Haaga-Helia
ammattikorkeakoulu Oy

Android sovelluskehittäjän arki startup-yrityksessä

Katri Vilonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

Päiväys 12.05.2016

Tekijä(t)
Katri Vilonen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Android sovelluskehittäjän arki startup-yrityksessä

Sivu- ja liite-sivumäärä
53

Opinnäytetyön otsikko englanniksi
Day-to-day life of an Android Application Developer in a startup

Tämä päiväkirjaopinnäytetyö seuraa Android sovelluskehittäjän työtehtäviä startup-yrityksessä keväällä 2016. Opinnäytetyössä raportoidaan päivittäistä työtä, josta joka viikon jälkeen tehdään analyysi. Raportointia on kuuden viikon jaksolta, 1.2. – 17.4.2016.

Työympäristönä toimii noin kymmenen henkilön startup-yritys. Opinnäytteen laatija toimii yrityksessä Android pääkehittäjänä. Työnkuva sisältää Android sovelluksen ylläpitämistä ja uusien ominaisuuksien kehittämistä, sekä uusien designien implementointia. Tehtävät yrityksessä sisältävät myös hiukan tehtävien jakoa ja rekrytointia, Android-tiimin koostuessa yhdestä henkilöstä.

Päivittäiset työtehtävien ja -tapojen perustelut validoivat kirjoittajan tietotaitoa ja tarjosivat tilaa reflektioille. Raportoinnissa myös havainnollistetaan Android-kehityksen ongelmanratkaisua ja analysoidaan eri ratkaisuja ohjelmistokehityksen maailmassa. Viikoittainen tutkimus tiedonhakuanalyysiä varten osoitti kirjallisuuden hyödyn käytännön menetelmiä valitessa. Raportoinnin aikana opinnäytteen laatijan työ- ja kommunikaatiotaidot ovat kasvaneet.

Kirjoittaja on havainnut, että työn onnistuminen vaatii laajaa osaamisen keräämistä sekä kommunikaatiotaitoja startup-yrityksen suhteellisen pienen työympäristön takia. Tämän lisäksi kirjoittaja näkee, että pienen yrityksen toiminnassa joutuu usein oman työnkuvan ulkopuolelle, mikä kasvattaa tarvetta uuden opiskeluun.

Asiasanat
Android, päiväkirjaopinnäytetyö, ohjelmistokehitys, startup-yritys, startup

Haaga-Helia
University of Applied Sciences Ltd.

Abstract

20th May 2016

Author(s)
Katri Vilonen

Degree programme
Business Information Technology

Report/thesis title
Day-to-day life of an Android Application Developer in a startup

Number of pages and appendix pages
53

This diary type thesis project followed an Android Developer's workload in a startup company in the spring of 2016. The diary consists of daily reports which at the end of the week are gathered into an analysis of the work week. The diary was maintained over a six-week period, from 1.2.2016 to 17.4.2016.

The working environment is a startup company with ten employees. The author is the company's main Android Developer. The job description covers maintaining the Android application, developing new features, and implementing new designs. As the Android team consists of one person, the tasks in the company also include some division of labour and recruiting.

The daily rationalisation of work methods has validated the author's knowhow and offered room for reflection. The reporting also demonstrates Android development centric problem solving methods and analyses software development. The weekly search for information has proven the importance of literature in the development of best practices. During the reflection period, the author's work and communication skills have been enhanced.

Due to the relatively small work environment of a startup company, the study notices that in order to succeed at the job one must develop an ever expanding skill set and excel at the art of communication. Additionally, the study concludes that in a small company you often have to be quick learners as they might have to go beyond their established work routine to do their share of the workload.

Keywords
Android, diary thesis, software development, startup company, startup

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena on seurata ja analysoida aloittelevan Android sovelluskehittäjän päivittäisiä työtehtäviä. Opinnäytetyö sisältää päiväkirjamerkintöjä kuuden viikon ajalta ja kirjoituksen aikaväli on 1.2. – 17.4.2016. Päiväkirjatyyppisen opinnäytetyön tavoitteena on seurata omaa kehitystä ja oppia uutta oman työn viikoittaisen analysoinnin kautta.

1.1 Yritys ja oma työympäristö

Singa Oy on aloittanut virallisesti toimintansa vuonna 2013. Singa aloitti tarjoamalla mobiili karaokepalveluaan Androidille, Windows Phonelle ja iOS:lle. Nyt palvelu on laajentunut selainsovellukseen, jonka löytää osoitteesta https://sin.ga. Seuraavien kuukausien aikana Singa tulee laajentamaan älytelevisioiden jännittävään maailmaan. Tällä hetkellä (2.2.2016) Singa on asennettu PlayStoresta noin 50 000 kertaa. Singalla on tällä hetkellä vakituisia työntekijöitä 10, joiden lisäksi töitä tekee myös joukko freelancereitä (Talouselämä, 11.11.2015). Tämän hetkisenä toimitilana toimii Fredrikinkatu 42:ssa sijaitseva startupkiihdyttämö, jossa sijaitsee myös startup-konferenssi Slushin toimisto. Singan CEO Atte Hujanen on myös Slushin perustajajäsen. Viimeisellä rahoituskierroksella Singa keräsi 525 000 euron siemenrahoituksen, jossa sijoittajina ovat Reaktor Ventures, Sisu Game Ventures ja joukko yksityissijoittajia (Talouselämä, 11.11.2015).

