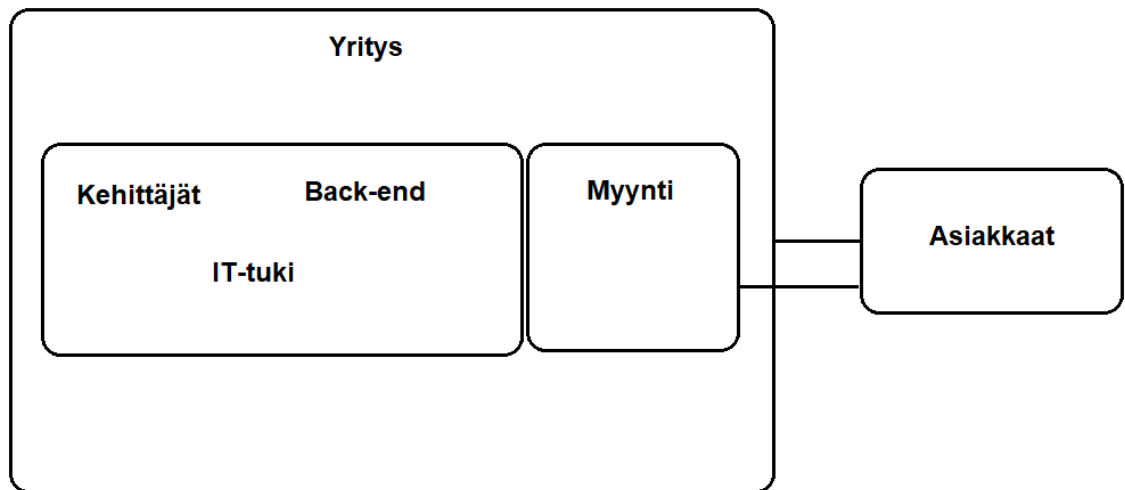
Yleensä selviän kohtaamistani ongelmista omatoimisesti. Suurin osa oppimisestani työharjoittelun aikana onkin tapahtunut juuri näin: olen törmännyt jonkinlaiseen ongelmaan, jota en osaa itse selvittää omalta tietoperusteeltani, joten etsin internetistä vastauksia ja koetan soveltaa niitä omaan ongelma-instanssiini.

2.2 Sidosryhmät työpaikalla

GBC Networks on pienikokoinen yritys, se työllistää 7 työntekijää. IT-tuki ja kehitystiimissä työskentelee neljä työntekijää, myynnissä yksi, ja palvelimien parissa kaksi.

Toimitusjohtaja vastaa kirjanpidosta, hallinnollisista asioista ja toimii asiakasrajapinnassa. Työpaikalla käytetään hyvin löyhää hierarkiaa: sekä IT, asiakas että yrityksen asioista neuvotellaan niiden työntekijöiden kesken, ketä asia koskee.

GBC:en työntekijöitä on melko hankala niputtaa tarkkoihin IT-talon kategorioihin: IT-tuessa työskentelevät työntekijät hoitavat myös laskutusta ja asiakastapaamisia. Oma esimieheni on lähin tukihenkilöni, hän oli työharjoitteluni ohjaaja ja täten ensimmäinen henkilö ketä konsultoin ongelmatilanteissa. Merkittävimmät ja ainoat sidosryhmäni ovat IT-tuki ja toimitusjohtaja, heiltä saan palautetta työstäni ja he välittävät asiakkaiden pyyntöjä ja huomioita minulle.



Kuva 1. Yrityksen sidosryhmät.

Kuvassa 1 on kuvattuna yrityksen rakenne ja opinnäytetyön kannalta olennaiset sidosryhmät. Laatikko "yritys", joka sisältää kaikki työntekijät, on jaettu kahteen alalaatikkoon. Alalaatikot koostuvat myynnistä ja kaikesta teknisestä toteuttamisesta, sillä ne ovat ainoat karkeat jaot vastuualueiden väleillä. Myynti on yhteydessä asiakkaisiin, ja asiakkaat ovat yhteydessä koko yrityksen myyntiin tai tekniseen tukeen tilanteesta riippuen.

Muut palvelu ja sovelluskehittäjät ovat lähimmät sidosryhmäni. Muut kehittäjät keskustelevat sovellusratkaisuista back-end koodaajien kanssa. Omassa työssäni käytän itsenäisesti Googlen Firebase back-endiä, joten työni ei koske muiden järjestelmien back-endiä. Myynti neuvottelee asiakkaiden ja potentiaalisten asiakkaiden kanssa myytävistä palvelukokonaisuuksista ja hinnoista. Kehittäjät ja Back-end koodarit toimivat IT-tukena tapauskohtaisesti. Virallista työntekijä hierarkiaa ei ole, joten useimmista asioista liittyen asiakkaisiin ja palveluihin päätetään yhdessä.

2.3 Vuorovaikutustaidot työpaikalla

Koodarina en toimi asiakkaiden kanssa suoraan, kommunikointi työpaikalla tapahtuu täysin työtovereiden kanssa. Työtoverini toimivat asiakasrajapinnassa, joten palautteet ja uudet työtehtävät liittyen asiakkaiden toiveisiin tulevat heidän kautta.

Suurin osa työstäni on itsenäistä koodaamista. Silloin kun kysyn jotain työhöni liittyvää työtovereilta, se liittyy useimmiten sovelluksen toimintaan. Todellisen kykyäni ylittävän ongelman ratkaisuun tarvitsen avunpyyntöjä melko harvoin. Useimmiten ongelmat joihin törmään itse koodaamisessa ovat sellaisia, että tiedän jo valmiiksi, että työtoverini ovat

samalla tasolla ongelman ratkaisemisen kanssa, sillä kenelläkään muulla työpaikallani ei ole varsinaisesti syvällistä React Native tuntemusta. Tämä on toisinaan haaste, sillä silloin olen ongelman kanssa yksin. Koodaaminen voi jäädä jumiin samaan ongelmaan jopa muutamaksi päiväksi, jolloin työnteko voi olla erittäin turhauttavaa. Nämä haasteet ovat kuitenkin oppimiseni kannalta hyvä asia. Koen oppivani parhaiten itsenäisesti kohtaamistani ongelmista.

3 Päiväkirjaraportointi

3.1 Seurantaviikko 1

Maanantai 07.05.2015

Päiväkirjan ensimmäiselle päivälle ei ole vielä tiedossa selkeitä tavoitteita. Viime perjantaina tutkin erilaisia npm:iä, joilla voisin asettaa sovelluksen elementeille varjoefektejä. Tällä hetkellä kaikki elementit ovat syvyydeltään täysmälleen samalla tasolla, joten ulkoasu ei ole kovin dynaaminen. Tavoitteenani on jatkaa varjo-liitännäisten tutkimista, mikäli en saa päivän alussa toista toimeenpanoa.

Töihin saapuessani en ehtinyt kauaa etsiä tietoa varjostuksen toteuttamisesta, kun toimitusjohtaja kävi näyttämässä, miten sovellus hidastelee Sony Experia-puhelimessa. Kyse oli käyttötapauksesta, jossa käyttäjä on juuri avannut sovelluksen ensimmäistä kertaa ja kirjautunut sisään. Tämän jälkeen sovellus näyttää käyttäjälle sovelluksen sisällön, mutta osa painettavista toiminnoista ei toimi välittömästi. Oma hypoteesini hitaudelle oli se, että koska käynnistyessä sovellus kutsuu useaa funktiota, jotka lataavat dataa firebasesta ja renderöivät sitä näkyville sitä mukaa, syntyy pullonkaula vasteaikaan. Tämä bugi oli minulla ennestään tiedossa, mutta se ei ennen tätä ole ollut tehtävänä.

Alanavigaatiopalkin ja muiden toimimattomien painikkeiden sijasta näkyvä muutaman sekuntin latausanimaatio, riittäisi siihen että kaikki on valmista käyttäjälle. Minun täytyi ensimmäiseksi selvittää, miten muokkaan (eli tässä tapauksessa piilotan) React Nativen navigaation propseja käyttämällä konditionaalisesti. Se, miten propseja tulisi oikeaoppisesti käyttää, oli epäselvää minulle, eivätkä React Nativen omat ohjesivut ohjeista kehittäjää perusteita syvemmin.

Konsultoin esimieheni mielipidettä asiasta, ja tulimme yhteispäätökseen että paras ratkaisu tilanteessa on ehdollistaa sivujen renderöinti, eli palauttaa käyttäjälle latausanimaatiosivu, joka vaihtuu sovelluksen aloitussivuksi. Tämä eroaa siten alkuperäisestä ideastani, että omassa ideasani palautan käyttäjälle suoraan sovelluksen aloitussivun elementit piilotettuna, kunnes tarvittava data on ladattu ja elementit renderöity. Tässä ratkaisussa ei piiloteta elementtejä, vaan hallitaan niiden rendautumista sivulle. Tarkoituksenani on siis kehittää sovellukseen seuraavanlainen logiikka:

Käyttäjän kirjaututtua annetaan sivujen ladata dataa ja käyttäjälle näytetään samanaikaisesti latausanimaatiota. Latauksen jälkeen käyttäjälle annetaan aloitusivu.

Tämä taski siirtyi kuitenkin suurimmilta osin seuraavalle päivälle, sillä suurin osa päivästä ehti mennä erilaisia ratkaisuja pohtiessa ja tietoa etsiessä. Päivän aikana ehdin lukea jonkin verran siitä, miten propsia käytetään oikein ja väärin. Props käsitteenä on yksinkertainen, se on lauseenomainen ilmoitus objektin ominaisuudesta, kuten kuva 2 havainnollistaa.

```
1 <elementti  
2 väri={punainen}  
3 koko={suuri}  
4 />
```

Kuva 2. Esimerkki elementin propsista pseudokoodina.

Punaisuus on pallon propsi ja itsessään se on helppo ymmärtää. Propsien käyttö monimutkaistuu kuitenkin nopeasti kun propsit abstraktoidaan ja niitä lähetetään komponenteilta toiselle, tai tässä tässä tapauksessa muutetaan konditionaalisesti.

Tiistai 08.05.2015

Aamulla aloitin siitä mihin jäin eilen. Tarkoitukseni oli muuttaa sivujen renderöintiä siten, että sivujen sisältö renderöidään vasta kun kaikki listaelementit ovat valmiita. Sitä ennen käyttäjä näkee vain latausanimaation. Tavoitteenani on saada tämä tehtävä valmiiksi. Vaikka kyseessä ei ole kriittinen sovelluksen kaatava bugi, kuvailisin sen prioriteettiä tärkeäksi. Salamannopeiden sovellusten maailmassa sekuntinkin hitaus voi tuntua käyttäjältä turhauttavalta. (miten edellinen lauseeni on oma sitaattini? Tai miksei se toimi omana lauseenaan)

Työskenneltyäni asian kanssa hetken, tulin siihen tulokseen että eilen keksimämme ratkaisu ei ole toteuttavissa. Latautuvien listojen sovellusnäkyminen on pakko palauttaa suoraan käyttäjälle, muuten listat eivät lataudu. Pohdimme ratkaisumahdollisuuksia esimieheni kanssa ja tulimme siihen yhteistulokseen, että miten ikinä ratkaisinkin ongelman emme halua sovellukseen enää yhtään lisää vaikeasti skaalautuvaa koodia. Sovellus pohja on tilattu freelancerilta, joka toimitti jokseenkin toimivan sovelluksen, mutta sovelluksen koodi ei ole toteutettu parhaita React Native käytäntöjä noudattaen.

Ainut metodi latausanimaation näyttämiseen hyviä toimintatapoja noudattaen, on näyttää animaatio kun käyttäjiä listaava map() metodi on valmis. Map() –metodia kuvaisin seuraavin sanoin ”Jokaiselle elementille arrayssä, tee tämä toiminto”.

```

1 var käyttäjäArray = [käyttäjä1, käyttäjä2, käyttäjä3....]
2
3 {Object.entries(käyttäjäArray).map(
4   listaa käyttäjä
5  )}
6

```

Kuva 3. Yksinkertaistettu pseudokoodi käyttäjien listaamisesta

Ensiksi tutkin internetistä, mitä eroa on map() ja forEach metodeilla. Tiesin että molemmat kertovat sovellukselle tehtävän, joka tulee suorittaa jokaisen arrayn elementin kohdalla, mutta merkitysero oli epäselvä. En saanut omia ratkaisujani toimimaan objektin sisällä, joten päätin etsiä lisää tietoa JSX objekteista ja niiden toimintatavoista.

Jotta en turhaan vaivaisi päätäni saman ongelman kanssa liian pitkään, esimieheni antoi minulle uuden tehtävän: node.js funktio, joka kuuntelee muutoksia firebase tietokannan tietyistä osista, ja muuttaa muutosten mukaan toista osaa. En varsinaisesti ehtinyt aloittaa funktion kirjoittamista, koska sitä ennen minun tuli koota olemassaolevat node.js funktiot omille tiedostoilleen, sen sijaan että kaikki funktiot olisivat samassa tiedostossa. Funktion kirjoittaminen jäi seuraavalle päivälle.

Päivän aikana ehdin lukea map() ja forEach() metodeiden eroista ja JSX syntaksista. Metodeista lukiessani selvisi, että ongelman ratkaisu ei piile varsinaisesti kummassakaan, kyse on siitä miten saan ratkaisuni kirjoitettua oikeaoppisesti JSX syntaksiin. Node.js taskia tehdessäni opin miten node.js funktiot asetetaan omille sivuilleen siten, että kaikki funktiot kokoavaan index.js tiedostoon ei keräännä liikaa koodirivejä. En varsinaisesti päässyt päivän tavoitteeseen, mutta sain selville että sen ratkaisemiseksi on olemassa vain yksi oikea tapa.

Keskiviikko 9.5.2018

Päivän tavoitteena on saada ratkaistua eilisen node.js funktio. Kypsyttelin eilen mielessäni ratkaisua sovelluksen hidasteluongelmaan, joten saatan työstää myös sitä jos sille liikenee aikaa.

Olen aiemmin toteuttanut vain muutamia node.js funktioita, joten sen toimintaperiaatteet ovat minulle suhteellisen tuntemattomia, vaikka itse koodikieli on tuttu. Toteutettavan funktion olisi määrä kuunnella kaikkia muutoksia tietyssä osaa tietokantaa (tietokannan

osa kuvassa 3). Muutos tietokannassa tarkoittaa tässä yhteydessä sitä, että kun yritykseen saapuu puhelu johon kukaan ei vastaa niin puhelun tiedot lisätään ensiksi tietokantaan. Kanta näin ollen muuttuu lisäyksen myötä, haetaan kannan toisesta osasta kaikista käyttäjistä ne käyttäjät, jotka ovat työntekijöitä siinä yrityksessä johon puhelu on tullut. Kun kaikki käyttäjät on haettu, heidän puhelimiinsa lähetetään ilmoitus, eli notifiointi vastaamattomasta puhelusta.

Ratkaisun toteuttamiseksi minun tulisi ottaa ainakin kaksi snapshotia tietokannasta kuvan 3 osoittamalla tavalla: ensiksi siitä osasta johon kirjataan vastaamattomien puheluiden tiedot ja toiseksi kaikista käyttäjistä. Jäin jumiin melko pitkäksi ajaksi tähän osaan taskista. Kun koetin lähettää firebaseen palvelimelle funktiota, se ei mene läpi, sillä lähetettävä funktio sisältää virheilmoituksen mukaan liika promiseja ja returneja. Selvittääkseni ongelman, etsin tietoa internetistä node.js ja firebaseen yhteistoiminnasta

En ehtinyt lukea kauaa, kun sain uuden toimeksiannon. Firebaseessa on kanta, johon kertyy numeerisesti vastaamattomien puhelujen määrä sitä mukaa kun puheluita jää vastaamattomiksi. Minun tuli koodata VAIHDE-sovelluksen "vastaamattomat puhelut" kohtaan muutama rivi koodia, joka miinustaa yhden vastaamattoman puhelun kaikista vastaamattomista puhelusta aina kun vastaamattomaksi jääneen puhelun soittajalle soitetaan takaisin. Tehtävä oli helppo, ehdin saada sen valmiiksi alle puolessa tunnissa. Tämän jälkeen buildasin sovelluksen ja päivitin sen Play kauppaan.

Edellämämainitua toimintoa tehdessäni sain uuden idean hidasteluongelman ratkaisemiseksi, ja käytin suurimman osan loppupäivästä sen kanssa työskentelemiseen, sillä sen prioriteetti oli korkeampi kuin node.js funktio. Artemij Fedosejev jakaa kirjassaan React.js essentials (2015,65) komponenttien elämänkaaren vapaasti suomennettuna kolmeen osaan: lisäämiseen, päivittämiseen ja poistamiseen. Tämä tarkoittaa käytännössä sitä, että jos koodari haluaa kutsua funktiota esimerkiksi juuri kun elementti tulee käyttäjän näkyville, hän voi tehdä sen komponentin `componentDidMount` (komponentti poistettiin) funktiossa.

Kokeilin, voisinko linkittää latausanimaation yksinkertaisesti React Nativen `componentDidMount()` funktioon, jota kutsutaan heti kun komponentti, eli tässä tapauksessa kun "Yhteystiedot"-sivu on ladannut. En harmikseni saanut ongelmaa ratkaistuksi: `componentDidMount()` kutsuttiin liian aikaisin, sillä komponentin sisällä latautuvat elementit olivat vielä kesken. Pulma jäi vielä toistaiseksi odottamaan uusia ratkaisuideoita.

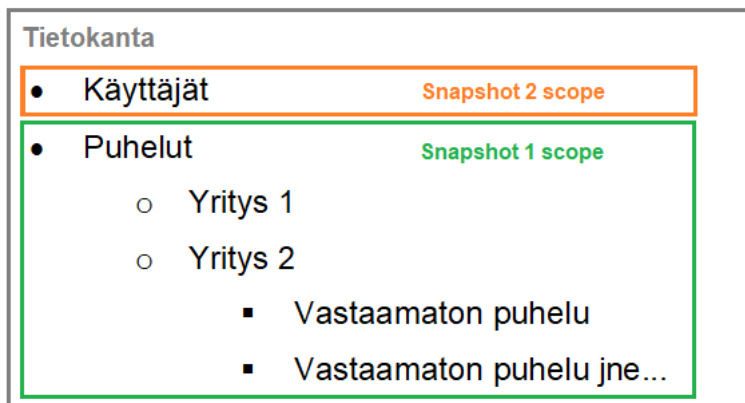
Torstai 10.5.2018

Helatorstai, ei töitä.