Ura Singalla alkoi vuonna 2015, kun aloitin työskentelyn nuorempana Android applikaatiokehittäjänä. Android kehittäjänä olen pääasiallisesti tehnyt front end puolen kehittämistä sekä toteuttanut erinäisiä uusia ominaisuuksia Singan Android sovellukseen. Teen töitä koulun ohella ja työaikani on hyvin joustava.

Singalla työtehtävät ovat jakautuneet segmentteihin, joista jokainen on vastuussa. Olen ainoa Androidille omistautunut työntekijä, joten vastuullani on jakaa Android sovellukseen liittyvät työtehtävät itselleni, fullstackeille ja freelancereille niin että saamme mahdollisimman paljon aikaiseksi rajatussa ajassa. Toteutamme projektimme ketterästi Scrum-menetelmällä. Sprinteittäin luomme itsellemme roadmapin, jonka mukaan teemme töitä. Viikottain pidämme palaverin, jossa kerromme muille, miten edistymme ja miten jatkamme seuraavalla viikolla.

Taulukko 1. Singan vuoden 2016 ensimmäisen sprintin roadmap Androidille

vko	4 - 5	6 - 8	9 - 10
Android	Playlist Notifications Optimointi	Soittojono Search	Chromecast

1.2 Työtehtävässä tarvittava osaaminen

Android sovelluskehittäjän työtehtävät vaativat korkeakoulutasoista koulutusta ja jatkuvaa mielenkiintoa jatkuvasti kehittyvää alaa kohtaan. Koulussa ja vapaa-ajalla saatu teoriatieto ja harjoitustyöt ovat antaneet hyvän pohjan ohjelmistokehitykseen. Työharjoittelu samassa yrityksessä on ollut omakohtaisesti tärkeää oppimisen ja kehitysalustaan tutustumisen kannalta. Suurin osa mobiilituotekehityksen kirjallisuudesta on englanniksi, joten työ vaatii englannin kielen osaamista. Tärkeimmät vaatimukset ovat puhtaan koodin kirjoitustaito, UI/UX keskeinen ajattelu, hyvä java ja Android SDK tuntemus. Myös hyvät ryhmätyötaidot ovat eduksi, koska töitä tehdään yhdessä pienen tiimin kanssa. Yleisesti ottaen on katsottu, että ohjelmistokehittäjä työskentelee erilaisissa suunnittelu- ja kehitystehtävissä.

Android sovelluskehittäjän roolissa Singalla tarvitaan tietämystä Android Studion käytöstä, jonka perusteet voi opetella kirjasta Android Studio Application Development (Zapata, 2013). Kirjassa käsitellään muun muassa UI:n luonti ja debuggaus-mahdollisuudet Android Studiolla. Toinen hyvä opus Android kehittäjien luettavaksi on Professional Android 4 Application Development (Meier, 2012), jossa sukelletaan syvemmälle UX:n kiehtovaan maailmaan, antamalla käytännön esimerkkejä esimerkiksi GUI elementtien animaatioista, jotka ovat hyvin tärkeitä käyttäjälähtöisessä kehitysprosessissa. Olen myös kerryttänyt osaamistani käymällä Helsinki Android Meetupeissa, joita järjestetään noin kerran kuussa Android kehittäjille. Meetupeissa on yleensä noin kolme mielenkiintoista ja ajankohtaista esitelmää, joiden jälkeen esitelmöijät vastaavat aiheeseensa liittyviin kysymyksiin. Tämänlaiset tapahtumat ovat elintärkeitä nopeasti kehittyvällä alalla ja antavat paljon tarvittua informaatiota uusista ja hyödyllisistä kolmannen osapuolen kirjastoista sekä valaisevat hämäräksi jääneitä ongelmia ADB:ssä.

1.3 Käsitteet

Aktiviteetti — Aktiviteetit ovat Androidin sisäisiä luokkia, missä käyttäjä voi keskittyä yhteen tiettyyn sovelluksen toiminnallisuuteen.

Callback	Suoritettava metodi, joka lähettää argumentin toiselle metodille, jonka vuorostaan pitäisi lähettää vastaus takaisin.
Debuggaus	Virheenkorjaus
Eclipse MAT	Eclipse Memory Analyzer Tool, muistin analysointi työkalu, joka ottaa vastaan heap dumpeja.
Fragmentti	Fragmentit ovat osa sovelluksen käyttöliittymää tai osa käyttäytymistä mikä voidaan sijoittaa aktiviteettiin.
Git	Jaettu versionhallintajärjestelmä, joka tallentaa tehdyt muutokset tiedostoon tai joukkoon tiedostoja, jotta tiettyihin versioihin voidaan palata myöhemmin.
Git flow	Versionhallinnan kehityshaarojen standardien omainen käyttö.
Heap dump	Tilannevedos Java-prosessin muistinkäytöstä.
Kanban	Projektihallinnan viitekehys, jota käytetään ketterässä ohjelmistokehityksessä.
Konteksti	Konteksti-luokka on rajapinta sovellusympäristön globaaliin informaatioon.
Lähdekoodi	Tietokoneohjelman tai sovelluksen ohjelmointikielellä kirjoitettua tekstiä, jonka kääntäjäohjelma kääntää suoritettavaan muotoon niin että sen voi ajaa.
Merge	Versionhallinnassa merge tehdään, kun halutaan lisätä yhdessä haarassa tehdyt muutokset toiseen haaraan.
Merge conflict	Merge conflict tapahtuu, kun Git ei tiedä miten yhdistää tiedoston eri versioita. Merge conflicteja sattuu esimerkiksi, kun kaksi ihmistä on muokannut samaa riviä tai toinen on muokannut tiedostoa ja toinen poistanut tiedoston.