Perjantai 11.5.2018

Päivän tavoitteena on keskiviikkona aloitetun node.js funktion jatkotyöstäminen. Viimeksi jäin jumiin pitkäksi aikaa samaan ongelmaan, jossa en saanut haettua kaikkea tarvitsemaani dataa firebasesta. Uskon että saan sen päivän aikana selvitettyä.

Ongelmana oli etten saa lähetettyä firebasen palvelimelle funktiota, jossa on sisäkkäin kaksi snapshottia. Sain ensimmäisen snapshotin datan helposti, mutta en keksinyt miksen saa samasta snapshotista dataa tietokanta "käyttäjät" osiosta. Tovin tietoa etsittyäni selvisi ensiksi, että ensimmäisen snapshot scopesta ei ole pääsyä haluamaani osaan tietokannasta. Vaikka ensimmäinen snapshotti "puhelut"-kannasta sisälsi dataa juuresta, snapshotti ei kuitenkaan koskenut sitä. En vielä kukaan ymmärtänyt miten toiseen scopeen tulisi viitata.



Kuva 4. Kahden snapshotin scopet.

Internetistä tietoa etsiessäni löysin ratkaisun. Rivi, jolla viitataan kantaan molemmissa ei käy molempiin snapshotteihin, sillä ensimmäinen snapshotista triggeröityy kuuntelijasta, toinen ei. Käyttämäni rivit olivat:

4. `exports = module.exports = functions.database.ref('/puhelut/{firma}).onWrite(event`
5. `amin.database().ref('/users/').once('value').then(snapshot`

Ero näiden kahden rivin välillä oli se, että toinen niistä käyttää admin SDK:ta. Ratkaisuni toimi, mutta en ymmärtänyt miksi. Kysymysmerkille jäi, olivatko admin tason oikeudet tarpeelliset toisen snapshotin ottamiseksi. (täydentyy, tarvitsisin muistiinpanoni)

Päivän päätteeksi olin ehtinyt lukea paljon tietoa node.js funktioista. Pääsin kärryille, miten node.js funktioissa käytetään return keywordia ja miten eri osista kantaa haetaan dataa. Tutkiessani node.js funktioita, törmäsin kahteen uuteen hyödylliseen käsitteeseen: "callback" ja "callback hell". Functiona programming in JavaScript (2015, 143) kirjassa määritellään callback seuraavasti: Callback on funktio, joka voidaan lähettää toiselle funktiolle myöhempää käyttöä varten. Sekä VAIHDE-sovellus ja kirjoittamani node.js funktio käyttävät runsaasti callbackeja, ja olenkin kirjoittanut sellaisia funktioita tietämättäni. Callback ei itsessään ole JavaScriptin avainsana, tai edes erityinen metodi, se on vain tapa kirjoittaa JavaScriptiä. Callback on yleinen tapa kirjoittaa koodia "ylhäältä alas" tapahtumajärjestyksessä. Kyseessä on siis osittain tapa visualisoida ja kirjoittaa koodi tarinan omaisesti. Tavassa on kuitenkin ongelma: callbackien yltiöpäinen käyttö johtaa hankalasti luettavaan koodiin, jota on hyvin vaikea skaalata ja muokata.

```
1 funktio(){
2     tee tämä, ja sitten(funktio){
3         tee tämä, ja sitten(funktio){
4             tee tämä, ja sitten(funktio){
5                 jne...
6             })
7         })
8     })
9 }
```

Kuva 5. Pseudokoodi esimerkki callback hell:istä.

Kuten ylläoleva kuva havainnollistaa, callback funktioiden käyttö tekee koodista aluksi loogista, helposti muokattavaa ja ymmärrettävää. Kuvan 5 funktiossa on 3 callback kerrosta: Rivillä 1 kutsutaan pääfunktioita, joka päättyy riville 9. Toisella rivillä kutsutaan ensimmäistä callback funktiota, joka päättyy riville 8. Kolmannen rivin callback loppuu riviin 7 ja vastaavasti neljännen rivin viimeinen callback funktio päättyy riville 6. Pääfunktio sisältää kolme callback funktiokerrosta.

Esimerkin kolmitasoinen callback logiikka on vielä hyvien toimintatapojen mukaista, mutta mitä enemmän callback kerroksia syntyy, sitä vaikealukuisempaa siitä tulee: syntaksivirheitä voi olla hankala löytää, ja alun helppolukuisuuskin katoaa. Jos keskitason callback funktioita tarvitsee muokata, alemman tason funktiot voivat tuottaa bugeja.

Vaihtoehto callbackeille, eli asynkroniselle ohjelmoinnille on koodin modulariteetin lisääminen. Modulariteetti tässä yhteydessä tarkoittaa funktionaalisen ohjelmoinnin tapaa jakaa koodi uudelleen käytettäviin osiin eli moduuleihin. Pienistä moduuleista voidaan luoda suurempi moduulikokonaisuus, jolloin koodi ei ole tarinamuotoista vaan segmentoitua. Modulaarisessa koodissa debuggaus ja muokkaaminen on helpompaa: sen sijaan että koko sovellus tuottaa epämääräisen virheilmoituksen, kehittäjä näkee mikä osa koodista tuottaa virheen.

Viikkoanalyysi

Viikon teemoina olivat suurimmilta osin React Nativen renderöinti ja node.js kirjoittaminen. Ehdin töitä tehdessäni hankkia jonkin verran uutta tietoa React Nativen perusteista, kun selvitin maanantaina ja tiistaina raportoimistani ongelmista. Osaamiseni kehittyi uusien konseptien (callback, callback hell) ja JavaScript metodien (Object.entries, .map(), Promise.all) muodossa. Kehityksen voi siis jakaa kahteen osaan: laajemmat, kauttaaltaan ohjelmointiin vaikuttavat konseptit ja yksittäiset, tapauskohtaiset metodit. Lisäksi opin lisää jo jokseenkin tutusta komponenttien elämänkaari-konseptista, sekä firebasen ja node.js yhteistoiminnasta.

React Native komponenttien elämänkaari-konsepti on yksi olennaisimmista React ja React Nativen ominaisuuksista. Bonnie Eisenman kuvaa kirjassaan ”Learning React Native” miten koodari voi kirjoittaa erilaisia toimintoja (funktioita) käynnistymään, kun jokin komponentti on tulossa käyttäjän näkyville, tai poistumassa näkyviltä. Ennen tämän päiväkirjan kirjoittamisen aloittamista tämä konsepti oli outo minulle. Päiväkirjan edetessä yksi tavoitteistani on pitää silmällä sitä, miten parhaiten käyttää hyödyksi elämänkaareen liittyviä teknisiä seikkoja.

Erityisesti React Nativella työskennellessäni huomasin, miten ongelmien ratkaisun haasteina ei ole erityisesti jokin toimintamalli jota käytän, vaan ennemminkin toimintamallien puute. Ennen kuin siirryin kehittämään VAIHDE-sovelusta, en ollut lukenut mitään React Nativen perusteista tai toimintaperiaatteista, aloitin suoraan tarkastelemaan koodia ja tein siihen intuitiivisesti muutoksia. Tämä johtaa usein siihen, että kohtaamani ongelmat selkeytyvät hitaammin, ja debugatessa kestää kauan.

Suurin osa viikon töistä kului korjatessa erilaisia vikoja. React Nativen vajavainen tuntemus johtaa usein siihen, että sovellus palauttaa jonkin yleisen virheilmoituksen, jonka ympäriltä löytyy keskustelua internetfoorumeilta, mutta en silti osaa korjata koodiani.

Kenelläkään ei ole juuri sitä versiota virheestä, joka omassa koodissani on, joten muilta kehittäjiltä saadun tiedon soveltaminen on vaikeaa. Sujuva soveltaminen edellyttää perusteiden tuntemusta. Internetistä työpäivän aikana etsimäni tieto on usein hyödyllistä, mutta satunnaiset kommentit keskustelufoorumeilla avaavat itse Reactia pintapuolisesti.

Tulevilla viikoilla tavoitteenani onkin jokaisen ongelman kohdalla löytää perimmäinen toimintatapa josta yksittäiset virheilmoitukset kumpuavat. Viikon töiden perusteella aion perehtyä syvällisemmin erityisesti propseihin, stateen ja komponenttien elämänsykliin, jotta tulevaisuudessa olen paremmin perillä siitä, miten mihinkin virheilmoitukseen tulisi reagoida.

3.2 Seurantaviikko 2

Maanantai 14.05.2015

Maanantain tavoitteena on työstää node.js funktiota mahdollisimman lähelle valmista funktion kanssa, ennen kuin jatkan töitä sovelluksen parissa.

Muutamien työtuntien jälkeen funktiota vaadittu data ilmoituksen lähettämiseen oli kerättyä: yrityksen nimi, soittajan puhelinnumero ja tokenit. Asetin datan ilmoitusfunktioon ja aloitin funktion testaamisvaiheen. Firebasen konsoli, johon logitin tietoa funktion eri vaiheista palautti viestin notifiointin lähetyksen onnistumisesta, mutta en kuitenkaan saanut notifiointia laitteeseeni. Tämän bugin selvittämiseen minun piti soveltaa firebase tuntemustani, sillä ilman selkeää virheilmoitusta on mahdotonta sanoa johtuuko virhe firebasesta, sovelluksesta, laitteestani vai nodesta.

Luin firebaseen dokumentaatiosta miten adminilla lähetetään notifiointia.

Dokumentaatiot eivät auttaneet, sillä tein kaiken niiden mukaan oikein. Firebaseen oman testisivun kautta onnistuin lähettämään viestejä suoraan laitteeseeni, joten tiesin virheen johtuvan jostain muusta kuin laitteestani. Asiaa tutkittuani selvisi, että firebaseen lähettämäni objekti, joka sisältää notifiointiviestin, oli puutteellinen. Pelkän viestin lisäksi objektiin piti lisätä data-ominaisuus kuvan 6 osoittamalla tavalla. Koodin 3 rivi puuttui objektista. Firebaseen dokumentaation esimerkeissä ei ollut mainintaa data ominaisuuden välttämättömyydestä. Minulle ei selvinnyt oliko dokumentaatio puutteellinen vai oliko omassa funktiossani jokin ominaisuus, jonka vuoksi dokumentaation mukainen notifiointi ei ilmestynyt laitteeseeni.

```
1 notifiikaatio = {  
2   viesti: {"viestin sisältö"},  
3   data: {}  
4 }
```

Kuva 6. Pseudokoodi notifiikaatio-objektista

Kuva 6. kuvaa notifiikaatio objektia, jolla on kaksi ominaisuutta: viestiominaisuus ja dataominaisuus. Sain funktion toimimaan siltä osin, että notifiikaation lähettäminen omaan laitteeseeni onnistui. Seuraavaksi koodia piti muokata siten, että korvaan oman laitteeni tokenin arraylla, johon lisätään yrityksen jokaisen työntekijän tokeni. Tämä vaihe oli nopeasti tehty, sillä olin viime viikolla saanut valmiiksi snapshot funktiot, joista tarvittavat tokenit sai arrayhyn yksinkertaisella forEach()d kutsulla. Päivän aikana ehdin debuggauksen yhteydessä lukea paljon notifiikaatioiden lähettämisestä node.js funktioilla. Firebasen ja node.js yhteistoiminta on minulle nyt paljon selkeämpi ja koen että tulevaisuudessa osaan tehdä vastaavia funktioita paljon nopeammin.

Tiistai 15.5.2018

Tavoitteenani on tehdä muutamia muutoksia node.js funktioon ja se on valmis. Sen jälkeen pyydän uutta toimeksiantoa, palaan todennäköisesti sovelluksen pariin selvittämään viimeviikkoista hidastelua.

Ehdin muutaman hetken tarkastella funktiota, kun sain uuden toimeksiannon. Minun tuli ottaa screenshotteja sovelluksesta markkinointia varten. Screenshoteista piti muokata nimet siten, ettei mainoksiin tule asiakkaiden tai työntekijöiden puhelinnumeroa tai oikeita nimiä, jotka näkyvät esimerkiksi yhteystiedot sivulla. En päässyt aloittamaan tehtävää, kun etätöitä tekevä esimiehi soitti kertoakseen sovelluksen kaatumisesta. Asiakas oli ilmoittanut, että jos käyttäjä on kirjutuneena ulos sovelluksesta ja vastaanottaa normaalin puhelun, sovellus saattaa kaatua.

Testatakseni vikaa, minun piti buildata, eli koota sovellus puhelimeeni. Buildaaminen onnistui, mutta sain virheilmoituksen React Nativen ja JavaScriptin yhteensopimattomuudesta. En ehtinyt selvittää virheilmoitusta kauaa, kun sain kuulla jälleen uudesta virheestä sovelluksessa: asiakkaan Samsung Galaxy J3-puhelimessa sovellus ei käynnisty ollenkaan. Buildasin sovelluksen apk-tiedostoksi, ja latasin sen bitbariin, mutta bitbar ilmoitti sovelluksen toimivan oikein. Koska virheen toistaminen testiympäristössä ei onnistunut, palasin takaisin selvittämään "mismatch" virheilmoitusta.

Juuri ennen työpäivän päättymistä sain yhteensopimattomuus virheilmoituksen korjattua: sovellus koetti avautua ikäänkuin uudempana React Native versiona kuin se oli. Löysin yksittäisen kommentin erään keskustelufoorumin ketjusta, jossa vian korjaaminen neuvottiin. Ilman tätä kommenttia olisin voinut jäädä ongelmaan jumiin pitkäksi aikaa, sillä vastaavaan vikailmoitukseen löytyi internetistä useita vastauksia, jotka eivät kohdallani toimineet. Ongelma liittyi 15.5.2018 (eli tänään) lisättyyn react native maven päivitykseen, joka antaa kaikille React Native buildeille virheilmoituksen. Vian korjaamiseen riitti vain yksi rivi koodia, jossa React Native ja JavaScript pakotettiin toimimaan samasta versiosta vaikka uudempi versio olisi saatavilla. Keskustelufoorumin ohjeen mukaan sovelluksen manifesti-tiedostosta piti muuttaa kaksi arvoa vastaamaan toisiaan. Ratkaisun löytyminen oli silkkaa tuuria. Vaikka virhe oli yksittäistapaus, se liittyi puutteeseen osaamisessani, React Nativen buildaaminen ja buildaamisen periaatteet. Törmään työssäni noin kerran kuukaudessa virheeseen sovelluksen buildaamisessa, jota en osaisi selvittää ilman apua Internetistä ja työtovereiltani.

Keskiviikko 16.5.2018

Keskiviikkona tavoitteenani on saada node.js funktio vihdoin valmiiksi, sekä saada pyydetyt screenshotit lähetettyä.

Noin muutamassa tunnissa sain otettua, muokattua ja lähetettyä screenshotit. Kuvista piti editoida oikeat puhelinnumerot ja nimet esimerkkinumeroiksi (0401234567) ja nimiksi (Matti Meikäläinen). Kuvien editoinnin suoritin Gimpillä, joka ei ole minulle ennestään tuttu ohjelma. Tehtävän aikana opin, miten Gimpillä siirretään ja poistetaan valittuja alueita, lisätään kuvaan tekstiä ja viedään kuva esimerkiksi png muotoon. Kuvien lähettämisen jälkeen siirryin node.js funktion pariin.

Node.js funktio valmistui suurimmiltaan työpäivän loppuilla, mutta se ei ollut valmis päivitettäväksi asiakkaille. Kun notifiikaatiot otetaan käyttöön seuraavassa päivityksessä, käyttäjällä täytyy olla asetuksissa valokatkaisijanomainen kytkin josta käyttäjä saa notifiikaatiot pois päältä. Tämä vaikuttaa väkisinikin tekemääni funktioon, sen täytyy etsiä käyttäjien tokenien lisäksi ne käyttäjät, jotka ovat määrittäneet notifiikaatiot näkyväksi.

Päivän aikana tutustuin gimpin perusteisiin, jokseenkin pintapuolisesti, sillä tekemäni editoinnit eivät vaatineet layereiden käyttöä, joka on yksi kuvankäsittelyn perustaidoista. Node.js funktiota kirjoittaessani opin miten firebasesta haetaan dataa sekä snapshotin sisältä että ulkoa, joten tulevaisuudessa voin keskittyä enemmän itse funktion logiikkaan, kuin uhrata paljon aikaa pelkän datan hakemiseen ja manipuloimiseen.

Torstai 17.5.2018

Torstaina tavoitteenani on saada valmiiksi notifi kaatioiden back-end ja front-end. Sovelluksen asetukset sivulle pitää saada kohta, josta käyttäjä voi hallita notifi kaatioiden näkymistä puhelimesta päälle/pois-tyyppisellä kytkimellä. Kytkimen säätäminen tulisi asettaa kyseisen käyttäjän kohdalle firebaseen tieto notifi kaatioiden näkymisestä.

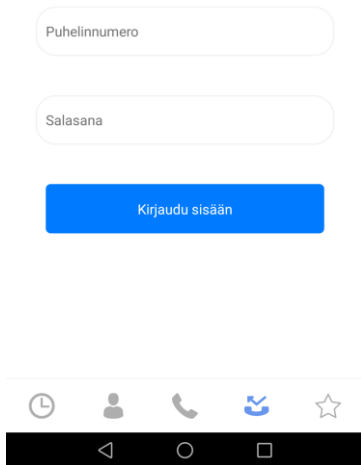
Päivän päätteeksi sain node.js täysin toimivaksi firebaseen. Ehdin tehdä myös sovelluksen asetuksiin kohdan, josta käyttäjä voi asettaa ilmoitukset pois päältä. En päivittänyt sovellusta vielä Play kauppaan, sillä seuraavaan päivitykseen pitää saada asetuksen lisäksi myös muita toimintoja ja mahdollisia korjauksia. En myöskään yhdistänyt node.js funktiota ja sovelluksen front-endiä. Esimieheni mukaan oli tärkeämpää saada notifi kaatiot toimimaan kaikille käyttäjille ensin, jotta asiakkaat soittaisivat puheluita sovelluksella mahdollisimman paljon. Tämä tarkoittaa käytännössä sitä, että kaikille asiakkaille näkyvät notifi kaatiot perusolettamuksena, ja käyttäjä voi halutessaan estää notifi kaatiot android puhelimiensa asetuksista.

Perjantai 19.5.2018

Perjantaille ei ole selkeitä tavoitteita tai tehtäviä. Aion kuitenkin tarkistaa, että eilen firebaseen lähettämäni funktio toimii odotetulla tavalla ja sen jälkeen palata sovelluksen pariin.

Huomasin, että eilen firebaseen lähettämäni node.js funktio ei lähetä kaikkia viestejä onnistuneesti. Tutkittuani asiaa hetken selvisi, että kaikilla asiakkailla ei ole ajan tasalla olevaa tokenia, joten notifi kaation lähettäminen firebaseesta epäonnistuu. Tämä ei vaatinunut minulta kuitenkaan toimia, sillä asiakas saa päivitetyn tokenin aina kirjautuessaan sisään, jolloin notifi kaation lähettäminen alkaa toimia. Jos asiakas ei ole kirjautunut sisään, hän ei näkisi notifi kaatioita muutenkaan. Funktio toimi odotetulla tavalla.

Siirryin VAIHDE sovelluksen pariin. Tarkoitukseni on selvittää navigaatioon liittyvä ulkoasullista ongelma: kun käyttäjä kirjautuu ulos, alanavigaatiopalkki jää edelleen näkyviin. Sen kautta hän voi "pyyhkäistä" eri sivuille jotka ovat kirjautumissivu. Tämä on ulkoasullisesti huono ominaisuus (paremman sanan puutteessa), joka laskee koko sovelluksen vakuuttavuutta.



Kuva 7. Kuvakaappaus kirjautumissivusta ja siinä virheellisesti näkyvästä alanavigaatiosta.

Alanavigaatio tarkoittaa kuvan 7. alaosassa näkyvän mustan osan yläpuolella olevaa leveää osiota, jossa on 5 symbolia eri navigaationsivuja varten.

Kyseessä ei ole koodin bugi, ominaisuus johtuu sovelluksen sivujen rakenteesta. Kaikkien sivujen logiikka on seuraavanlainen, jos käyttäjä ei ole kirjautunut sisään, näytetään kirjautumissivu. Tämä tarkoittaa sitä, että navigaation näkyminen ei ole sidottu siihen onko käyttäjä kirjautunut vai ei, joten ensimmäisen kirjautumisen ja uloskirjautumisen jälkeen navigaatiopalkki näkyy käyttäjälle aina kuvan 7 osoittamalla tavalla.

Päivän aikana varmistin node.js funktion toiminnan ja luin paljon React Nativen navigaatiosta. Tutustuin siihen miten alanavigaatiopalkki liitetään sovelluksen juureen app.js tiedostoon. Vaikka sovelluksen navigaationsysteemi on koodattu kehittäjän kannalta hankalaksi, se toimii käytännössä käyttäjän kannalta sulavasti. Ainoa ongelma on kuvassa 4 kuvattu ulkoasullinen seikka, jonka selvittämisessä etenin jonkin verran tänään. Navigaatio on mahdollista piilottaa yhdellä komennolla, mutta en vielä ratkaissut miten saan kyseisen komennon toimimaan oikein.

Viikkoanalyysi

Viikon aikana tehtävien kirjo oli melko laaja: se sisälsi kuvanmuokkausta, front- ja back-end koodausta ja virhekorjausta, virhesimulaatioiden läpikäyntiä ja sovelluksen buildaamisen debuggausta. Koen, että opin jokaisesta tehtävästä jotain. Tärkeimmät

oppimiskokemukset koskivat React Nativen buildausta ja navigaatiota sekä node.js funktioiden kirjoittamista. Tulevaisuudessa node.js ja firebasen yhteistoiminnallisten funktioiden kirjoittaminen on minulle nopeampaa, sillä osaan nyt hakea funktion vaatimusten mukaisen datan firebasesta sujuvammin. Aikaa jää enemmän itse funktion logiikan suunnitteluun ja koodamiseen, kun perus datan haku- ja kirjoitusmenetelmät ovat osa taitojani. React Nativen navigaatiojärjestelmä on minulle viikon töiden jälkeen tutumpi ja hahmotan nyt koko VAIHDE-sovelluksen rakenteen paremmin ja uskon, että keksin ensiviikolla miten ratkaisen pitkään vaivanneen ulkoasuongelman (kuva 5).

Kokemattomuuteni React Nativen buildaussäännöistä ja periaatteista ilmeni tiistaina, sillä noin 80% päivän työstä kului ongelman selvittämiseen, jonka ratkaisu oli lopulta vain yksi rivi koodia, jossa React Native pakottaa käyttämään samaa JavaScript versiota. Vastaavat tapaukset eivät ole koodaamisessa harvinaisia, yksi virheellinen merkki riittää lopettamaan minkä tahansa sovelluksen toiminnan. Oman työni sujuvuuden kannalta olisi hyvä, jos oppisin hahmottamaan mitkä buildaamisen vaiheet luovat tietynlaisia virheilmoituksia, jotta debugaamiseen menisi huomattavasti vähemmän aikaa. React Nativen buildaaminen on monimutkainen prosessi, johon liittyy useita vaiheita. Viikon aikana ehdin tutustua minulle täysin uuteen konseptiin ja termiin, maveniin, joka on tärkeä elementti buildaamisessa. Maven on työkalu, joka hallitsee sovelluksen sisäisiä riippuvuuksia. Riippuvuuksilla tarkoitetaan sitä, että esimerkiksi tietty versio jostakin palvelusta riippuu toisen palvelun versiosta.

3.3 Seurantaviikko 3

Maanantai 21.5.2018

Maanantaille ei ole selkeitä tavoitteita. Jatkan töitä sovelluksen navigaation parissa.

Viime viikon ratkaistava ongelma oli navigaation näkyminen käyttäjälle silloin kun sen pitäisi olla piilotettu. Sovellus renderöi alanavigaation silloin kun käyttäjä on kirjautunut ulos, jolloin käyttäjä voi edelleen navigoida pyyhkäisemällä oikealle ja vasemmalle.

Ensiksi koetin ratkaista, miten voisin asettaa konditionaalisesti jokaisen tabnavigaattorin sivun näkymättömäksi. Jokaiselle tabnavigaattorin tabille voi asettaa ominaisuuden "tabNavVisible: boolean", joka hallitsee näkykö navigaattori käyttäjälle. Etsin tietoa internetistä tämän ominaisuuden konditionaalisesta muokkaamisesta. Kävi ilmi, että muokkaaminen on mahdollista state-objektin avulla, mutta ratkaisu ei toiminut kohdallani. Konditio jonka mukaan olisin muokannut tätä ominaisuutta, oli ulkona navigaattorin

scopesta, joten idea ei sellaisenaan toiminut. React Native navigaattorin dokumentaatiot auttoivat ymmärtämään vain perusteet navigaattorin toiminnasta, mutta eivät antaneet syvempää ymmärrystä ongelmani ratkaisemiseksi.

Työpäivän päätteeksi sain ratkaistua ongelman erilaisella lähestymistavalla. Sen sijaan että olisin piilottanut tabnavigaattorin pois näkyviltä, kokeilin voiko navigaattoria jättää rederöimättä konditionaalisesti. Ideani toimi ja sain navigaation näkymään halutulla tavalla. Ero ensimmäiseen ideaan oli se, että ensimmäisen ratkaisun logiikka oli seuraavanlainen: "renderöi alanavigaatio, mutta piilota se kunnes käyttäjä on kirjautunut sisään". Toimivassa ratkaisussa navigaatiota ei piilotettu, vaan navigaattorin renderöinti ehdollistettiin: "renderöi navigaatio ja sivu ainoastaan, jos käyttäjä on kirjautunut sisään, muutoin renderöi pelkkä sivu"

Suurin osa päivästä kului navigaation dokumentaatiota lukiessa. Kun vertasin dokumentaatioiden esimerkkejä sovellukseen, opin hahmottamaan paremmin sovelluksen navigaation logiikan. Huomasin myös, että navigaatio ei ole tehty parhaiden käytäntöjen mukaan, sillä kaikki muutosmahdollisuudet navigaattorin ominaisuuksiin (kuten esimerkiksi tabnavigaattori piilottamiseen) olivat ulkona tarvittavasta scopesta. Kun koetin siirtää navigaattoreita scopeen, törmäsin useisiin virheilmoituksiin, joiden selvittämiseen olisi mennyt paljon aikaa.

Tiistai 22.5.2018

Tiistain tavoitteena on tutkia ratkaisuja sovelluksen hitaalle käynnistymiselle. Aamulla tarkoitukseni on tarkastella firebaseen indexointisääntöjä, mikäli niistä olisi apua.

Selvisi, että sovelluksen back-end oli jo indexoitu firebasesta. Ideani oli, että jos haettavaa dataa ei olisi indexoitu, se olisi voinut olla hitauden syy. Palasin työskentelemään koodin pariin. Muutamien kokeilujen jälkeen tulin samaan päätelmään kuin viimeviikolla: hitautta ei voi korjata koodista siten, että koodi pysyisi helppolukuisena ja skaalautuvana. Buildasin sovellukseni puhelimeeni tarkastellakseni vasteaikaa. Vaikka testipuhelin ei ole tehokas, sovelluksen vasteaika oli hyvä. Häiritseviä hitauksia on toistaiseksi tullut vastaan vain esimieheni Sony Experiassa.

Tämä nostaa esiin yhden mobiilikehittämisen teemoista: android fragmentaatio, eli se miten Android versioiden väleillä on kehittämiseen vaikuttavia eroja. Android versioita on tämän päiväkirjamerkkinnän kirjoittamisen aikaan julkaistu kahdeksan, ja versioiden (1.0, 2.0, 3.0...) välillä on useita väliversioita (1.2, 2.3...). Open Signalissa 2018 julkaistun

artikkelin mukaan erilaisia Android laitteita on jo yli 24 000. Tämä vaikeuttaa kehittäjän työtä. Sen lisäksi että sovelluksen tulisi toimia virheettömästi eri android versioiden välillä, sen tulisi olla optimoitu mahdollisimman useille android laitteille. Lucas M. kirjoittaa artikkelissaan "Google attacks Android fragmentation, pushes use of newer API, OS versions", että vuonna 2018 Google muuttaa vaatimuksiaan Android sovelluksille siten, että tuki tietyille vanhentuneille sovelluskomponenteille lakkautetaan. Tämä vaikuttaa konkreettisesti työhöni ja siihen, millaista koodia koodaan. VAIHDE sovellukseen saattaa tulla jatkossa ominaisuuksia, joissa ei voida hyödyntää vanhentuneita kirjastolaajennuksia.

Fragmentaatio ilmenee VAIHDE sovelluksen kohdalla suoritusnopeuden lisäksi esimerkiksi visuaalisten elementtien skaalautumisessa. React Nativessa elementit skaalautuvat, jos koodari on noudattanut hyviä käytäntöjä, suhteessa toisiinsa, ei suhteessa näytön pikselimäärään. Tietyllä näyttökoolla elementit skaalautuvat oikein, mutta toisella laitteella elementtien suhdanteet saattavat skaalautua odottamattomalla tavalla. Pahimmassa tapauksessa väärin skaalautuvat elementit tekevät sovelluksen käytön mahdottomaksi.

Keskiviikko 23.5.2018

Keskiviikolle ei ollut tiedossa tehtäviä.

Aamulla sain tehtäväkseni kehittää GBC takaisinsoittopalvelua, joka on web-pohjainen palvelu GBC:een asiakkaille. Tehtävän ensimmäinen koski äänitteen lataus-painikkeen ulkoasua. Mallista kopioitu painike rendautui virheellisesti domiin. HTML input-tagin koodaajien keskuudessa tunnettu huonoista muokkaus mahdollisuuksista, sillä sen ulkoasu määräytyy selaimen mukaan. Yleensä HTML tagien ulkoasua muokataan CSS kielellä, mutta HTML sisältää poikkeuselementtejä joita voi muokata rajatusti, tai ei ollenkaan. Poikkeuselementtien kohdalla koodaajan täytyy keksiä tilannekohtaisesti sopiva ratkaisu, jossa elementin ympärille suunnitellaan muista elementeistä tapa ohittaa poikkeus elementin rajoitukset.

```
1 <tekstilinkki inputille>
2     <painike>
3         <input tyyli="piilotettu" />
4         Valitse tiedosto
5     </painike>
6 </tekstilinkki inputille>
```

Kuva 8. Pseudokoodi input painikkeelle.

Ratkaisin ongelman kokeiltuani muutamia ratkaisuja. Kuvassa 8 on kolme elementtiä. Uloin elementti (alkaen riviltä 1, päättyen riviin 6) on näkymätön teksti, joka on yhdistetty sisimpään input elementtiin (rivit 3 ja 4) siten, että kun tekstielementtiä painetaan, sisimmän elementin toiminto käynnistyy. Toiminto avaa ikkunan käyttäjän tietokoneen tiedostoihin, joista hän voi valita haluamansa tiedoston. Uloimman ja sisimmän elementin välissä (rivit 2 ja 5) on painike elementti, joka on kaikista elementeistä ainut käyttäjälle näkyvä elementti. Yhdistämällä kuvan 8 koodilla kolme elementtiä sain kuva 9 osoittaman tuloksen. Kuvan 9 oikealla puolella on ratkaisuni tulos.

Normaali input-elementti

Kuvan 6 osoittaman koodin tulos

No file selected.

Kuva 9. Normaalin input-elementin ja kuvan 8 koodin tuottaman painikkeen vertailu.

Tehtävä haastoi ajattelemaan HTML:lää uudella tavalla. HTML ei ole vain sivun rakenteen kuvaamista varten, sillä voidaan tilannekohtaisesti suunnitella ulkoasullisia ratkaisuja, joita pelkällä CSS:ällä ei voida saada aikaiseksi.

Torstai 24.5.2018

Torstaina tavoitteena on koodata äänitteen lähetys-toiminto valmiiksi.

Päivän aikana valmistui äänitteen lähetys-toiminnon front-end ja osa back-endistä. Käyttöliittymän koodaus tapahtui HTML ja CSS kielillä ja toiminnallisuus JavaScriptillä. Tämän tehtävän osalta en oppinut uusia tekniikoita tai toimintamalleja, ennemminkin palautin mieleeni HTML syntaksin. Back-end valmistui firebasen osalta. Kun käyttäjä on valinnut haluamansa äänitteen ja klikannut "tallenna" painiketta, selain ottaa yhteyden firebaseen ja lähettää tiedoston, sekä tiedoston URLin tallennettavaksi. Toimintokokonaisuudesta puuttuu vielä node.js funktio, joka hakee firebasesta tapauskohtaisesti oikean yrityksen äänitteen firebasesta.

Perjantai 25.5.2018

Perjantain tavoitteena on saada äänitteen lähetys-toiminnon back-end valmiiksi, sekä muuttaa tekemäni funktiot angularJs muotoon. Sijoitin eilen tekemäni funktiot HTML sivulle script-tagien väliin, jotta pääsisin mahdollisimman nopeasti koodaamaan funktion

toiminnallisuutta. Eilen valmistuneessa funktiossani kaikki firebaseen lähetettävä data ohjautui testitietokantaan, joten tämän päivän tehtävän on siirtää funktiot AngularJS tiedostoon, jossa on tarvittava data funktiota varten.

Funktioiden siirtäminen onnistui, sillä AngularJs perusteet olivat minulla ennestään tiedossa. Tehtävän suoritettua siirryin muokkaamaan asetukset sivun ulkoasua, sillä sen skaalautumisessa mobiililaitteelle oli virheitä, elementit skaalautuivat toistensa päälle.

Päivän aikana korjasin virheet asetukset sivun ulkoasusta ja siirsin äänitteen lähetysoiminnon funktiot oikeisiin JavaScript tiedostoihin.

Viikkoanalyysi

Viikon aikana ymmärryksen VAIHDE-sovelluksen navigaatorakentesta ja renderöintifunktiosta syventyi, kun tutkin ratkaisuja kirjautumissivun alanavigaation piilottamiseksi. Ongelman ratkaiseminen renderöintivaiheessa onnistui VAIHDE-sovelluksessa, mutta en ollut varma onko sovelluksen logiikan käsittely renderfunktiossa osa hyviä React Native käytäntöjä. Fedosejevin (2015, 33) mukaan metodi on oikein, sen lisäksi että render palauttaa sivun elementit, se määrää myös miten ne palautetaan. Sovelluksen logiikka voi käsitellä propsien ja staten lisäksi renderöintifunktiossa. Oivalsin, että renderöinti ei ole vain sivujen palauttamista varten, koodari voi asettaa renderöintivaiheeseen myös ehtoja.

Viikko sisälsi moninaisia tehtäviä joiden osalta opin yksittäisiä taitoja, mutta en niinkään laajempialaisia konsepteja edellämainittua lukuunottamatta. Tulevaisuudessa osaan asettaa ja suunnitella indexointisääntöjä firebaseen, sekä ennaltaehkäistä fragmentaatio-ongelmia koodatessani.

3.4 Seurantaviikko 4

Maanantai 28.5.2018

Maanantaille ei ole tiedossa tehtäviä.

Päivän aikana korjasin sekalaisia bugeja ja virheitä GBC Vaihdepalvelusta. Korjaukset koskivat sivujen kielioppivirheitä, elementtien skaalautumisongelmia ja englanninkielisten tekstien kääntöä suomeksi. Lisäksi korjasin sivujen käyttäjänluontisivulta bugin, joka mahdollisti uuden käyttäjän luomisen minkä tahansa yrityksen nimissä. Sivuston

väärinkäyttäjä olisi ohjelmointivirheen myötä voinut tehdä uuden käyttäjän väärillä tiedoilla ja tarkastella minkä tahansa asiakasyrityksen puhelutietoja. Korjasin virheen asettamalla ehdon käyttäjän luomiseen, joka tarkastaa onko tekstikenttään syötettyä yritystä kirjoitettu tietokantaan. Tänäpä korjatut virheet ratkaisin yksinkertaisilla HTML, CSS ja JavaScript koodilla.