Pino	Abstrakti tietotyyppi, jossa viimeksi pinoon tullut operaatio poistuu ensimmäisenä, Last In First Out -periaatteen mukaan.
Prosessi	Ohjelman ilmentymä, joka on käyttöjärjestelmän ajossa.
Pull	Versionhallinnassa pull-komento päivittää jaetun git-hakemiston muutokset lokaaliin ympäristöön.
Push	Versionhallinnassa push-komento päivittää lokaalit muutokset jaettuun git-hakemistoon.
Repository	Versionhallinnassa oleva varasto, joka voi olla käyttäjällä paikallisesti koneella tai kehitystiimin käytössä verkossa olevalla palvelimella
Roadmap	Etenemissuunnitelma, joka Scrumin periaatteiden mukaan tehdään ennen jokaista sprintiä.
Scrum	Projektihallinnan viitekehys, jota käytetään yleisesti ketterässä ohjelmistokehityksessä.
Skripti	Komentotulkissa voi ajaa skripti-tiedoston, joka suorittaa sisältämänsä komentosarjan.
Sorsa	Slangi sana lähdekoodille, joka on lähtöisin tämän termin englannin kielisestä versiosta source code.
Sprint	Scrumin mukainen, yleensä alle kuukauden pituinen aikajakso, jolla pyritään saamaan tuoteversio asetettujen tavoitteiden pohjalta valmiiseen tilaan.
Startup	Suhteellisen nuori kasvuhakuinen yritys.
Säie	Sovelluksen itsenäisesti suorittuva osuus, joka on prosessia kevyempi.
Weekly meet	Scrumin mukainen viikoittainen tapaaminen.

1.4 Lyhenteet

ADB	Android Debug Bridge on debuggaustyökalu Androidille.
ALSA	Advanced Linux Sound Architechture, eli Linux-kernelin äänirajapinta
API	Application programming interface, eli ohjelmointirajapinta, jonka avulla erinäiset ohjelmat voivat vaihtaa tietoja keskenään ja tehdä pyyntöjä.
APK	Android Application Package, eli Android-sovelluspaketti
AV	audiovisuaalinen, audiota ja videota yhdistävä
CEO	Chief executive officer, eli toimitusjohtaja
GUI	Graphical User Interface, eli graafinen käyttöliittymä koostuu käyttöliittymäelementeistä
IDE	Integrated Development Environment, eli ohjelmointiympäristö on ohjelma, jolla ohjelmia voi suunnitella ja toteuttaa.
JSON	JavaScript Object Notation, on yksinkertainen tiedostomuoto tiedonvälitykseen.
OOM	Out of Memory error, on virhe, joka ilmenee muistitilan loppuessa.
OS	Operating System, eli käyttöjärjestelmä mahdollistaa ohjelmien toiminnan.
PCI	Peripheral Component Interconnect, väylä, jonka avulla komponentteja voidaan liittää tietokoneeseen.
SDK	Software Development Kit:llä viitataan kehitystyökaluihin, jotka mahdollistavat ohjelmistokehityksen.
UI	User Interface, eli käyttöliittymä viittaa käyttäjän ja laitteen väliseen yhteyteen.
URL	Uniform Resource Locator, eli WWW-osoite.

USB	Universal Serial Bus, väylä, jonka avulla komponentteja tai oheislaitteita voidaan liittää tietokoneeseen.
UX	User Experience, eli käyttäjäkokemus viittaa kokonaisvaltaiseen käyttökokemukseen.
VR	Virtual Reality, eli virtuaalinen todellisuus.
XML	Extensible Markup Language, on kieli, jolla Android-sovelluksen graafisen käyttöliittymän voi piirtää.

2 Lähtötilanteen kuvaus

Tässä luvussa kerron tarkemmin omista työtehtävistäni ja siihen vaadittavasta osaamisesta. Lisäksi kerron, kuinka pääsin vaadittavalle osaamistasolle.

2.1 Oman nykyisen työn analyysi

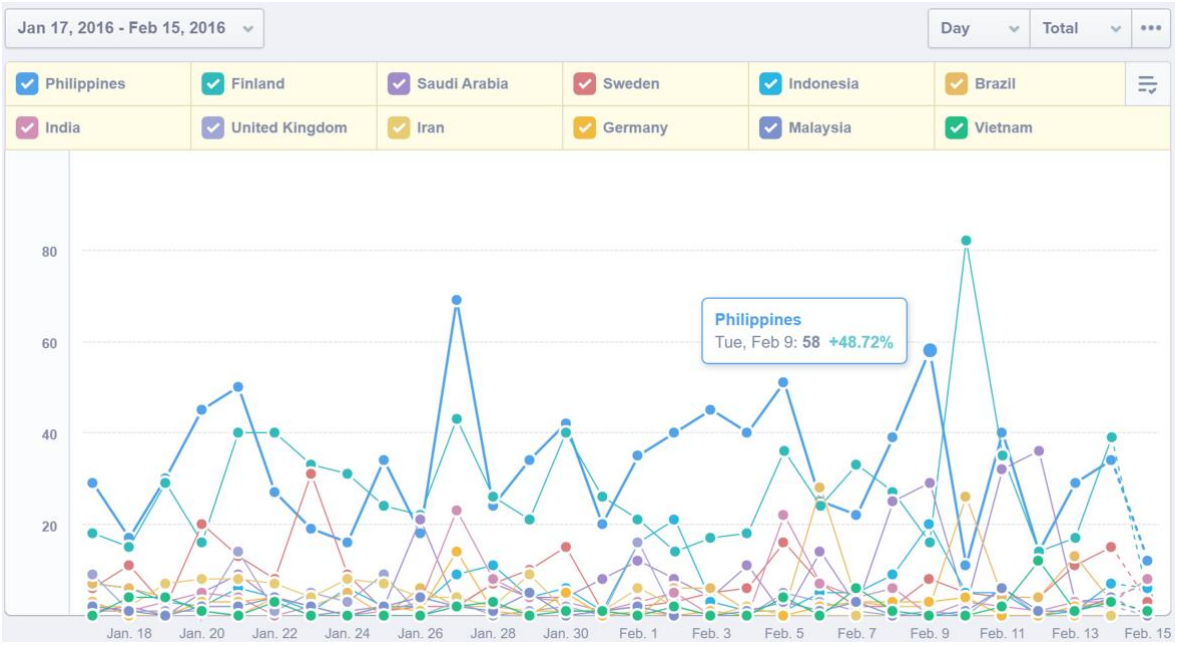
Tehtäväni Singalla on toimia Android sovelluskehittäjänä. Työtehtävistä on tarkempi kuvaus taulukossa 2.