Tiistai 29.5.2018

Tiistaille ei ole tiedossa tehtäviä

Sain päivän alussa tehtäväksi muokata VAIHDE-sovelluksen tilatietofunktiota.

Tilatietofunktio käynnistyy, kun käyttäjä vaihtaa omaa tilastatustaan (paikalla, poissa, lounaalla, kokouksessa). Käyttäjän vaihtaessa statusensa joksinkin muuksi kuin paikallaolevaksi, hänet pitää siirtää pois soittoryhmästä, jos hän onsellaisessa. Jos käyttäjä vaihtaa itsensä takaisin lounaalle, hänet pitää lisätä takaisin soittoryhmään, jos hän oli siellä ennen lounaalle lähtöä.

Muokkasin olemassa olevaan funktioon koodia, joka suorittaisi edellä mainitun tehtävän. Huomasin, että funktiosta on tulossa sekava, sillä sama funktio suorittaa käyttäjän statusen muutoksen sekä lounas ajastimen asettamisen ja nollaamisen. Kolme toisiinsa liittyvää erillistä toimintoa samassa funktiossa ei ole hyvien toimintatapojen mukaista. Mikäli tulevaisuudessa on tarvetta muokata funktion yksittäisen toiminnon koodia (esimerkiksi lounasajan nollaaminen), funktion muiden osien logiikka täytyy ottaa huomioon tarpeettomasti.

Päivän aikana muokkasin funktion funktionaaliseen muotoon. Tehtävä haastoi ajattelemaan ongelmaa funktionaalisen ohjelmoinnin paradigmasta siten, että olemassa oleva funktio suunnitellaan ja koodataan uusiksi.

Keskiviikko 30.5.2018

Keskiviikon tavoitteena on koodata valmiiksi eilen uudelleen suunniteltu funktio.

Käyttötapaüksessa jossa käyttäjä valitsee jonkin muun tilatiedon kuin "Paikalla", käyttäjä poistetaan soittoryhmistä, jotta yrityksen puhelut eivät ohjaudu käyttäjälle.

Ongelmaksi muodostui soittoryhmien ehdollinen säätely käyttötapaükskohtaisesti. Kaikissa käyttötapaüksissa joissa tilatieto muuttuu muuksi kuin paikallaolevaksi, käyttäjät poistetaan soittoryhmistä. Saman logiikan koodaaminen käänteisenä ehtona "jos käyttäjä

on jälleen paikalla, lisää käyttäjä soittoryhmään” on virheellinen, sillä jos käyttäjä ei ollut missään soittoryhmässä ennen tilan muuttamista poissaolevaksi, hänet lisätään soittoryhmiin.

Helppo ratkaisu ongelmaan olisi ollut lisätä tietokantaan jokaisen soittoryhmän alle tieto soittoryhmään kuulumisesta, mutta tässä tapauksessa helppo ratkaisu ei ole hyvä ratkaisu. Tietokanta on tiedon varastointia varten, ei ”muistiinpanoja” varten. Koetin myös suunnitella node.js kuuntelufunktiota, mutta ratkaisusta oli tulossa liian monimutkainen. En keksinyt ongelmaan ratkaisua, joten vaihdoin tehtävää vaihtelun vuoksi toiseen.

Suurin osa toiminnoista ei toimi jos VAIHDE-sovellusta käyttävä laite ei ole yhteydessä internetiin. Sovelluksessa ei ollut metodia ilmoittaa internetyhteyden puuttumisesta käyttäjälle. Ratkaisin ongelman käyttämällä aiemmin oppimaani metodia, ehdollista renderöintiä. Asetin jokaiselle sivulle ehdon, jonka mukaan internetyhteyden puuttuessa käyttäjälle palautetaan tyhjä sivu, joka kehottaa tarkistamaan internetyhteyden.

Päivän aikana käytin paljon aikaa suunnitellessani ratkaisua soittoryhmäongelmaan, sillä halusin ratkaisun olevan hyvien toimintatapojen mukainen. Ratkaisuja suunnitellessani opin, miten node.js funktioilla palautetaan firebaseen arvoja ennen ja jälkeen kuuntelufunktion kutsumisesta. Internetyhteyden puuttumisilmoitusta koodatessani sain harjoitella aiemmin oppimaan ehdollista renderöintiä.

Torstai 31.5.2018

Torstain tavoitteena on tutkia ratkaisuja sovelluksen käynnistyessä ilmenevään hitauteen. Lisäksi aion siirtää sovelluksen yläelementin omaksi komponentiksi, jotta sama koodi ei toistu useita kertoja sovelluksessa.

Päivän aikana en päässyt tavoitteeseeni funktion suhteen. Metodi, jolla käyttäjätiedot haetaan firebaseesta on monimutkainen ja moniosainen. Sain selville hakufunktiosta sen, että yksi osasy syy hitaudelle on samojen käyttätietojen hakeminen useita kertoja. Syy tähän on minulle toistaiseksi tuntematon, mutta tieto funktion toimintavirheestä valotti osaltaan hitauden syytä. Siirsin sovelluksen yläelementin omaksi komponenttikseen. Tehtävän tekemisen aikana oivalsin että React Nativessa data kulkeutuu komponenttihierarkiassa yhteen suuntaan, toisin kuin esimerkiksi AngularJs:ässä, jossa data vaihtuu komponenttien välillä hierarkiasta riippumatta. Koodaamani yläelementti oli nyt kaikkien sivujen saatavilla yhdellä koodirivillä.

Perjantai 1.6.2018

Perjantain tavoitteena on koodata tiistaina aloittamani funktio valmiiksi. Käyttäjä valitessa jonkin muun tilatiedon kuin ”Paikalla”, hänet täytyy poistaa soittoryhmistä, jotta yrityksen puhelut eivät ohjautu käyttäjälle.

Päivän tuloksena ratkaisin soittoryhmän kytkentäongelman. Ratkaisussani käyttäjän soittoryhmän boolean arvo asetetaan komponentin stateen. Kun funktio ajetaan ensimmäisen kerran, eli käyttäjä kytkee itsensä tauolle, tarkistetaan firebasesta mikä on käyttäjän soittoryhmän sen hetkinen arvo joka asetetaan stateen. Kun käyttäjä käynnistää funktion uudelleen asettamalla itsensä paikalla olevaksi, funktio tarkistaa stateen asetetun arvon ja päivittää soittoryhmän arvon tarvittaessa. Lopuksi state asetetaan tyhjäksi, jotta seuraavalla kerralla kun käyttäjä vaihtaa tilaa, prosessi alkaa alusta. Tehtävää ratkaisu vaati abstraktia ajattelua ja kykyä hahmottaa mikä on ratkaistava ongelma kokonaisuudessaan. Vastaavia JavaScript funktioita en löytänyt internetistä, joten tehtävä haastoi käyttämään omaa tietotaitoani koodin suunnittelussa.

Viikkoanalyysi

Viikon aikana suurin osa työstä kului VAIHDE-sovelluksen parissa, ja siitä työstä suurin osa kului soittoryhmäfunktioita suunnitellessa. Keskiviikkona funktiota kirjoittaessa oivalsin uuden tavan lähestyä funktioiden koodaamista. Aiemmin lähestyin useimpia funktioita hakemalla ensiksi tarvittavan datan tietokannasta tai sovelluksesta ja sen jälkeen suunnittelemalla funktioiden logiikan. Lähestymistapa on hyvä sellaisissa tilanteissa, joissa tiedän minkä datan varmasti tarvitsen funktiota koodatessani. Muissa tilanteissa metodi haittaa ongelmanratkaisua.

Metodin rajoite on, että jos data haetaan ennen logiikan suunnittelua, funktion logiikan paradigmat muodostuu haettu data. Käytännössä tämä siirtää itse ongelmanratkaisua kauemmas tulevaisuuteen. Nämä kaksi vaihetta kannattaa siirtää toisinpäin: abstrakti ajattelu ja ongelmanratkaisu tulisi tehdä ensiksi, jotta aikaa ei kulu turhaan datan arvojen käsittelyyn. Funktion kirjoittamiseen kului aikaa useiden työpäivien ajan, koska koetin ”mahduttaa” datani logiikkaan, enkä hakea logiikkaan tarvittavaa dataa. Kun käänsin metodini toisinpäin, ratkaisu valmistui muutamissa tunneissa.

Vastaavien funktioiden suunnittelun aloitan tulevaisuudessa siten, että pohdin ensiksi mikä on pääongelma, joka pyritään ratkaisemaan funktiolla. Kun käsityksen

perimmäisestä ratkaistavasta ongelmasta on hahmoteltu, funktion logiikka voidaan suunnitella haltutusta lopputilanteesta ”takaperin” alkutilanteeseen. Tällä tavoin saadaan nopeammin selkeä käsitys siitä, millaisia datankäsittelyvaiheita funktiossa on pakko olla ja millaisia React Native komponentteja tulisi kirjoittaa. Adhithi R. kirjoittaa artikkelissaan ”Props and State in React Native explained in Simple English”, että propsien avulla React Native komponentteja voidaan käyttää uudelleen sovelluksen eri näkymissä, jolloin jokaista näkymää kohti ei tarvitse kirjoittaa uutta komponenttia. Kun funktioiden logiikka kirjoitetaan ”takaperin”, on helpompi hahmottaa mahdollisuudet uudelleen käytettäville komponenteille.

Ratkaisulähtöinen lähestyminen funktioon auttaa hahmottamaan, miten funktio voidaan koodata funktionaalisen ohjelmoinnin periaatteilla. Kun funktion datankäsittelyvaiheet on hahmoteltu, niistä voidaan rakentaa modulaarisia funktiokomponentteja, jotka ovat funktionaalisen ohjelmoinnin periaatteiden mukaisesti arvojen palauttamista varten. Dan Mantylan (2015, 12) mukaan tällaiset komponenteista koostuvat funktiot ovat selkeästi spesifioituja, helpommin debugattavissa ja helpompia ylläpitää. Koodaamastani statuksenvaihtofunktiosta tuli edellämainitun määritelmän mukainen: sen eri vaiheet ovat omissa funktiomoduuleissaan, jolloin jokaista vaihetta voidaan ylläpitää ja debugata riippumattomasti.

3.5 Seurantaviikko 5

Maantaina 4.6.2018

Maanantain tavoitteena on koodata vaihdepalveluun komento, joka tallentaa käyttäjän laitteeseen pikakuvakkeen työpöydälle, jota klikkaamalla vaihdepalvelu aukeaa selaimeen. Toinen tehtävä on muokata suosikit-sivun rakennetta muotoon, jossa yhteystietojen lataamisen aikana näkyvä latautumisanimaatio olisi helpommin toteutettavissa.

Päivän aikana muokkasin suosikit-sivun rakennetta helppolukuisammaksi, mutta en edistynyt lopullisessa tavoitteessani, joka on hitausongelmien korjaaminen. Loin vaihdepalvelua varten HTML-tiedoston, joka ohjaa käyttäjän suoraan websovellukseen, mutta luin internetistä jälkepäin ettei kyseinen ratkaisu oimi. HTML-tiedostoa lukeva sovellus ei ole kaikissa android puhelimissa selain, vaan esimerkiksi tekstieditori. Päätin siirtää tehtävän myöhemmäksi, sillä vaihdepalvelussa oli prioriteetiltaan tärkeämpiä virheitä selvitettävänä.

Tiistai 5.6

Päivän tavoitteena on korjata vikoja vaihdepalvelussa. Sivu palauttaa käyttäjälle väärää uudelleenohjaussivuja ja viestejä.

Päivän aikana korjasin useita logiikkavirheitä vaihdepalvelun etusivulta. Sivujen uudelleenohjaamislogiikassa oli useita virheitä. Sivuille oli asetettu logiikka, jonka mukaan tiettyjen toimintojen käynnistäminen ohjaa käyttäjän uusille sivuille sen mukaan, onko käyttäjä kirjautunut sisään. Esimerkki tällaisesta uudelleenohjauksesta on käyttötapaus, jossa sisäänkirjautumaton käyttäjä klikkaa itsensä uuden käyttäjän luonti sivulle. Sivu palauttaa välittömästi käyttäjälle viestin ”Varmistussähköposti lähetetty”, jonka tulisi näkyä vasta kun uusi käyttäjä on luotu onnistuneesti. Virheet johtuvat sivuille asetetuista kuuntelijafunktioista, jotka käynnistyvät käyttäjän tilan vaihtuessa. Luin firebasen dokumentaatiota käyttäjän tilaa koskevista kuuntelufunktioista, joista selvisi missä tilanteissa funktio käynnistyy. Dokumentaatioista saatua tietoa hyödyntämällä korjasin osittain sivujen virheet. En saanut kaikkia virheitä korjattua, joten jatkan tehtävien parissa seuraavana työpäivänä.

Keskiviikko 6.6.2018

Keskiviikon tavoitteena on korjata loput vaihdepalvelussa ilmenneistä virheistä, joiden korjaamisen aloitin eilen.

Korjasin virheet käyttäjän luonti sivulta. Sivusto ilmoittaa käyttäjälle virheistä, kuten esimerkiksi liian lyhyestä salasanasta tai puuttuvista tekstikentistä, eikä käyttäjä voi rekisteröityä tietokannasta löytyvän yrityksen nimellä.

Siirryin seuraavaan virheeseen, joka koski uloskirjautumista. Käyttötapauksessa jossa käyttäjä on kirjautunut sisään, etusivulla näkyy ”kirjautu” painike, joka johtaa sisään kirjautumissivulle vaikka käyttäjä on jo sisäänkirjautuneena. Sivulla näkyy edelleen tekstikentät käyttäjänimelle ja salasanalle, sekä painike sisäänkirjautumiselle jossa lukee ”kirjautu ulos”.

Uloskirjautuminen pitää koodata siten, että etusivun ”kirjautu” painike muuttuu kirjautuneelle käyttäjälle muotoon ”kirjautu ulos”, ja tätä painiketta klikkaamalla käyttäjä voi kirjautua ulos.

Päivän aikana korjasin käyttäjän luonti sivun virheet ja aloitin kirjaudu-painikkeen korjaamisen. Käyttäjän luonnin virheitä korjatessani opin suunnittelemaan validaatiologiikkaa. En saanut jälkimmäistä ongelmaa toistaiseksi ratkaistua, sillä en saanut firebasea tuotua etusivulle oikein. Sain virheilmoituksen, jonka mukaan firebasea ei ole määritelty, vaikka muilla sivuilla firebasen toiminnot ovat käytettävissä. Luin firebasen dokumentaatiota toimintojen tuomisesta HTML-sivulle, mutta ne eivät auttaneet oman ongelmani ratkaisemisessa.

Torstai 7.6.2018

Torstain tavoitteena on luoda VAIHDE-sovellukseen toiminto, joka palauttaa käyttäjälle latausanimaation, kun dataa ladataan firebasesta.

Muokkasin suosikit-sivun käyttäjän haku metodin uusiksi. Edellinen metodi palautti firebasesta yhteystietoja käyttäjän nähtäville yksi kerrallaan. Metodin ongelmana oli kontrollin puuttuminen yhteystietojen palauttamisessa. Sovelluksen koodissa ei ollut tapaa tarkistaa milloin kaikki käyttäjät on ladattu, jonka vuoksi animaation pysäytyskomento ei ollut toteutettavissa. Käyttäjien latautuessa sovelluksen käyttäminen oli hidasta, sovellus toimi normaalilla nopeudella vasta kun käyttäjiä hakeva metodi oli suorittanut loppuun.

Päivän aikana sain käyttäjiä hakevan metodin uusittua. Uusi metodi erosi vanhasta siten, että käyttäjätiedot palautetaan käyttäjälle kerralla. Ensiksi sovellus tarkistaa onko käyttäjätieto array tyhjä, jos on, käyttäjälle palautetaan latausanimaatio, samalla kun käyttäjätietoja haetaan firebasesta ja lisätään arrayhyn. Arrayn sisältäessä kaikki tarvittavat käyttäjätiedot, käyttäjätietolista palautetaan käyttäjälle ja latausanimaatio kytketään pois näkyviltä.

Perjantai 8.6.2018

Perjantain tavoitteena on hyödyntää eilen koodaamaani datanhaku metodia kaikille tarvittaville sivuille. Numeronäppäimistö sivulla numeroiden painaminen ei toimi ennen kuin kaikki käyttäjäteidot on haettu firebasesta.

Päivän aikana en päässyt tavoitteeseeni. Ongelmaksi muodostui datan lähettäminen sellaisten sivujen välillä, jotka ovat komponenttihierarkiassa sisaruksia. React Nativessa hyvien toimintatapojen mukaan data kulkeutuu aina komponenttihierarkiassa ”ylhäältä alas”. Tämä tarkoittaa sitä, että sovelluksen juuressa (sanastossa) määritetään ja haetaan

kaikki data, ja tämä data lähetetään propsien avulla juuren lapsille ja lapsenlapsille. Asetan maanantain tavoitteeksi siirtää suosikit-sivulla olevan käyttäjienhaku funktion juureen, jotta latausanimaation kytkentä olisi helpommin kontrolloitavissa kaikilla sivuilla.

Viikkoanalyysi

Viikon aikana opin kenttävalidaatioiden suunnittelua ja debuggausta, sekä syvensin ymmärrystäni React Nativen komponenttihierarkiasta.

Sain koodattua ratkaisun kauan sovellusta vaivanneeseen hitausongelmaan. Vaikka varsinainen hitaus ei ratkaisuni myötä kadonnutkaan sovelluksesta, käyttäjätietolistojen lataamisesta johtuva käyttäjäkokemusta haittaava toimintojen hitaus on ratkaistu. Sovelluksissa joissa haetaan dataa usein tietokannasta esiintyy aina jonkinlaista hitautta. Kyse onkin usein hitauden hallitsemisesta, ei poistamisesta, sillä kommunikaatio palvelimeen vaatii aina oman aikansa.