Taulukko 2. Työtehtävät

Työtehtävä	Kuvaus
Vanhojen ominaisuuksien kehitys ja ylläpito	Kun sovelluksemme kasvaa, minun täytyy testata ja muokata vanhoja ominaisuuksia, että ne ovat yhteensopivia uudistusten kanssa.
Uusien ominaisuuksien implementointi	Uusien ominaisuuksien kuuluu olla yhteensopivia taustajärjestelmämme kanssa. Teen tiukkaa yhteistyötä Project Managerimme kanssa, että saisimme otettua kaiken hyödyn taustajärjestelmäpalveluistamme.
Chromecast-sovelluksen tekeminen	Alustavasti suunnittelen ja toteutan Android sovellukseemme Chromecast-tuen, että peliämme voi pelata Chromecast-mediasoittimen avulla television kautta.
Designien implementointi	Designerimme tekee iOS-pohjaisia suunnitelmia, joita joudun muokkaamaan, että ne olisivat mahdollisimman yhdenmukaiset Androidin Material Designin kanssa.

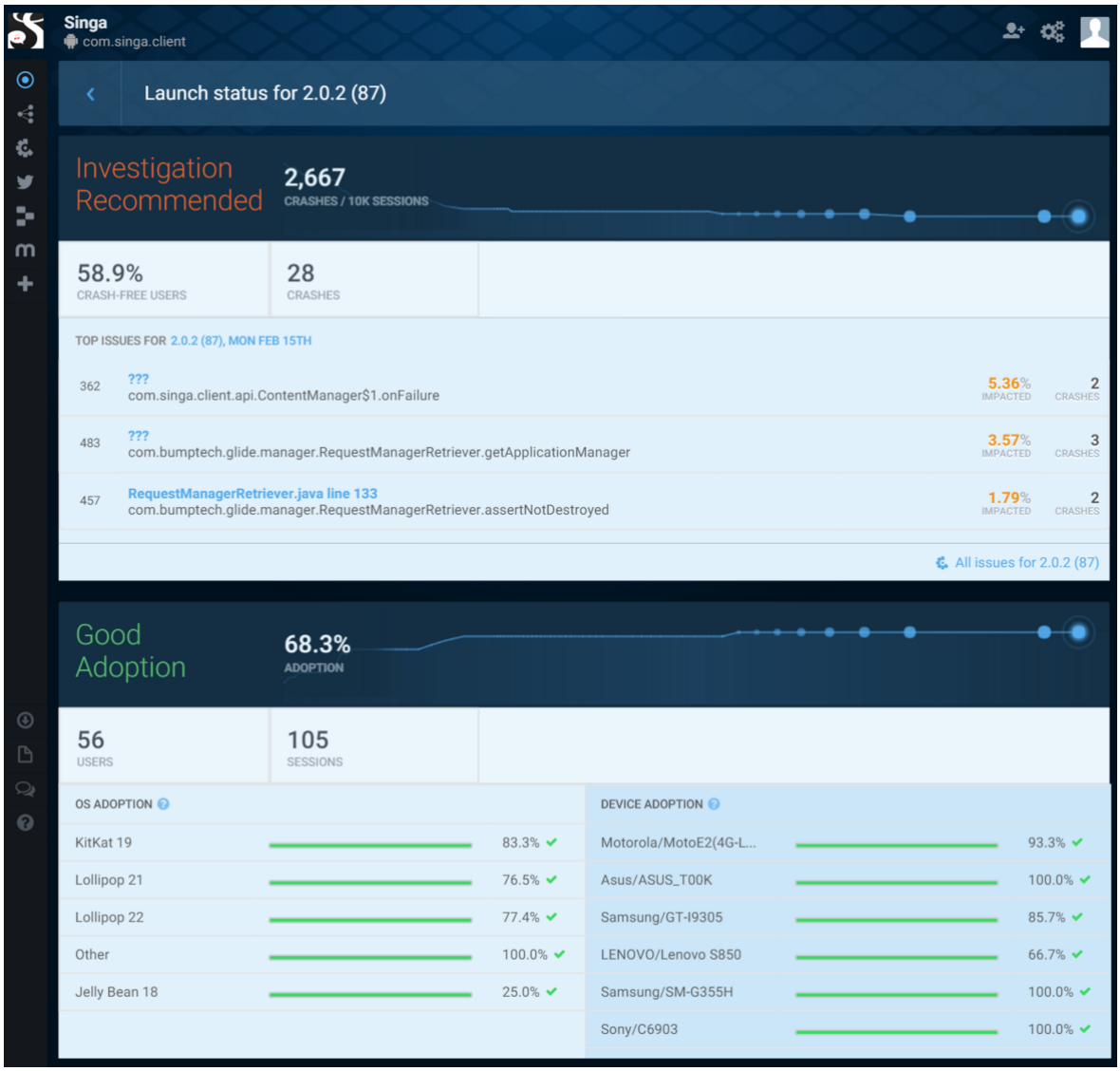
Työnteon ohessa olen oppinut monia uusia asioita. Esimerkiksi Android Studion optimointi, JSONin parsiminen – en edes aikaisemmin tiennyt mitä JSON on, ketterä ohjelmistokehitys, Mixpanelin ja Fabricin käyttö appdatan havainnollistamisessa, alphatestaus ja julkaisu AppStoressa, signeeratun APK:n generoiminen, Monkey-testaus, LeakCanary-kirjaston käyttö, miten luoda Navigation Drawer, Picasso-kirjaston käyttö,

kahvinkeitto Aeropress-menetelmällä, sovelluksen pitchaaminen, mukautettujen graafisten elementtien luonti näkymään, teeman vaihto ja Android Material Designin tarkempi hallussapito ovat asioita mitä olen joutunut kantapään kautta oppimaan työssäni. Mixpanel on analytiikkapalvelu, jolla voi seurata käyttäjän vuorovaikutusta sovelluksen kanssa. Mixpanelin avulla voimme havainnollistaa esimerkiksi suurimmat käyttäjämaamme, mikä on käytetyin app versiomme, millä puhelimilla käytetään eniten Singaa ja ketkä käyttäjät käyttävät appiamme eniten (Kuva 1). Kun mietimme laajennuksia appiin, on myös hyvin tärkeää tarkistaa Mixpanelista mitä Android OS versiota käytetään eniten, että voimme arvioida aiheuttaako vanhentuneen koodikonvention muuttaminen ongelmia suurelle osalle käyttäjistämme.



Kuva 1. Mixpanelin generoima graafinen kuvaaja Singan käyttäjämaista.

Fabricin Crashalyticsillä voimme saada raportteja, kun sovelluksemme kaatuu käyttäjän puhelimella. Fabric kertoo mitä kaatumisia tapahtuu eniten, missäkin sovellusversiossa, ja millä laitteilla ja käyttöjärjestelmällä nämä kaatumiset ovat käyneet (Kuva 2).



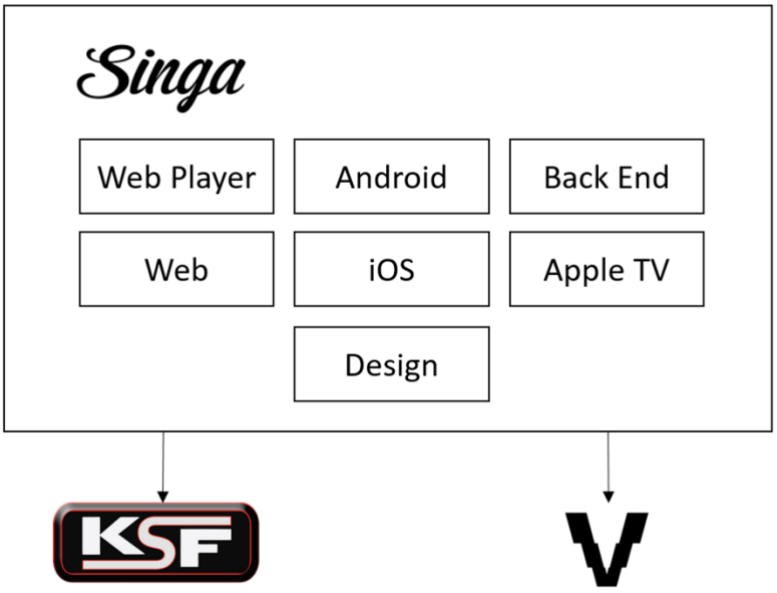
Kuva 2. Fabricin launch status näkymä

Tiedot, joita tarvitsen työtehtävien suorittamiseksi, koostuvat Javan ja XML:än perusteista ja Android-näkymien hierarkiasta. Olen joutunut hyödyntämään tiedonhakutaitojani ja aktiivista kiinnostumistani alaa kohtaan, että olen voinut pysyä työtehtävieni tasolla. Koska olen aloittelija, työtehtävistäni suoriutuminen vaatii vielä jonkin verran työkavereiden ohjeistusta, mutta teen yleensä hyvin itsenäisesti töitä sekä toimistolla että kotona. Työtehtäväni pyrin aina tekemään huolellisesti ja hätiköimättä, parhaani mukaan ja olen saanut tästä kiitosta aikaisemmalta Android pääkehittäjältä.

Kehittymiseni ammatissani on vielä alkutekijöissään. Ohjelmistokehityskokemus ja kiinnostus alaa kohtaan luovat hyvän pohjan uudelle uralle. On todella tärkeää, että pystyy hahmottamaan työnsä kokonaiskuvan ja jatkuvasti kehittää itseään ohjelmistokehittäjänä. Jatkuva oppiminen ja panostus työtehtäviin ovat vaatimuksia ohjelmistokehittäjän uralla.