Ratkaisuni oli koodin kannalta yksinkertainen, mutta yksinkertaisenkin debuggauksen koodaaminen voi olla vaikeaa, kun olemassa oleva koodi on vaikeasti ymmärrettävää. Käyttäjiä hakeva metodi suosikit-sivulla sisälsi useita koodirivejä, joiden oleellisuutta funktion toimintoon aloin epäillä. Lopulta paljastui, etteivät kyseiset rivit suorittaneet mitään funktiossa. Freelanceri jolta koodi oli tilattu oli koodannut rivit hakufunktiota varten, mutta hän ei koskaan viimeistellyt koodia. Rivit jäivät koodiin ainoastaan haittaamaan koodin johdonmukaisuutta.

Latausanimaatiota koodatessani opin tulkitsemaan ja debuggaamaan toisen koodarin kirjoittamaa koodia sekä hallitsemaan dataa komponenttihierarkioiden välillä. Taito lukea koodia jota ei ole itse kirjoittanut on erittäin oleellinen koodarin ammatissa, sillä harvoin mitään koodia kirjoitetaan täysin yksin, ellei kyseessä ole tilaustyö freelancerilta. Kuten minkä tahansa puhutun ja kirjoitetun kielen kanssa, koodarit kirjoittavat kukin omanlaistaan koodia. Koodin tyyliin vaikuttavat uset tekijät, eritoten osaamistaso, kielitausta ja tapa ajatella tietojenkäsittelyongelmia. Toisen koodarin koodia luettaessa on tärkeää pitää mielessä kriittisyys koodia kohtaan,

Perjantain merkinnässä kirjoitin, etten saanut lähetettyä tarvittavaa dataa käyttäjätietolistojen latautumisesta komponenttien välillä. Oivalsin, että käyttäjien hakua ei kannata tehdä suosikit-sivulla, vaan sovelluksen juuressa. Sovellus ei tarvitse käyttäjätietolistaa muualla kuin suosikit-sivulla, mutta tieto toiminnon aloittamisesta ja päättämisestä tarvitaan kaikilla sivuilla. Sen sijaan, että suunnittelen monimutkaisen tavan

hakea tieto latausanimaation alkamisesta ja päättymisestä joka sivulle, tieto voidaan lähettää suoraan juuresta kaikille sivuille kerralla, jos hakufunktio on sijoitettu juureen. Tällä tavoin sovelluksen toiminta on hyvien React Native periaatteiden mukainen, ja koodi saadaan pidettyä yksinkertaisempaan.

3.6 Seurantaviikko 6

Maanantain 11.6.2018

Maanantain tavoitteena on siirtää suosikit sivulla uusittu käyttäjätietojen hakufunktio sovelluksen juureen.

En päässyt tavoitteeseeni päivän aikana. Huomasin, että sovelluksen konsoliin oli ilmestynyt varoituksia, jotka täytyi korjata ennen funktion siirtämistä juureen. Varoitukset koskivat renderöinnin aikana tapahtuvia staten muutoksia. Vaikka sovellus toimikin toivotulla tavalla, virheilmoitukset viestivät huonosta koodista joka on hyvä muuttaa ennen kuin koodia muokataan lisää. Jälkeenpäin debuggaamisesta tulee hankalaa, tai jopa mahdotonta. Luin internetistä tietoa virheestä, ja sain sen korjattua muutamalla koodirivillä. Käynnissä olevan renderöinnin aikana ei voi muokata statea, sillä staten muokkaaminen käynnistää aina uuden renderöinnin. Toisinsanoen, renderöinnin aikana ei voida käynnistää toista renderöintiä, vaan renderöinnit tulee käynnistää aina yksittäisinä komentoina. Tekemäni muutokset estivät viime viikolla tekemiäni muutosten toimivuuden, joten jouduin suunnittelemaan funktion uudestaan.

Tiistai 12.6.2018

Tiistain tavoitteena on korjata eilen ilmenneet virheet käyttäjätietojen haussa.

Päivän aikana pääsin tavoitteeseeni. Siirsin funktion sovelluksen juureen, josta tieto käyttäjätietojen latautumisesta saadaan välitettyä kaikille sovelluksen sivuille. Latausanimaatio palautuu käyttäjälle käyttäjälisterien latautuessa, ja sivujen sisältö palautetaan vasta kun kaikki toiminnot toimivat ilman viivettä. Vikatestasin sovellusta käymällä läpi erilaisia käyttötapauksia sovelluksen toiminnoissa. Virheitä ei ilmennyt, joten buildasin sovelluksen ja päivitin sen Play kauppaan.

Keskiviikko 13.6

Keskiviikon tavoitteena on päivittää VAIHDE-sovelluksen iOS versio. Eilen Play kauppaan päivittämässäni Android versiossa on useita muutoksia, jotka tulisi saada toimimaan myös iOS-alustalle.

Kopioin päivitettyt tiedostot Android kansioista iOS kansioon. Kun testasin sovellusta simulaattorissa, sain virheilmoituksen "react-native kuuntelijafunktio ei ole funktio". Olin aiemmin saanut saman virheilmoituksen, mutta en muistanut enää missä yhteydessä virheilmoitus ilmestyi tai miten korjasin virheen. Tiesin, että iOS versio toimi ennen kuin kopioin kansiot, joten poistin lisäämäni kansiot ja aloitin kansioiden kopioimisprosessin uudelleen. Ilmeni, että virhe ilmestyi vain projektin juuritiedoston kohdalla. Kopioin kaikki muut päivityskansiot projektiin. Päätin kopioida kaikki funktiot Androidin juuresta iOS juureen yksi kerrallaan, jotta virheilmoituksen syy selviäisi.

Päivän aikana siirsin suurimman osan android version uusista ominaisuuksista iOS versioon. Virheilmoituksen syyksi ilmeni kuuntelijafunktio, joka kertoo sovellukselle onko sovellus aktiivinen käyttäjän mobiililaitteessa, eli käyttääkö käyttäjä sovellusta. Toiminto ei ollut tarpeellinen, sillä tieto sovelluksen aktiivisuudesta/epäaktiivisuudesta liittyy ilmoitusten vastaanottamiseen. Ilmoitusten vastaanottaminen on toteutettu VAIHDE-sovelluksessa node.js funktiolla, joten poistin koodista virheen aiheuttavan rivin.

Torstai 14.6

Torstain tavoitteena on päivittää eilen valmistunut iOS-päivitys App storeen.

Sovelluksen päivittäminen App storeen on kolmivaiheinen prosessi. Ensin sovellus avataan xcodella ja validoidaan virheiden varalta. Jos validointi onnistui, se arkistoidaan Applen iTunes connect-palveluun, jossa sovelluskehittäjät hallitsevat julkaisemiaan sovelluksia ja niiden päivityksiä. Arkistoinnin jälkeen sovelluspäivitys ilmestyy iTunes connectiin, josta se voidaan lähettää Applen tarkistettavaksi. Kun Apple on hyväksynyt sovelluksen, se ilmestyy ladattavaksi App storeen.

Törmäsin virheeseen arkistointivaiheessa. Xcode ilmoitti virheellisistä viittauksista sovelluksen buildauskansioihin, joita ei löytynyt arkistointiprosessissa. Etsin internetistä tietoa virheestä. Selvisi, että olin avannut xcode projektin klikkaamalla väärää projektitiedostoa. Arkistointi ja sovelluksen lähettäminen arviointiin onnistui odotetulla tavalla avattuani projektin oikeasta tiedostosta.

Perjantai 15.6

Perjantain tavoitteena on korjata vikoja vaihdepalvelussa.

Päivän aikana korjasin useita toiminto- ja ulkoasuvirheitä. Muokkasin kaikki sivuilla olevat ilmoitusviestit tyyllitettyyn muotoon, joka on visuaalisesti parempi. Selaimen sisäänrakennettu ilmoitusmetodi palauttaa ilmoitukset ponnahdusviestinä muodossa "Viesti sivulta vaihde.io: viesti", joka täytyy kuitata painamalla viestin "ok" nappia, jotta viesti katoaa näkyviltä. Muokkasin ilmoitusviestit toimimaan "toastr" JavaScript kirjastolaajenuksella, joka korvaa selaimen oletusilmoitukset visuaalisesti paremmalla ja automaattisesti katoavalla ilmoituksella.

Korjasin sisäänkirjautumissivulta virheen, joka esti salasanan palautus-toiminnon toiminnan. Virhe johtui javascript validaaion puutteesta. Sivusto ei ilmoittanut tyhjästä sähköpostikentästä, jonka vuoksi "palauta salasana"-painike ei suorittanut haluttua funktiota painalluksesta. Asetin sivulle ilmoitusviestin, joka ilmoittaa sähköpostin puuttumisesta tai sähköpostiosoitteen väärästä muotoilusta.

Viikkoanalyysi

Viikon teemaksi muodostui debuggaus. Debuggaustaitojen tärkeys ilmenee työssäni joka päivä. Aina kun koodiin tehdään muutoksia tai lisäyksiä, syntyy virheitä. Useimmiten virheilmoitukset ilmoittavat suoraan virheen syyn ja sijainnin, jolloin koodi on helppo korjata. Virhe on myös helppo päätellä koodaajan kirjoittaman koodin perusteella tarkastelemalla muokattua koodin osaa. Jos sovelluksen tai internetsivun kehityksessä koodi toimi odotetulla tavalla, mutta ei enää sen jälkeen kun siihen tehtiin muutoksia, on ilmeistä, että tehdyt muutokset aiheuttivat virheen.

E erityisesti silloin kun koodiin lisätään kerralla paljon koodia, syntyy riski vaikeammin debugattavista virheistä. React Native kehityksessä tämä ilmenee esimerkiksi siten, että sovellukseen lisätään jokin toiminnallisesti yleinen toimintokokonaisuus tai komponentti (esimerkiksi navigaatiojärjestelmä, tietynlainen animaatio tai ulkoasullinen skaalausjärjestelmä) jonka toinen koodari on kirjoittanut. Komponentti itsessään voi olla toimiva, mutta konteksti johon se liitetään saattaa luoda virheitä sovellukseen, joiden debuggaus on haastavaa. Tällaisissa tilanteissa koodarilla on edessään virhe, joka ei aiheutunut koodarin itse kirjoittamasta koodista, vaan laajemmasta sovelluksen toimintalogiikasta.

Esimerkki kontekstista, jossa valmis koodikokonaisuus ei toiminut, on tällä viikolla ilmennyt "react native kuuntelufunktio ei ole funktio" virheilmoitus joka haastoi debuggaus prosessiin, jossa virheen syy täytyi selvittää virheilmoitusten "johtolankojen" perusteella. Virheilmoitus kertoi tarkasti mitkä rivit aiheuttivat ongelman, mutta tieto sovelluksen kaatavista riveistä ei ole koodarin näkökulmasta ns. "selkokieltä", joka iteseselitteisesti kertoisi virheen syyn laajemmasta perspektiivistä.

Yksi React Nativen eduista on se, että sama JavaScript koodi toimii sekä iOS että android laitteilla. Tämä ei kuitenkaan tarkoita sitä, että täsmälleen sama koodi toimii kaikissa tapauksissa. Liittämäni toimiva android alustan JavaScript kaatoi koko iOS version käynnistyessään, ja virheilmoitus ohjasi tarkastelemaan sovelluksen liitännäisen koodia.

Debuggausprosessini ongelman korjaamiseksi ei ollut ohjelmoinnin kannalta hienostunut, vaikka se johtikin haluttuun päämäärään. Tiesin virheilmoituksen perusteella virheen liittyvän liitännäisen toimintaan, sillä virheilmoituksen jokainen rivi viittasi kyseiseen liitännäiseen. Poistettuani liitännäisen sovelluksesta, kaikki toimi odotetulla tavalla. VAIHDE-sovelluksen tapauksessa liitännäinen oli sovelluksen toiminnan kannalta turha, mutta jos sitä olisikin tarvittu, virheen korjauksesta olisi tullut vaikeampaa. Virheilmoitus kirjaimellisesti tarkoitti sitä, että jokin osa liitännäisen koodista odottaa, että sille palautetaan funktio, mutta funktion sijasta sille palautettiin jotain muuta. Jos olisin joutunut korjaamaan virheen manuaalisesti muokkaamalla liitännäisen koodia, epäilen että korjaamisessa olisi mennyt useita työpäiviä liitännäisen koodin monimutkaisuuden vuoksi.

Virheen korjaaminen muistutti minua siitä, kuinka tärkeää JavaScriptin perusteiden tunteminen on. Vastaavien virheiden korjaaminen on mahdotonta, jos syvälinen ymmärrys JavaScriptin toiminnasta kielenä on puutteellista. Hyvän ja huonon koodarin erottaa debuggauksen osalta siinä, että kun ammattitaitoinen koodari selvittää virheen syytä suoraan annetun virheilmoituksen perusteella yksityiskohtaisesti, taitamattomampi koodari välttelee koodikokonaisuuksia joista virhe aiheutuu ja korjaamisen sijasta keksii tavan kiertää virheet. Aina kun koodissa kierretään virhe sen korjauksen sijasta, on suuri riski tuottaa ammattislangilla ilmaistuna "purkkakorjaus". Ilmaisuu on osuva, purukumilla voi korjata esineitä toisinaan jopa vakuuttavasti, mutta korjaus ei ole luotettava eikä ohjelmoinnin yhteydessä skaalautuva.

Omassa työssäni uuden oppimisen ja osittain tämän päiväkirjan merkintöjen kautta olen oppinut tunnistamaan jo ennalta, jos olen kirjoittamassa huonoa debuggausratkaisua. React Nativen kohdalla tämä tarkoittaa yleensä sitä, että hahmottelen mielessäni huonojen toimintatapojen mukaista koodia, joka ei myötäile React Nativen toimintatapoja.

Koodi voisi teoriassa toimia, mutta oman oppimiseni sekä sovelluksen toiminnan kannalta on aina parempi vaihtoehto käyttää enemmän aikaa oikeaoppisen korjauksen suunnitteluun.

3.7 Seurantaviikko 7

Maanantai 18.6.2018

Maanantain tavoitteena on korjata vaihdeplanelun skaalautumisongelmia 13-tuumaisille näytöille.

Päivän aikana pääsin tavoitteeseeni. Sivuille oli asetettu CSS- sääntöjä, jotka skaalasivat elementit näytölle väärin. En ehtinyt tehdä paljon töitä, sillä olin suurimman osan päivästä toimittamassa henkilökohtaisia menoja.

Tiistai 19.6.2018

Tiistain tavoitteena on muokata vaihdeplanelun raportit-sivun ulkoasuvirheitä.

Vaihdepalvelun raportit-sivulla asiakas voi tarkastella yrityksen puheluiden tietoja. Sivulla kuvataan päivän, viikon, kuukauden ja vuoden vastaamattomien puheluiden määrät, sekä työntekijät jotka ovat soittaneet takaisin soittajille xy-akselisilla pylväskaavioilla. Tehtäväni on muokata kaavioiden ulkoasullisia virheitä. Kaaviot toimivat chart.js JavaScript kirjastolaajenuksella, johon on asetettu valmiit parametrit datan syöttämistä varten. Otin korjattavaksi kaavioiden y-akselin liiallisen tarkkuuden. Chart.js kuvaa y-akselin luvut automaattisesti yhden desimaalin tarkkuudella, mutta koska vastaamattomien puhelujen määrä voi olla vain kokoluku, y-akselin intervallien tulisi olla kokolukuja.

Päivän aikana pääsin osittain tavoitteeseeni. Chart.js dokumentaatiossa oli ohje, kuinka y-akselin intervallit asetetaan. Asetin intervallit kokoluviiksi, ja ongelma ratkesi. Raportit sivu sisälsi myös muita korjausta vaativia ulkoasullisia seikkoja, joiden korjaaminen siirtyi seuraavalle päivälle.

Keskiviikko 20.6.2018

Päivän tavoitteena on korjata raportit sivun pylväskaavioiden ulkoasulliset seikat ja skaalautuminen näytön koon mukaan.

Päivän aikana korvasin kaavioiden oletusväriteeman. Värien asettaminen osoittautui yllättävän vaikeaksi. Useimmiten sivulla näkyvien elementtien ulkoasua muokataan CSS tyylisäännöillä, mutta chart.js tyylit asetettiin suoraan JavaScriptistä. Chart.js dokumentaation ohjeistus tyylien muutoksiin oli suppea ja osa sivuilla ilmoitetuista tärkeistä avainsanoista oli muuttunut päivitysten myötä. Tyylien muokkaamisen logiikka ei ollut niin yksinkertainen kuin muissa vastaavissa JavaScript kirjastoissa. Yleensä vastaavissa JavaScript laajennuksissa tyylit asetetaan esimerkiksi objektiin, ja tähän objektiin viitataan tyylinä. Chart.js kaavioiden oletustyyli piti ”perua” jokaisen kaaviopylvään kohdalla ja asettaa haluttu väri jokaisen pylvään kohdalla erikseen. Koska jokaisen tolpan värin asettaminen manuaalisesti olisi johtanut useisiin kymmeneen koodiriveihin jotka tekevät saman asian, käytin värien asettamisessa ”for loop”:pia (sanastossa), eli iteraatiota.

Torstai 21.6.2018

Torstain tavoitteena on koodata vaihdeplavelun pylväskaavioiden logiikka ja ulkoasu valmiiksi.

Päivän aikana pääsin tavoitteeseeni. Kaavioihin piti asettaa logiikka, jonka mukaan yksikään kaaviopylväs ei koskaan saavuta kaavion y-akselin pituutta, vaan yläosaan jää ulkoasullisista syistä muutaman intervallin verran tilaa. Suurin osa päivästä meni chart.js ja sen AngularJs-version dokumentaatiota lukiessa, sillä kaavioiden muokkaamiseksi oli asetettu tarkat JavaScript säännöt. Tekemäni muutokset koodiin olivat suhteellisen pieniä, mutta suppean dokumentoinnin ja esimerkkien vuoksi ratkaisujen löytämiseen meni paljon aikaa.