2.2 Sidosryhmät työpaikalla

Koska Singa on kymmenen työntekijän startup-yritys, sisäisiä sidosryhmiä ei erityisemmin ole. Jokaisella työntekijällä on periaatteessa omat vastuualueensa. Olen laskenut sisäiset vastuualueet sisäisiksi sidosryhmiksi kaaviossa 1, enkä tästä syystä ole laskenut erinäisiä freelancereitä mukaan sidosryhmiin. Alla kaaviossa 1 on kuvattu läheisimmät sidosryhmät, joiden kanssa olemme tekemisissä. Läheisimmät sisäiset sidosryhmät ovat Android, josta olen periaatteessa vastuussa. Teen myös yhteistyötä läheisesti iOS puolen kanssa, koska designit tehdään pääasiallisesti iOS:lle ja teemme työtä saadaksemme mobiilisovelluksemme mahdollisimman yhdenkaltaisiksi. Designit tehdään myös erikseen selainsovellukseemme ja web playerimme – eli pelinäkymä selainsovelluksessamme – tulisi näyttää myös yhtenevältä mobiilisovellusten pelinäkymäin kanssa. Taustajärjestelmien puolelta saamme kaiken tarvittavan tiedon, miten toteuttaa uusia ominaisuuksia sovellukseemme. Apple TV:n kanssa tulen olemaan hyvin läheisessä kanssakäymisessä Chromecastin tiimoilta. Chromecast ja Apple TV ovat hyvin erilaiset palvelut, mutta käyttäjäkokemuksen tulisi olla samankaltainen. Palvelumme on suunnattu kuluttajille, joten asiakkaamme ovat periaatteessa karaokepalvelun käyttäjät. Asiakkaidemme mielipiteet ovat hyvin tärkeitä kehitystyölle, ja ne johtavat designin muutoksiin ja antavat suuntaa työtehtävien tärkeysjärjestyksessä. Ulkoisina sidosryhminä voidaan pitää monikansallista KSF Entertainment Group Ltd:iä, jolta saamme lisensoitua karaokemusiikkia sovellukseemme. Sijoittajistamme Reaktor Ventures on aktiivisimmin mukana käymällä toimistollamme säännöllisin väliajoin katsomassa työntekoamme.



Kaavio 1. Singan sidosryhmät. Sisäiset sidosryhmät koostuvat pääasiallisesti yhdestä tai kahdesta työntekijästä.

2.3 Vuorovaikutustaidot työpaikalla

Vuorovaikutustaidot ovat erittäin tärkeitä työtehtävissäni, koska Singalla työntekijämäärä on niin pieni. Aloittelevana ohjelmistokehittäjänä joudun kyselemään paljon muilta työntekijöiltä, jotka ovat itseäni ammattitaitoisempia ja tämä vuorovaikutus muiden työtekijöiden kanssa on elintärkeää oman kehitykseni kannalta. Koska olen vastuussa Android-puolesta, myös tehtävien jakamisessa minun täytyy käyttää vuorovaikutustaitoja – enhän halua jakaa epäreilusti tehtäviä tai pakottaa ketään tekemään suhteellisesti katsottuna tylsiä tehtäviä.

Ryhmätyöskentelyssä on tärkeää pitää huolta, että jokainen ryhmässä työskentelevä tietää mitä tehdään ja mitä tullaan tekemään. Työtovereiden osaamistasot on hyvä tietää, kun heiltä tarvitsee kysyä apua tai delegoida heille oman osaamisen ulkopuolisia työtehtäviä. Kun omaa kiinnostusta näyttää aktiivisesti, on muistakin mukavampaa ohjeistaa työtehtävien parissa.

En ole itse tekemisissä ulkoisten sidosryhmien kanssa, mutta koska suurin osa yrityksen rahoista tulee sijoittajilta, suhteemme heihin on todella elintärkeää yrityksen toiminnalle. CEO Atte Hujanen pitää yllä yrityksen suhteita sijoittajiin – kuten Reaktor Venturesiin - ja sisällöntuottajiin – KSF Entertainment Groupiin.

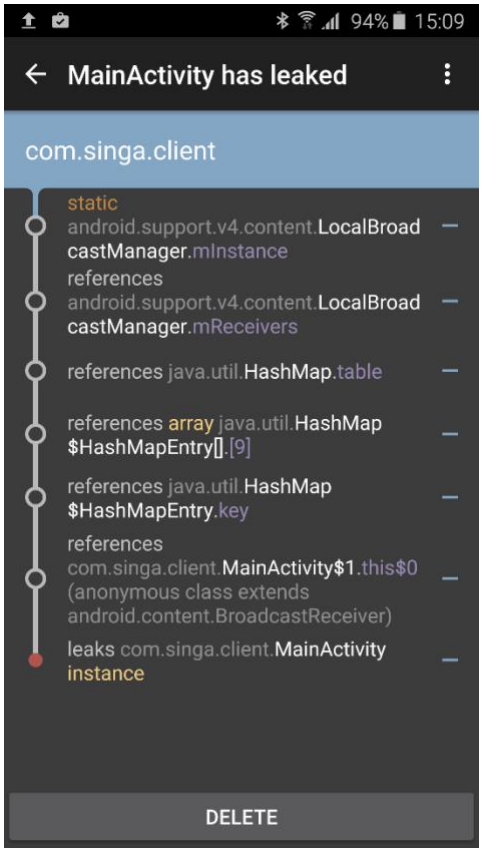
3 Päiväkirjaraportointi

3.1 Seurantaviikko 01

Maanantai 01.02.2016

Tämän viikon tavoitteeksi otan kolmannen osapuolen kuvanlatauskirjaston vaihdon Picassosta Glideen. Olen havainnoinut heap dumpeja, LeakCanary-kirjastoa ja Eclipse MATia apuna käyttäen, että suurin osa sovelluksemme kaatumisista johtuu OutOfMemory virheitä, jotka taas johtuvat kuvien latauksista.