Perjantai 22.6.2018

Juhannus, ei töitä.

Viikkoanalyysi

Jatkaen viime viikon pohdinnasta, tämän viikon teemaksi muodostui debuggaaminen. Viimeviikolla debuggaaminen kohdistui virheilmoituksen ilmoittaman virheen korjaamiseen. Tällä viikolla debugattava ongelma oli virhe, joka ei tuottanut virheilmoitusta. Debuggaamisen voisi tämän myötä jakaa kahteen osaan: virheilmoitukselliset ja virheilmoituksettomat toimintavirheet. Viimeviikolla pohdin

virheilmoituksellisten virheiden korjaamisen haasteita, korjaustekniikoita ja seikkoja joiden saralla minun tulisi kehittyä ollakseni parempi debuggaaja.

Tällä viikolla tein pääasiallisesti töitä chart.js JavaScript kirjaston parissa, joka tuotti useita virheilmoituksettomia virheitä joiden korjaamiseen kului paljon aikaa.

Virheilmoituksettomien virheiden korjaamisessa on virheilmoituksellisiin virheisiin verrattaessa omat helpot ja vaikeat puolet debuggaamisen kannalta.

Virheilmoituksettomissa virheissä asianyhteys virheelle kertoo koodarille muutamia debuggausta helpottavia seikkoja koodista, joka aiheuttaa virheen. Syynä virheelle ei voi olla syntaksivirhe, sillä se tuottaa aina virheilmoituksen. Myöskään mikään funktioille asetettu argumentti ei ole määrittelemätön, sillä se tuottaisi myöskin virheilmoituksen. Mikään muuttuja ei ole määrittelemätön, mutta jokin muuttuja voi olla ulkona jonkin funktion scopesa. Virheilmoituksen virhe antaa siis koodarille jonkin verran osviittaa asioista, jotka eivät voi olla syynä virheelle, jolloin syyn etsinnän laajuus supistuu.

Haasteena virheilmoituksettomassa virheessä verrattaessa virheilmoitukselliseen virheeseen on se, että pelkän kirjoitetun koodin ymmärtäminen ei riitä. Tästä syystä en voi tämän pohdinnan yhteydessä paneutua syihin mistä virheilmoituksen virhe syntyy, sillä se on läheisesti sidottu siihen asianyhteyteen ja koodikieleen jossa se syntyy. Koodissa käytetyn koodikielen toiminta täytyy ymmärtää syvällisesti, sillä virhe syntyy ns. ”konepellin alla”, eikä kirjoitetussa koodissa, toisin kuin suurimmassa osassa virheilmoituksellisista virheistä. JavaScriptin kohdalla yksi tärkeimmistä peruskonsepteista, jota tulisi pohtia tällaisessa tilanteessa on JavaScriptin objektorientoituneisuus. Lähes kaikki JavaScriptissä on jonkinlainen objekti, koodin toiminta perustuu objektien interaktioon toisten objektien kanssa. Tämä voi viitata koodaria oikeaan suuntaan virheiden korjaamisessa, sillä jos melkein kaikki JavaScriptissä on objekti, virheen syy on todennäköisesti jollain tavalla objekti, joka ei interaktioi jonkin objektin kanssa oikein. Virheen instanssista koodari voi päätellä mikä tällainen objekti voisi olla.

Virhe toiminnallisuudessa aiheuttaa useimmiten virheilmoituksellisen virheen, sillä suurin osa koodivirheistä kuuluu muutamaan virhetyypin kategoriaan, jotka tuottavat selkeän virheilmoituksen joka osoittaa koodirivit, jotka tuottavat virheen. HTML ja CSS tuottaa usein virheilmoituksettomia virheitä kielten luonteen vuoksi: ne kuvaavat sivujen semantiikkaa ja ulkonäköä, eivät toiminnallisuutta. Koodari voi kirjoittaa mielestään halutun tuloksen tuottavaa HTML koodia, mutta tarkasteltaessa lopputulosta koodin virheellisyys käy ilmi. Selain tulkitsee koodin aina siten kuin se on kirjoitettu, eikä siten miten koodari ajatteli sen toimivan. Virheellinen HTML tai CSS koodi tulkitaan selaimessa virheettömästi, mutta haluttua lopputulosta ei väärin kirjoitetun koodin vuoksi saavuteta.

Vastaavasti kuten JavaScriptissä, tai muissa toiminnallisissa koodikielissä, kielen toimitaperiaatteet olisi hyvä tuntea. Jos tuntemus on heikkoa, koodari törmää samoihin perusvirheisiin työssään useita kertoja.

Kuten viime viikon pohdinnassa painotin, koodikielien ja koodikehysten peruskonsepteja ja toimintaperiaatteita ei voi koskaan tuntea liian hyvin. Omassa työssäni suurimpia turhautuneisuuden tuntemuksia aiheuttavat tilanteet, joissa huomaan debuggaavani tunteja sellaisia virhetyyppejä, joiden ratkaisu olisi paljon nopeampaa ja yksinkertaisempaa jos tuntuisin kielet paremmin. Kaikista suurinta arvoa omalle oppimiselleni ja kehittymiselleni tuottaakin tässä pohdinnassa mainittujen seikkojen perinpohjainen opiskelu. Koodaamisen ja debuggaamisen prosessista tulee sitä helpompaa ja tuotteliaampaa, mitä varmemmin peruskonseptit ovat hallussa.

3.8 Seurantaviikko 8

Maanantai 2.7.2018

Maanantaille ei ole tiedossa tehtäviä.

Huomasin virheen vaihdepalvelun raportointisivulla. Pylväskaavio, joka kuvaa vastaamattomien puheluiden määrää viikon ajalta, ei renderöitynyt sivulle. Pylväskaavion koodiin oli asetettu sääntö, jonka mukaan sovellus hakee firebasesta viikon puheluiden datan puhelutietojen päivämäärien mukaan. Filtröintisäännön piti palauttaa puhelutiedot alkaen kyseisen viikon maanantain päivämäärästä päättyen saman viikon sunnuntaihin. Epäilin virhettä koodin logiikassa, sillä edellämämainitun päivämäärsäännön olisi pitänyt toimia, mutta haluttu pylväskaavio ei renderöitynyt sivulle.

Päivän aikana selvitin syytä virheelle tuloksetta. Selasin suurimman osan päivästä firebasen dokumentaatiota ja internetfoorumeita liittyen firebaseen, mutta ne eivät auttaneet ongelman ratkaisussa. Koodiin asetettu metodi datan hakemiselle vaikutti oikein koodatulta, mutta se ei kuitenkaan toiminut halutulla tavalla. Metodi, jolla data filtröidään on yksinkertainen, mutta vaihdepalvelun yhteydessä se ei toiminut.

Tiistai 3.7.2018

Tiistain tavoitteena on korjata eilen havaittu vika vaihdepalvelun pylväskaavioissa.

Huomasin, että parametrit jotka filtteröivät firebasesta haettavan datan, olivat vääränlaisia objekteja. Pylväskaavio, joka kuvasi viikon aikana vastaamattomiksi jääneitä puheluita, haki firebasesta puhelutietoja, joiden päivämääräparametri oli kirjoitettu muodossa "PP-KK-VVVV". Dataa hakeva filtteri odotti, että firebaseen kirjoitetut puhelutiedot olisivat numeromuotoisia. Koska firebasesta haettu data palautettiin tekstimuotoisena eli stringinä, filtteri ohitettiin, jolloin tulokseksi muodostui kaikkien kuukausien data.

Päivän aikana aloitin raportit-sivun koodin uudelleen suunnittelun, sillä nykyinen metodi paljastui toimimattomaksi. Uudelleen suunnitteluun kuuluu puhelutietojen muokkaaminen sellaiseen muotoon, jossa firebaseen filtterointimetodit ovat käytettävissä. Tämä tarkoittaa käytännössä sitä, että puhelutietojen alle kirjoitetaan unix aikaleimatieto, jonka perusteella puheluiden filtteriöiminen ajan mukaan on käytettävissä firebaseen metodeilla.

Keskiviikko 4.7.2018

Keskiviikon tavoitteena on saada valmiiksi eilen aloitettu pylväskaavioiden koodin uusiminen vaihdepalvelun raportit-sivulta.

Päivän aikana sain funktion osittain toimimaan. Korvasin alkuperäisen funktion logiikan while-looppeihin pohjautuvalla logiikalla. Funktio hakee ensiksi tiedon niistä päivistä joiden aikana puheluja on jäänyt vastaamatta. Kun data on asetettu arrayhyn, while-looppi vertaa viikonpäiviä arrayhyn ja tarkistaa onko puheluita jäänyt vastaamatta. Kun while-loopin tarkistus osuu päivälle, jonka kohdalla ainakin yksi puhelu on jäänyt vastaamatta, toinen while-looppi tarkistaa montako puhelua on jäänyt vastaamatta sinä päivänä. Lopuksi Funktio palauttaa sivulle arrayn, joka sisältää tarvittavan datan pylväskaaviota varten.

Toistaiseksi tuntemattomasta syystä tapauksissa, joissa kuukauden alussa on päivä, jonka aikana useita puheluita jäi vastaamatta, eli jolloin pylväskaavion korkeus on enemmän kuin yksi, loput pylväskaavion palkeista eivät näy kaaviossa. Ongelman ratkaiseminen siirtyi seuraavalle päivälle.

Torstai 5.7.2018

Torstain tavoitteena on korjata eilen ilmennyt ongelma pylväskaavioissa, ja jatkaa pylväskaavioiden uuden logiikan koodaamista.

Päivän loppupuolella ongelma selvisi. Virheen syy oli koodaamani ohjelmointivirhe ja osittain sekava koodi. Sellaisissa tapauksissa, joissa firebaseen oli kirjoitettu useampi

puhelutieto samalta päivältä, koodaamani while-looppin toistava ehto valitsi puhelutieto arraystä virheellisen indexin, jolloin loput puhelutiedoista ohitettiin. Muutin koodin logiikkaa, joka laskee yhteen vastaamattomien puheluiden määrän niiltä päiviltä, joina puheluita on jäänyt vastaamatta useampia. Yksinkertaistettu logiikka johti samaan virheeseen, mutta tässä yhteydessä virheen syy oli helpompi debugata ja korjata.

Perjantai 6.7.2018

Eilen uusittu logiikka vaihdepalvelun raportit-sivun pylväskaavioihin valmistui. Päivän tavoitteena on kopioida sama uusittu toimintalogiikka sivun kolmeen muuhun pylväskaavioon.

Päivän aikana sain kopioitua logiikan pylväskaavioon, joka kuvasi vastaamattomien puhelujen määrää kuluneen vuoden ajalta. Kohtasin ongelmia työpäivän aikana työtietokoneeni kanssa, joka aloitti useita tunteja kestäneen päivityksen ilman ennakkovaroituksia. Suurin osa päivästä meni laatiessa testejä funktioille. Käytännössä tämä tarkoitti esimerkkipuhelutietojen kirjoittamista firebaseen eri päiville, viikoille, kuukausille ja vuosille jotta voin testata erilaisia mahdollisesti poikkeavia tilanteita, kuten esimerkiksi karkausvuotta, helmikuuta, joka on muita kuukausia lyhyempi ja erilaisia tilanteita joissa kuukauden tai vuoden ensimmäinen päivä ei ole maanantai.

Viikkoanalyysi

Viikon aikana suurin osa konkreettisesta työstä jota tein, oli minulle entuudestaan tuttua JavaScript koodaamista. Tehtävien kautta ei tällä viikolla ilmennyt uusia tekniikoita tai konsepteja, mutta sain vahvistaa osaamistani looppien kirjoittamisessa ja testaamisessa. Testaamisen osalta viikon tehtävät vaativat ennakkointia, sillä kirjoitettu koodikokonaisuus ei ollut sellainen, jonka oikeanlaisen toiminnan voisi todeta kokonaisuudessaan muutamalla testillä. Koska kyseessä oli aikaan pohjautuva IT-ratkaisu, (yrityksen vastaamattomat puhelut kuvattuna pylväskaavioina päivien, viikkojen, kuukausien ja vuosien aikana) koodi vaati testausta useissa erilaisissa käyttötapauksissa.

Testaamisprosessi paljasti virheellisen koodin keskiviikkona. Käyttämäni for-looppi, jolla yritin tarkastaa montako puhelua on jäänyt vastaamattomaksi esimerkiksi tiettyinä kuukautena ei toiminut, sillä logiikassa ei ollut järkevää tapaa ottaa huomioon karkausvuotta. Kun korvasin for-loopin while-looppiin, eli jokin tehtävä tehdään niin kauan kuin jokin ehto on totta, ongelma ratkesi. Testaustyyppi tässä tapauksessa oli niin sanottu "unit test" eli yksikkötesti. Yksikkötestissä sovelluksen toimintaa testataan erittelemällä

yksittäinen sovellukse osa, eli tässä tapauksessa funktio, omaksi testikokonaisuudeksi. Toimimattoman for-loopin tapauksessa testausmetodi oli manuaalinen eli varmistin funktion eri suoritusvaiheissa oikean tyyppisen ja oikeanlaisen datan. Yksikkötestauksessani ilmeni vääränlainen data sekä vääränlainen datatyyppi eli numero, kun haluttu datatyyppi oli teksti, jolloin debuggauksessa katsoin parhaaksi muuttaa for-looppi while-loopiksi.

Vastaavanlaisten ongelmien toteaminen ja testaaminen raportit-sivun kehitysvaiheessa on tärkeää, koska jos asiakkaiden käyttöön tulee tuote joka toimii aluksi muttei enää kuukausien tai vuosien päästä, korjaaminen voi käydä erittäin työlääksi ja aikaa vieväksi. Kehittämisprosessissa kehittäjä testaa koodia toistuvasti, kun tarkastellaan miten sovelluksen toiminta on muuttunut viimeisimmä koodimuutoksen vuoksi. Tämä on välttämätön osa koodausprosessia, mutta sitä itsessään ei voida kutsua sovelluksen testaamiseksi sanan varsinaisessa merkityksessä. Testausprosessi alkaa vasta kun sovelluskomponentti tai muu kokonaisuus on valmis. Kehittäjän virheajatus olisi ajatella että koska tiettyä koodikokonaisuutta testataan pintapuolisesti kehittämisen yhteydessä, se toimii varmasti kaikissa tilanteissa. Mitä perinpohjaisempi testausprosessi on, sitä suuremmalla todennäköisyydellä ongelmat vältetään sekä lyhyellä, että pitkällä aikavälillä.

3.9 Seurantaviikko 9

Maanantai 9.7.2018

Maanantain tavoitteena on koodata vaihdepalvelun raportit-sivun viikon puheluja kuvaavan pylväskaavion logiikka samanlaiseksi, kuin se on sivun muissa pylväskaavioissa.

Päivän aikana sain muutokset valmiiksi. Tehtävä oli haasteellinen, sillä puheluiden määrää tarkistava while-looppi ei voinut perustua numeroihin kuten kuukauden ja vuoden puheluita tarkistava looppi. Puheluiden määrä ilmoitetaan pylväskaaviossa tekstimuotoisena (Ma, Ti, Ke, To, Pe, La, Su). En voinut kopioida aikaisempien kaavioiden logiikkaa täysin, sillä pylväskaavion datatyyppi oli teksti. Aikaisemmin koodaamani pylväskaaviofunktiot noudattivat suurilta osilta samaa logiikkaa mitä aion käyttää, joten käytin sitä uuden funktion logiikan pohjana.

Ratkaisussani tekstimuotoiset viikonpäivät asetetaan arrayhyn, jota verrataan jokaisen vastaamattoman puhelun kohdalla puheluiden määrään jokaisen viikonpäivän kohdalla.

Tiistai 10.7.2018

Tiistain tavoitteena on jatkaa maanantaina aloitettuja tehtäviä.

Aamulla sain uuden työtehtävän. Käyttäjätietojen muokkaaminen on tähän asti tapahtunut suoraan firebasen tietokantaa muokkaamalla. Tämä on kuitenkin hidas ja vaivalloinen tapa muuttaa tietoja, joten sain tehtäväkseni luoda käyttöliittymän käyttäjien hallitsemista varten. Tämä tarkoittaa käytännössä AngularJs pohjaista ratkaisua, joka listaa kaikki käyttäjätiedot ja mahdollistaa niiden muokkaamista suoraan selaimesta. Pohjaksi käyttöliittymälle otin olemassaolevasta projektista etusivun, jossa oli valmista koodia käyttäjätietojen hakemiseen firebasesta.

Päivän aikana sain valmiiksi pohjan käyttöliittymälle. Käyttöliittymä listaa käyttäjät sivulle, mutta käyttäjien tietoja ei voi muokata. Muokkaamisominaisuuksien koodaaminen siirtyi seuraavalle päivälle.

Keskiviikko 11.7.2018

Keskiviikon tavoitteena on koodata käyttäjätiedon muokkaus-ominaisuus firebasen käyttöliittymään.

Sain eilen valmiiksi näkymän, jossa kaikki käyttäjätiedot näkyvät listana. Ongelmaksi muodostui päivän alussa käyttäjätietojen muokkaus ponnahdusikkunan koodaaminen AngularJs kirjastolaajenuksella. AngularJs ei ole vahvuuteni, joten jouduin kertaamaan paljon AngularJs:sän peruskonsepteja, jotta ymmärtäisin internetistä löytyviä koodiesimerkkejä joiden pohjalta voin kirjoittaa ponnahdusikkunan.

Päivän aikana sain valmiiksi toiminnon, joka avaa käyttäjätietoa klikkaamalla ponnahdusikkunan. Onnistuin myös logittamaan jokaisen käyttäjätiedon kohdalla kyseisen käyttäjän datan, mutta en saanut lähetettyä dataa ponnahdusikkunaan. Data oli saatavilla sivun käyttäjälista-elementissä, mutta ei ponnahdusikkunassa. Datan lähettäminen ponnahduselementtiin siirtyi seuraavalle päivälle.

Torstai 12.7.2018

Torstain tavoitteena on saada valmiiksi firebasen käyttöliittymä toimintoiheen.

Ratkaisin eilen ilmenneen ongelman, jossa en saanut käyttäjätietodataa siirrettyä käyttäjän nähtäville ponnahdusikkunaan. Ongelma johtui ponnahdusikkunan scopesta. HTML sivun HTML tagiin oli asetettu AngularJs kontrolleri, eli kaikki sivun elementit käyttävät samaa kontrolleria. Käyttämäni AngularJs palvelun (AngularJs service) vaatima palvelukohtainen kontrolleri oli koodattu sivun pääkontrollerin sisälle virheellisesti. Arvot joita koetin saada näkymään ponnahdusikkunassa olivat määrittelemättömät siellä missä koetin ilmoittaa ne, koska kontrollerit olivat sisäkkäin. Asetin tarvitsemani datan ponnahdusikkunan kontrollerin sisälle, jolloin arvot olivat saatavilla ponnahdusikkunassa.

Päivän aikana korjasin edellämainitun virheen, sekä koodasin ponnahdusikkunan ulkoasun, mutta en päässyt tavoitteeseeni valmiista projektista. Hyödynsin AngularJs:ssän ng-repeat (angular-toisto) direktiiviä käyttäjätietojen listaamiseen, jolloin kaikki tiedot haetaan automaattisesti tietokannasta. Sivun toiminnallisuuksista jäi puuttumaan muokattujen tietojen tallentaminen kantaan. Tallennustoiminnon koodaaminen siirtyi seuraavalle päivälle.

Perjantai 13.7.2018

Perjantain tavoitteena on saada firebasen käyttöliittymän tallennustoiminto valmiiksi.

Päivän aikana en päässyt tavoitteeseeni. Eilisen työn tuloksena sain käyttäjätiedot näkymään halutulla tavalla ponnahdusikkunan tekstikenttiin. HTML tekstinsyöttökenttiin on mahdollista asettaa ominaisuus "value" (arvo), jonka mukaan tekstikenttään ilmestyy haluttu teksti tai muuttujan arvo. Tekstikentän arvon voi lukea yksinkertaisella JavaScript koodilla, kun elementin Id on tiedossa. Ongelmaksi muodostui tekstikenttien arvojen lukeminen koodissa, sillä yhdelläkään tekstinsyöttökentällä ei ollut uniikkia id:että.

Firebasen käyttäjätiedoilla voi olla vaihtelevia arvoja, eli käyttäjätiedossa voi olla arvo, joka puuttuu toiselta käyttäjätiedolta. Tämän vuoksi en voinut kirjoittaa HTML koodiin yhteisiä arvoja kaikille käyttäjätiedoille, vaan hain tiedot käyttäjäkohtaisesti AngularJs:sän ng-repeat direktiivillä. HTML koodissa on täten vain yksi tekstinsyöttöelementti, joka toistetaan eri arvoilla käyttäjätietojen määrän ja sisällön mukaan. Koska kaikki tekstinsyöttökentät haetaan käyttäjätietokohtaisesti, tekstinsyöttökentillä ei ole uniikkia id:tä. Vaikken päässyt tavoitteeseeni, opin lisää AngularJs:sän perusteista dokumentaation ja koodin debukkauksen kautta.

Viikkoanalyysi

Viikon aikana suurin osa työstä liittyi AngularJs koodaamiseen. AngularJs:sää ja React Nativea ei voi verrata teknisesti suoraan toisiinsa, sillä React Native on yksinomaisesti mobiilisovellusten kehittämiseen, kun taas AngularJs on ohjelmistokehitys websovellusten kehitykseen, eikä sillä voida toteuttaa mobiilisovellusta. Mobiilisovellus on muista sovelluksista riippumaton sovellus, kun taas websovellus toimii selainsovelluspohjaisesti. Molemmilla teknologioilla kehitetään kuitenkin sovelluksia, joten kehittäjän kokemusta sovelluskehityksestä voidaan vertailla teknologioiden välillä. Viikon oppimisen teemaksi nousivat AngularJs:sän perusteet ja haasteet. Viikon aikana koodasin annettua tehtävää silmälläpitäen sitä, millainen oppimiskokemukseni AngularJs:sästä on verrattuna React Nativen oppimiskokemukseen.

Aiempaa kokemusta AngularJs kehityksestä minulla oli ennen viikon tehtäviä niukasti, joten suurin osa viikon työajasta kului tietoa etsiessä ja oppaita selatessa. Koin AngularJs:sän dokumentaatiot monimutkaisiksi ja osittain epäselväksi. Ohjeistukset eivät ole mielestäni kirjoitettu oppijan kannalta helposti lähestyttävästä perspektiivistä, toisin kuin React Nativen dokumentaatiot, joista olen saanut paljon arvokasta ja käytännöllistä tietoa. AngularJs:sän verkkosivujen ohjeet olettavat kehittäjältä syvempää JavaScriptin tuntemusta, kuin vastaavat React Native ohjeet. Haastavan dokumentaation vuoksi suurin osa oppimisestani viikon aikana tapahtui yrityksen ja erehdyksen kautta, joka teki koodaamisprosessista turhauttavaa ja hidasta.

React Nativeen verrattuna AngularJs:sän perustoimintojen koodaaminen vaatii useiden komponenttien koodaamista, joiden oleellisuus on aluksi vaikea hahmottaa. Ykinkertainen AngularJs toiminto vaatii AngularJs lähdekoodin lataamisen ja tuomisen sovellukseen, AngularJS sovelluksen julistamisen, moduulin ja kontrollerin koodaamisen sekä julistamisen, oikean Angular direktiivin koodaamisen ja lopulta halutun toiminnon koodaamisen.

React Nativessa oleellisia vaiheita toiminnallisuuden ja käyttöliittymän kehittämiseen on vain muutama, renderöitävien elementtien ja niiden ominaisuuksien (propsien) koodaaminen, sekä renderöintifunktion koodaaminen. React Nativen perustoiminnan voisi ilmaista sanallisesti: "Näytä (renderöintifunktio) nämä elementit (DOM-elementit), joilla on nämä ominaisuudet (propsit) näillä paikoilla (elementtien CSS-koodi ja järjestys jossa elemntit julistetaan renderöintifunktiossa)". AngularJs:sän MVC-toimintamallia ei voi kuvata sanallisesti yhtä tyhjentävästi siihen liittyvien peruskonseptien runsauden vuoksi. Tämän vuoksi koin vaikeaksi opetella AngularJs:sän hyviä toimintatapoja ja perusteita valmiiksi määritellyn tehtävän kautta, sillä minulla ei ollut selkeää käsitystä hyvien ja huonojen toimintatapojen eroista AngularJs:sän kontekstissa.

Ponnahdusikkunan koodaamiseen meni useita työtunteja, vaikka itse toiminto on yksinkertainen. Jälkikäteen ajateltuna uskon, että olisin suoriutunut viikon tehtävistä paremmin, jos olisin sivuuttanut varsinaisen tehtävän ja keskittynyt internetissä saatavilla oleviin AngularJs ohjeisiin. Yrityksen ja erehdyksen kautta opin mainittujen peruskäsitteiden perusteet, mutta koen että oppisin sujuvampaan AngularJs:sän käyttöön nettikurssin tai kirjan kautta, kuin valmiiksi määritellyn tehtävän kautta.

3.10 Seurantaviikko 10

Maanantai 16.7.2018

Maananantain tavoitteena on saada firebasen käyttöliittymä valmiiksi

Päivän aikana en päässyt tavoitteeseeni. Ongelmaksi tehtävässä on muodostunut vähäinen ymmärrys ja taustatieto AngularJs direktiivien yhteistoiminnoista ja rajoituksista. Tehtävän ratkaiseminen on jäänyt jumiin tilanteeseen, jossa ponnahdusikkunassa on tarvittava data, mutta metodi, jolla lukea tekstikenttien muuttuneet arvot puuttuvat. Koetin useita erilaisia lähestymistapoja ongelman ratkaisemiseksi, jotka kaikki johtivat sekavaan ja lopulta toimimattomaan koodiin.

Yksi syistä, minkä vuoksi ratkaisun löytäminen on haastavaa, on ongelman yksityiskohtaisuus. Internetistä etsimäni ratkaisut jotka liittyvät käyttämiini AngularJs direktiiveihin eivät noteeraa varsinaista ongelmaani, joka on loopattujen tekstikenttien datan hakeminen funktiossa silloin kuin tekstikenttien arvot ovat muuttuneet. Tehtävän ratkaiseminen siirtyi seuraavalle päivälle.

Tiistai 17.7.2018

Tiistain tavoitteena on ratkaista muuttuneiden arvojen onglema käyttöliittymän ponnahdusikkunasta.

Päivän aikana ratkaisin ongelman. Alkuperäisessä ideassani tekstikenttien alla olisi yksi painike, jota klikkaamalla käyttäjä tallentaa kaikkien tekstikenttien muutokset firebaseen. Päätin muuttaa ponnahdusikkunan ulkoasuun jokaiselle tekstikentälle omat tallennus- ja poistopainikkeet, jolloin JavaScriptin "this" avainsana sisälsi kyseisen tekstikentän arvon.

Katsoin parhaaksi ratkaista ongelman JavaScriptillä ilman AngularJs:sän hyödyntämistä. Uusi ideani, joka kiersi tarpeen AngularJs:sän käytölle on ulkoasullisesti lähes yhtä hyvä, eikä se hidasta käyttöliittymän käyttöä merkittävästi. Opin epäonnistuneista yrityksistäni AngularJs:sän ng-repeat direktiivistä, sekä AngularJs \$scope ja \$index objekteista. AngularJs ratkaisu olisi voinut perustua jollain tavalla mainittuihin objekteihin, mutta asiansyhteys, jossa koetin hyödyntää niitä ylitti taitoni tehtävän osalta.

Keskiviikko 18.7.2018

Keskiviikon tavoitteena on jatkaa firebasen käyttöliittymän kehitystä mahdollisimman pitkälle.

Päivän aikana sain paljon aikaiseksi käyttöliittymän koodissa. Suurin osa ulkoasullisista seikoista valmistui, ja ponnahdusikkunan perustoiminnot toimivat halutulla tavalla. Käyttöliittymä vaatii jatkokehitystä, sillä kaikki käyttäjätiedot eivät näy ponnahdusikkunassa. Käyttöliittymään täytyy liittää oikea tietokanta, sillä kehitysvaiheessa käyttäjätietojen data haetaan testitietokannasta. Tietokannan muuttaminen on helppo tehtävä, mutta en ehtinyt tehdä sitä tämän päivän aikana. Ponnahdusikkuna listaa käyttäjätiedon nodet ng-repeat direktiivillä, jolloin käyttäjälle palautetaan klikatun käyttäjätiedon nodet. Ponnahdusikkuna ei hae nodejen lapsinodeja, joten ponnahdusikkunaan täytyy koodata sisäkkäin useampi ng-repeat direktiivi.

Täyden käyttäjätietorakenteen palauttamisen lisäksi käyttöliittymän koodiin täytyy lisätä tuki näppäimistökomennoille. Tietokantaan kirjoittaminen on nopeampaa ja helpompaa, kun käyttäjä voi tallentaa tietoja enter-näppäimellä ja navigoida rivien välillä tab-näppäimellä.

Torstai 19.7.2018

Torstain tavoitteena on jatkaa firebasen käyttöliittymän jatkokehitystä.

Sain töihin tullessani pyynnön koodata loppuun vaihdeplavelun raportit-sivun pylväskaaviot, sillä tuote tulee asiakkaan käyttöön seuraavana päivänä. Pylväskaavioiden osalta työtä ei ollut paljon. Testasin osan aiemmin tekemästäni koodista ja varmistin että olemassaolevat funktiot toimivat halutulla tavalla. Päivitin raportit-sivun asiakkaiden nähtäville "firebase deploy" komennolla ja siirryin työstämään firebasen käyttöliittymää.

Päivän aikana testasin ja kirjoitin puhtaaksi kirjoittamani koodin vaihdepalvelun raporttisivulla. Siistin koodista turhat välit ja debuggausrivit sekä kirjoitin koodiin kommentteja jotka selittävät koodin toimintaa, sillä metodi joka tarkistaa puheluiden määrän tiettyinä ajankohtina vaati useita looppeja, joiden toiminnan hahmottaminen voi olla haastavaa. Kommentit auttavat muita kehittäjiä ymmärtämään koodia paremmin, jotta aikaa ei kulu turhaan koodin tulkitsemiseen.

Perjantai 20.7.2018

Perjantain tavoitteena on koodata käyttöliittymän ponnahdusikkunaan ng-repeat direktiivi, joka palauttaa firebasesta käyttäjätietojen syvemmät nodet käyttäjälle.

Päivän aikana pääsin osittain tavoitteeseeni. Asetin sisäkkäin kaksi ng-repeat direktiiviä, joista toinen hakee ensin käyttäjän tiedot ja toinen listaa käyttäjätietojen alanodet, jos niitä on. Näppäimistö navigaation vaatimat funktiot valmistuivat työpäivän lopussa. Työpäivä jäi melko lyhyeksi, sillä olin suurimman osan päivästä hoitamassa henkilökohtaisia asioitani.

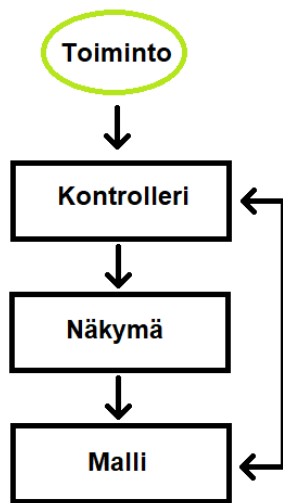
Viikkoanalyysi

Viikon aikana suurin osa työstä kului viimeviikon tavoin AngularJs:sän opetteluun ja koodaamiseen parissa. Viimeviikolla keottu tuskastuttava yrityksen ja erehdyksen kautta tapahtunut oppiminen opetti, miten AngularJs ei toimi. Tällä viikolla se miten AngularJs toimii, oli helpompi hahmottaa, sillä en yrittänyt ratkaista koodausongelmia toimimattomiksi todetuilla metodeilla. Viime viikon pohdinnassa kirjoitin myös siitä, miten eri AngularJs:sän vaatimien komponenttien oleellisuus oli vaikea ymmärtää. Tällä viikolla eri komponenttien rooli koodissa selkeytyi, ja aiemmin koettu monimutkaisuus alkoi selkeytyä.

Eniten viikon tehtävien koodausta auttoi syventynyt ymmärrys AngularJs:sän MVC-arkkitehtuurista (malli–näkyvä–kontrolleri). Advanced Kittenryssä julkaistussa opetusdokumentaatioissa ”Arkkitehtuuri ja MVC” kuvataan, miten sovelluksen näkymät hoitavat kaiken HTML:ään liittyvän koodin ja kontrollerit ottavat vastaan käyttäjän antamat pyynnöt. Vastaavia kontrollereita ei ole React Nativessa, joten viikon aikana minun tuli muuttaa tapaan ajatella websovelluksen koodaamista.

MVC malli on lyhyesti selostettuna tapa suunnitella sovelluksen arkkitehtuuri siten, että sovelluksen logiikka erotetaan käyttöliittymästä. Malli on vastuussa sovelluksen datan

käsittelystä, näkymä on käyttäjälle palautettava käyttöliittymä ja kontrolleri ottaa vastaan käyttäjän toiminnot (esimerkiksi painikkeiden klikkaukset).



Kuva 10. MVC-arkkitehtuuri havainnollistettuna.

Kuva 10 esittää MVC-arkkitehtuurisen sovelluksen toimintaa, eli tässä tapauksessa koodaamaani firebase käyttöliittymää. Käyttäjän toiminto, esimerkiksi painikkeen klikkaus, käynnistää tietyn kehittäjän koodaaman tapahtumasarjan. Kontrolleri ja malli kommunikoivat toiminnon suorittamisesta yhtäaikaisesti toistensa kanssa, jolloin käyttäjän näkymä, eli käyttöliittymä muuttuu välittömästi. Kun jokin muuttuu näkymässä, malli muuttuu ja vastaavasti kun malli muuttuu, näkymä muuttuu. Tämä on kaksiteinen datan sitominen (two-way databinding).

Syy, miksi kohtasin paljon perustason ongelmia AngularJs:sän koodaamisessa oli React Native taustani, joka vaikutti siihen miten lähestyin AngularJs ongelmia. React Native käyttää yksisuuntaista datan sitomista, jossa kaikki data siirtyy sovelluksen juuresta komponentteja pitkin. Koetin sovittaa oppimiani metodeja väärään kontekstiin, joka tuotti suurimman osan kohtaamistani vaikeuksista. Tulevaisuudessa osaan tarkastella sekä AngularJs, että React Native sovelluksia siitä arkkitehtuurin näkökulmasta, jota sovellus vaatii.

4 Pohdinta ja päätelmät

Päiväkirjaopinnäytetyön kirjoittamisen aikana front-end ja back-end kehittäjän taitoni kehittyivät useilta osa-alueilta, erityisesti React Nativen ja JavaScriptin saralta. React Nativen osalta tärkeimmät oppimisen osa-alueet ovat olleet React Nativen ominaisuudet ja toimintatavat, joiden kautta olen syventänyt käsitystäni siitä, millaisia sovelluksia React Nativella voi kehittää ja miten kehitysprosessi tapahtuu.

Reactin dokumentiossa on artikkeli, joka opettaa miten ”ajatella Reactisti”. Tämä viittaa tapaan visualisoida sovelluskokonaisuuden tai ominaisuuden koodaus Reactin hyvien toimintatapojen mukaisesti siten, että (React Native) sovellus on nopea ja skaalautuva. Taitoni ajatella Reactisti on kasvanut merkittävästi verrattuna opinnäytetyön kirjoituksen aloitushetkeen. Hahmotan miten sovelluksen komponenttihierarkia suunnitellaan siten, että sovellusta voi skaalata ja jatkokehittää ketterästi. Kehityksessä ei tarvitse turvautua ongelmien kiertoihin ja väliaikaisiin ratkaisuihin, kun kehitys on tehty alusta asti React Nativen hyvien toimintatapojen mukaisesti. Opinnäytepäiväkirjan kirjoittamisen alussa minulla oli vain pintapuolinen ymmärrys useista React Nativen peruskonsepteista, jonka vuoksi kirjoitin koodia, joka ei kuitenkaan ollut kirjoitettu ideaalilla tavalla. Kehittymiseni sekä syventyneen ymmärrykseni myötä kirjoitin osan aiemmin tehdyistä koodista uusiksi, jolloin sovelluksesta tuli nopeampi ja helpommin jatkokehitettävä.