Leak Canary-kirjaston avulla, jokaisesta muistivuodosta voidaan saada heap dump, joka on tilannevedos java prosessin muistinkäytöstä tiettynä hetkenä (Kuva 3). Leak Canary on todella helppo implementoida sovellukseen kuin sovellukseen, koska käyttääkseen LeakCanaryä, koodiin on lisättävä vain kaksi riviä (Kuvat 4 ja 5). Heap dumpien avulla voidaan löytää OutOfMemory virheiden syy. Eclipse MATin avulla sain selville, että suurin osa OOM:eista johtui kuvien lataamisesta ja siitä että Picasso-kirjaston takia ne eivät koskaan poistu muistista, latautuen uudestaan ja uudestaan (Kuva 6).



Kuva 3. LeakCanary-kirjaston löytämä muistivuoto. Kuvassa syyllinen OOM:iin löytyy Androidin sisäisestä luokasta, LocalBroadcastManagerista. Kuva otettu 12.12.2015

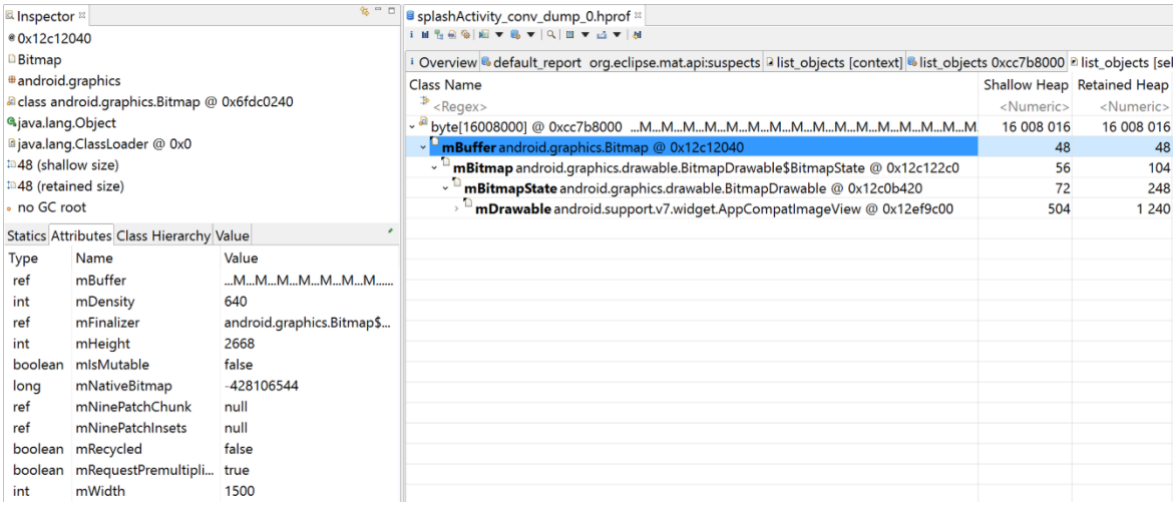
```
dependencies{
    debugCompile 'com.squareup.leakcanary:leakcanary-android:1.3.1'
}
```

Kuva 4. LeakCanary täytyy tuoda ensin build.gradlen riippuvuuksiin.

```
@Override
public void onCreate() {
    super.onCreate();
    singleton = this;
    contentManager = new ContentManager();

    LeakCanary.install(this);
```

Kuva 5. LeakCanary täytyy myös asentaa Application –tiedoston onCreate()-metodissa. Korostettu rivi on lisättävä koodiin.



Kuva 6. Eclipse MATin avulla löydetty objekti, joka vie eniten muistia heap dumpissa. Kuva otettu 16.02.2016

Picassossa ja Glidessa on monia pieniä eroja, jotka pitää ottaa huomioon vaihdossa. Kun kirjastot importoidaan, Picasso vaatii vain yhden rivin, kun taas Glide vaatii kaksi riviä, transformointi-lisäkirjaston takia (Kuva 7). Toisaalta Picasso vaatii alustusta Application-tiedostossa, jota kutsutaan, kun sovellus avataan, ennen muita sovelluksen objekteja. Myös transformointi toimii eri lailla, Picassossa fitin, centerCropin ja transformoinnin järjestyksellä ei ole väliä, kun taas Glidessä on.

```
// compile 'com.squareup.picasso:picasso:2.5.2'
compile 'com.github.bumptech.glide:glide:3.5.2'
compile 'jp.wasabeef:glide-transformations:1.2.1'
```

Kuva 7. Picasson ja Gliden alustus build.gradlen riippuvaisuuksissa.

```
Glide.with(MainActivity.this)
        .load(result.getImage().getUrl(Image.MEDIUM))
        .centerCrop()
        .fitCenter()
        .bitmapTransform(circle)
        .into(meImage);
/*
Picasso..with(MainActivity.this)
        .load(result.getImage().getUrl(Image.MEDIUM))
        .centerCrop()
        .fit()
        .transform(circle)
        .into(meImage);
*/
```