VAIHDE-sovellusta kehittäessäni kohtasin toistuvasti ongelmia, jotka johtuivat freelancer työnä tilatun sovelluksen pohjan kehittäjän koodaamasta epäoptimaalisesta koodista. Tästä huolimatta ja osittain tämän kautta opin ajatella Reactisti, sillä ongelmat liittyivät usein koodiin, jossa ei hyödynnetty niitä React Nativen ominaisuuksia ja periaatteita, jotka tekevät React Native kehityksestä nopeaa ja helppoa. Koodin lukeminen ja muokkaaminen opetti React Native koodauksen lisäksi sitä, miksi on elintärkeää kirjoittaa alusta asti mahdollisimman hyvää ja huolellista koodia. Huolellisella koodilla tarkoitan tässä yhteydessä tapaa koodata, jossa koodin sekaan ei jää sellaisia rivejä, jotka eivät tue sovelluksen toimintaa. Niin webkehityksessä kuin mobiilikehityksessäkin on tärkeää, että kehitys suuntatuu aina joko uusien ominaisuuksien kehitykseen, tai olemassaolevien ominaisuuksien muokkaamiseen. Jos sovelluksen jatkokehitys vaatii usein koko aiemmin koodatun komponenttihierarkian korjausta tai uusimista, kehitysprosessista tulee pitkä, vaivalloinen ja turhauttava, eivätkä sovelluksen rakenteelliset ongelmat tule korjatuiksi, vaikka jokin pienempi ongelma saataisiinkin ratkaistua.

Jako front- ja back-end kehittäjien välillä ei ole työpaikoilla aina tarkka. Työmarkkinoilla front-end kehittäjiltä toivotaan usein back-end osaamista tai vähintäänkin ymmärrystä

perus konsepteista. Google Firebase ja sen mahdollisuudet tulivat tutuiksi opinnäytepäiväkirjan kirjoittamisen aikana useiden tehtävien yhteydessä. Vaikka Firebase ei ole kovin laajalti skaalautuva back-end, on sen opettelu antanut uutta ymmärrystä siitä miten reaaliaikainen back-end toimii. Back-endin toimintaan tulee tosinaan kiinnittää erityistä huomiota silloin kun suunnitellaan ja koodataan jotain käyttöliittymän komponenttia tai toimintoa. Vastaan voi tulla tilanteita, joissa esimerkiksi jonkin back-end toiminnon hitaus voi vaikuttaa siihen, miten datan palauttaminen käyttäjälle koodataan käyttöliittymään, jolloin front-end kehittäjän tulee olla tietoinen tietyistä rajoitteista ja vaatimuksista käyttöliittymälle. Uskon, että tulevaisuudessa tulen hyödyntämään Firebasea omissa projekteissani sen helppokäyttöisyyden ja reaaliaikaisuuden takia.

React Native ja mobiilikkehitystaitojen ohella merkittävin oppimiskokemukseni on ollut JavaScript-ohjelmoinnissa kehittyminen ja JavaScript kielen ymmärryksen syventyminen. Aloittaessani tämän opinnäytepäiväkirjan kirjoittamisen yksi päätavoitteeni oli oppia ymmärtämään syvemmillä tasolla, miten JavaScript toimii ja mitä todellinen ohjelmointi JavaScriptillä on. Webkehityksessä suuri osa käytettävästä JavaScriptista on kokoelma yksinkertaisia käskyjä, jotka eivät vaadi merkittävää abstraktia hahmottamista. Työtehtävien yhteydessä törmäsin toistuvasti tehtäviin, joiden ratkaisua ei voinut koodata ns. ”lennosta”, vaan ratkaistava ongelma tuli hahmottaa kokonaisuudessaan ennen ensimmäisen rivin kirjoittamista. Näissä tehtävissä tuli olla selkeä käsitys siitä, miten aion jakaa ongelman osiin ja miten näistä osista kirjoitetaan koodi, joka on kirjoitettu hyvien toimintatapojen mukaisesti.

Vaihdopalvelun raportit-sivu, VAIHDE-sovelluksen tilatieto-ominaisuus ja latausanimaatiot latautuville käyttäjätietolistoille, sekä firebasen käyttöliittymä ovat esimerkkejä ohjelmointipulmista, jotka haastoivat minua ohjelmoijana. Erityisesti edellemainitut tehtävät vaativat ymmärrystä JavaScriptin peruskonsepteista, kuten scope, funktio, array, datatyytit ja erilaiset JavaScript metodit. Omalla kohdallani voisin todeta, että vaikka ymmärrys mainituista JavaScript konsepteista syventyi opinnäytetyön kirjoittamisen aikana, ei ole varsinaisesti mitään koodaamisen osa-aluetta jossa minulla olisi kuitenkaan todellista ammatillista itsevarmuutta. Olen työssäni havainnut useaan otteeseen, että on helppoa kopioida koodia muualta ohjelmasta tai internetistä ja saada jokin toiminto toimimaan, ilman ymmärrystä miksi juuri kyseinen koodi toimii. Työolosuhteissa uuden oppiminen jokaisesta koodirivistä on usein toissijaista. On tärkeämpää saada päivitys tai tuote asiakkaille mahdollisemman nopeasti. Valmista koodia kopioimalla kehittäjä voi saada paljon aikaiseksi, mutta kun vastaan tulee ongelma johon ei löydy apua muualta, on edessä todellinen pulma.

Kehittäjänä tahdon kehittää syvällistä ymmärrystä siitä, miten JavaScript, React, sekä React Native toimivat ”konepellin alla”. Kun tekniikoiden kivijalka on hyvin ja perusteellisesti valettu, uusia toimintoja, ominaisuuksia ja skriptejä on paljon nopeampi kehittää toimiviksi ja skaalautuviksi. Päiväkirjan kirjoittaminen ja tekstien lukeminen tätä pohdintaa kirjoittaessani auttoi hahmottamaan, mitkä ovat vahvuuteni ja heikkouteni web-kehittäjänä, ja mihin minun tulisi keskittyä tulevaisuudessa, jotta saavutan kokonaisvaltaisemman ja erikoistuneemman osaamistason. Valitsemani kirjallisuus auttoi työtehtävissäni vaihtelevalla menestyksellä. React.js essentials ja Learning React Native-kirjat auttoivat ymmärtämään React Nativen propseja, statea ja elinkaarimetodeja siten, että sain uuden tiedon avulla ratkaistua useita pieniä ja suuria ohjelmointipulmia VAIHDE-sovelluksesta. Harmikseni en kuitenkaan päässyt koodaamaan laajoja komentoja, joissa olisin voinut hyödyntää Functional Programming in JavaScript-teoksesta oppimiani tekniikoita.

Tulevaisuudessa aion omissa opinnoissani syventyä erityisesti JavaScriptiin. Perus JavaScriptiin syventyminen on kaikista tärkein itseopiskelun osa-alue minulle, sillä React Native, React, AngularJs ja Node.js perustuvat siihen. Mitä paremmin tunnen JavaScriptin, helpommin voin ottaa urallani käyttöön uusia JavaScript kirjastolaajennuksia ja teknologioita.

React Native on oppimisen prioriteeteissä seuraavana. Koin mobiilikehittämisen mielenkiintoisaksi ja palkitsevaksi prosessiksi ja koen React Nativen JavaScript kirjastona erittäin kiinnostavaksi. Uskon, että tulevaisuudessa React Native osaaminen on hyödyllinen valttikortti urallani, sillä natiivi mobiilikehitys on kallis ja aikaavievä prosessi, joka voidaan parhaimmillaan sivuuttaa täysin React Nativen ansiosta.

Uusien JavaScript kirjastojen, kehysten ja tekniikoiden oppimisen lisäksi kenties arvokkain oppimisen osa-alue, jolla kehityin päiväkirjan kirjoittamisen aikana, on debuggausprosessi. Sen lisäksi että opin paikantamaan yksinkertaisten toimintavirheiden syyt koodista nopeasti ja tehokkaasti, opin tulkitsemaan vaikeampien virheilmoitusten syitä paremmin. Debuggaus on yksi haastavimpia kehittäjän työn aspekteista, sillä usein virheilmoituksen korjaaminen vaatii kehittäjältä kokemusta ja näkemystä useista teknisistä seikoista liittyen koodiin, joka ei toimi. Erinäisiin ohjelmointipulmiin voi aina etsiä tietoa internetistä jolloin ratkaisu on usein helposti saatavilla. Debuggaus tuo oman haasteensa siinä, että se on usein läheisesti sidottu siihen koodiyhteyteen, jossa se ilmenee. Virheilmoitus voi, metaforaa käyttäen, ilmoittaa aamukiireessä töihin lähtevälle, että ”avaimet ovat hukassa”, mutta se missä avaimet ovat ja miksi avaimet ovat hukassa, on kokonaan toinen ongelma. Opinnäytetyön aikana kartutin kokemusta virheiden syiden

tunnistamisessa ja korjaamisessa sekä koodissa, että muissa kehitysprosessin vaiheissa, kuten esimerkiksi React Native sovelluksen buildaamisessa ja päivittämisessä.

Lähteet

Adhithi R. 2018. Props and State in React Native explained in Simple English. Luettavissa: <https://codeburst.io/props-and-state-in-react-native-explained-in-simple-english-8ea73b1d224e>

Kirjoittaja tuntematon. Callback Hell. Luettavissa: www.callbackell.com

Open Signal, 2018. Android Fragmentation (August 2015). Luettavissa: <https://opensignal.com/reports/2015/08/android-fragmentation/>

Lucas M. 2017. Google attacks Android fragmentation, pushes use of newer API, OS versions. Luettavissa: <https://www.computerworld.com/article/3242995/android/google-attacks-android-fragmentation-pushes-use-of-newer-api-os-versions.html>

Advanced Kittenry. Arkkitehtuuri ja MVC. Luettavissa: <https://advancedkittenry.github.io/koodaaminen/arkkitehtuuri/index>

Fedosejev A. 2015. React.js Essentials. PACKT Publishing. Englanti.

Mantyla D. 2015. Functional Programming in JavaScript. PACKT Publishing. Englanti

Eisenman B. 2016. Learning React Native. O'Reilly Media. Yhdysvallat

Liitteet

Liite 1. Käsitteet

Webkehityksen yleiskäsitteitä

- **back-end** – Sovelluksen tai järjestelmän palvelintaso.
- **back-end** – Sovelluksen tai järjestelmän palvelintaso.
- **front-end** – Sovelluksen tai järjestelmän käyttöliittymä, eli käyttäjän näkemän sisältö.
- **front-end** – Sovelluksen tai järjestelmän käyttöliittymä eli käyttäjän näkemän sisältö.
- **DOM** – document object model on tapa kuvata sivun rakenne ”vanhempi/lapsi” mallilla, jossa sivun rakenteelliset elementit perivät vanhempiensa ominaisuuksia ja noudattavat vanhemmillensa asetettuja sääntöjä.

JavaScript käsitteitä

- **bugi** – Sovelluksen toimintahäiriö.
- **debuggaus** – Sovelluksen toimintahäiriöiden syyn selvittäminen ja korjaaminen.
- **hyvä/huono toimintatapa** –Tapa koodata toiminto, joka vastaa tai ei vastaa tapaa, miten toiminto tulisi toteuttaa tietyssä koodikielessä, koodikirjastossa tai koodikehyksessä. Johtuu yleensä kyseisen koodikielen, koodikirjaston tai koodikehyksen periaatteiden huonosta tuntemuksesta.
- **syntaksi** – Säännöt koodikielen kirjoitusasulle ja koodikielelle varatuille sanoille, kuten esimerkiksi ”var”, ”return” tai ”function”.
- **scope** – Scope on JavaScriptin ominaisuus, missä eri objekteilla on ”pääsy” eri koodialueille. Jos jokin objekti on ulkona toisen objektin, esimerkiksi funktion, scopesta, kutsuttaessa tätä objektia sen arvo on määrittelemätön.
- **boolean** – Datatyyppi. Boolean voi olla joko ”true” (totta) tai ”false” (epätotta).
- **skripti** – Komentojono sovelluksessa, joka tuottaa ohjelmoijan haluaman lopputuloksen.
- **array** – Elementti, johon koodari voi varastoida tietoa. Esimerkiksi ”Hedelmät = [omena, päärynä, appelsiini]”.
 - **forEach()** – Arraymetodi. Metodi suorittaa sulkuhin määritellyn komennon arrayn jokaiselle elementille.
- **metodi** – Toiminto, joka voidaan suorittaa JavaScript objektilla. Esimerkiksi .log() on metodi ”console” objektilla.

- **unix aikaleima** – Millisekuntimäärä, joka on kulunut vuoden 1970 alusta. Yleisessä käytössä oleva metodi mitata ajan kulu koodissa.
- **return** – JavaScript keyword, joka keskeyttää funktion.
- **logittaminen** – Konsoliin logittaminen. Konsoli on kehitysympäristön (esimerkiksi selaimen) osa, jonne kehittäjä voi lähettää viestejä koodin testaamiseksi.
- **juuri** – Tiedostohierarkian ylimmän tason tiedosto.
- **MVC** – Model, View, Controller (Malli, Näkymä, Kontrolleri). AngularJs:sän toimintaperiaate.
- **for-looppi** – Iteraatio, eli koodikokonaisuus, joka toistuu kunnes tietty ehto on totta, eli ehdon boolean arvo muuttuu epätodesta todeksi.
- **while-looppi** – Iteraatio, eli koodikokonaisuus joka toistuu ”niin kauan kuin” jokin ehto ei ole totta, eli ehdon boolean arvo muuttuu truesta falseksi.

Firestore käsitteitä

- **firebase** – Googlen tietokanta, jota VAIHDE-sovellus käyttää.
- **snapshot** – Firebasen toiminto. Snapshot on JavaScript metodi, joka ottaa nimensä mukaan ”valokuvan” tietokannan osan sen hetkisestä tilasta ja palauttaa datan.
- **token** – Uniikki tekstinpätkä, joka identifioi mobiililaitteen tiettyyn käyttäjään.
- **indexointi** - Datan palauttamisen sääntö. Kehittäjä voi asettaa tietokantaan ennaltamääräytyjä osia, joista firebase palauttaa datan nopeasi.
- **konsoli** - Konsoli on firebasen osa, jonne kehittäjä voi lähettää viestejä koodin testaamiseksi.
- **kuuntelufunktio** – Funktio, joka ajetaan kun sille asetettu ehto tapahtuu. Esimerkiksi ”kun jokin kohta käyttäjän tiedoissa muuttuu”
- **notifikaatio** – Ilmoitus, jonka sovelluksen käyttäjä voi vastaanottaa mobiililaitteeseen.

React Native käsitteitä

- **buildaaminen** – Englannin kielen sanasta ”build/building”, eli kokoaminen. Tämän opinnäytetyön yhteydessä tarkoittaa JavaScript-mudossa olevan koodin muuttamista Androidin .apk tai Applen .app mutoon, joka on sovelluksen tiedostomuoto.
- **touchableopacity** – Responsiivinen painiketyyppi React Nativessa.
- **jsx** – React Nativen DOM elementti.

- **SDK** – “Software development kit”. Työkalupakki kehittämiseen, erikseen asennettava koodikirjasto.
- **maven** – Hallitsee projektin (sovelluksen) sisäisiä riippuvuuksia projektin käytänteiden välillä.
- **NPM** - JavaScript kirjastolaajennus, joka nopeuttaa sovellusten kehittämistä. Jos sovellukseen tarvitaan esimerkiksi kalenteri, on nopeampaa ladata valmis kalenteri-kirjastolaajennus, kuin koodata koko kalenteri itse.
- **props** – Englannin kielestä “prop, propertie”, eli elementin ominaisuus joka voidaan asettaa JSX elementille.
- **state** – Reactin ja React Nativen komponentti datan manipuloimiseksi. Statella voidaan hallita komponentin sisäisiä muutoksia.
- **navigaatio** – Painikkeista koostuva systeemi, joilla käyttäjä voi siirtyä sovelluksen eri sivuille.
 - **tabnavigator** – React Nativen navigaatiotyyppi. Navigoitavat sivut on asetettu ”vierekkäin” jolloin käyttäjä voi pyyhkäisyellä vierittää näkymän uudelle sivulle. Jokainen navigoitava sivu on nimeltään ”tab”.
 - **stacknavigator** – React Nativen navigaatiotyyppi. Navigoitavat sivut asetetaan toistensa päälle korttipakanomaisesti.
- **senderöinti** – Kutsu, joka palauttaa käyttäjälle sovelluksessa näkyvät elementit, eli näyttää halutun sivun.

Sekalaiset käsitteet

- **bitbar** – Internetpalvelu, johon kehittäjät voivat ladata sovelluksiaan virtuaalista testausta varten. Palvelussa voi simuloida sovelluksen toimintaa erilaisilla virtuaalisilla älypuhelimilla.
- **taski** – Englannin kielen sanasta ”task”, eli tehtävä. Ammattislangia jolla viitataan johonkin koodattavaan kokonaisuuteen.
- **interaktio** – Tapa, millä käyttäjä vuorovaikuttaa sovellukseen.
- **notifikaatio** – Ilmoitus, jonka sovelluksen käyttäjä voi vastaanottaa mobiililaitteeseen.
- **layeri** – Englannin kielen sanasta ”layer” eli taso. Kuvankäsittelyn termi, joka tarkoittaa eri kuvatasojen pinoamista toistensa päälle. Esimerkiksi irrallinen räjähdyssefekti liitetään jonkin toisen kuvan päälle tasona.
- **URL** – Merkkijono, joka osoittaa tiedoston paikan esimerkiksi tietokannassa.