Kuva 8. Picasson ja Gliden kutsuminen ja transformointi

Keskiviikko 03.02.2016

Keskiviikkoisin meillä on Scrumin mukainen viikoittainen palaveri, jossa kerromme mitä töitä olemme saaneet aikaan ja mitä pyrimme saamaan seuraavan viikon aikana tehtyä. Edellisenä viikonloppuna kävimme uuden fullstackimme – Juuso Koskenseppä - ja freelancerimme – Ville Valta - kanssa läpi erinäisiä bugeja sovelluksessa. Saimme selvitettyä miksi jotkin laulut eivät lataudu; sovellus latasi vain oletus sävelkorkeusvariaation laulusta ja kun sävelkorkeutta haluttiin muuttaa, laulutiedoston hakeminen ei onnistunut. Oikean sävelkorkeustiedoston lataaminen implementoitiin myös GameOverActivityyn ja local performance – tiedostoon. Latasimme myös viikonloppuna Android Studion version 2.0 Preview 9 että pääsimme testaamaan langatonta APK:n asennusta ja debuggausta. Suunnittelimme että julkaisisimme uuden alpha version sovelluksesta perjantaina ja julkaisisimme sen PlayStoreen seuraavana maanantaina. Keskustelimme freelancerin kanssa siitä olisiko parempi vaihtaa kolmannen osapuolen kuvanlatauskirjastoa Facebookin luomaan Frescoon vai Googlen suosittelemaan Glideen. Päädyimme Glideen, koska Fresco vaatisi, että jokainen ImageView olisi vaihdettu Frescon omaan DraweeViewhun. Halusin, että fullstackimme olisi tutkinut viime sprintiltä

jäänyttä pelinäkymän uutta designia, koska oli laulujen latausongelman takia nyt päässyt sisälle pelinäkymäämme, kun taas itse olisin keskittynyt Gliden implementointiin. Sovimme että Gliden jälkeen yrittäisin päästä tekemään Playlist-näkymiä uusien designien mukaisiksi. Android-puoli on nyt viikon myöhässä roadmapin aikataulusta, mutta koemme että sovelluksen vakaus menee uusien featureiden edelle.

Perjantai 5.2.2016

Asetin perjantain tavoitteeksi Gliden debuggauksen ja uuden alpha-version julkaisemisen Google Play kaupassa. Kun tulin töihin aamulla, huomasimme että uusin versio Android Studiosta oli julkaistu – Android Studio 2.0 Beta 2. Vaihdoimme siis tähän versioon, jonka jälkeen pyrimme tutkimaan sen hetkistä julkaistavaa APK:tamme. Odottamattomasti, release-versio ei suostunut valmistautua ajettavaksi.

Gradlen virheilmoitukset ilmaisivat, että keystorea olisi jotenkin sormeiltu, vaikka siihen ei oltu koskettu. Jos APK ei koostu ajettavaksi, emme saa sovellusta asennettua laitteeseen. Kokeilimme luoda uuden keystoren, mutta uudelleengeneroitu keystoremme oli täysin tyhjä. Yritimme lisätä keystorea muista versionhallintahaaroista, saadaksemme aikaisemman virheilmoituksen. Yritimme myös generoida APK:n komentorivin kautta, mutta tämä ei vaikuttanut ollenkaan asiaan. Keskustelufoorumeilta saimme idean, että gradle pistäisi vaihtaa uuteen stabiiliin versioon. Vaikka tämä ei toiminut, saimme idean kokeilla julkaistavan APK:n koostuttamista ajettavaksi stabiilissa Android Studio versiossa – 1.5.1 – ja julkaisuversio koostui ajettavaksi onnistuneesti.

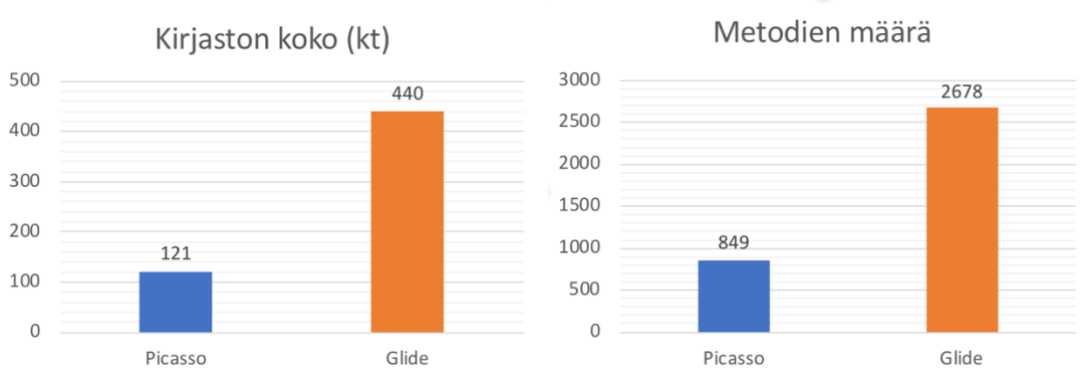
Valitettavasti muiden töiden teko ei perjantaina onnistunut, koska Runebergin päivän kunniaksi kävin hakemassa toimistolle Runebergin torttuja ja laskiaispullia lähimmästä leipomosta. Olen pannut merkille, että startupeissa ryhmäytyminen on hyvin tärkeää. Töitä tehdään tiiviissä tiimissä ja yritetään myös rentoutua yhdessä työviikon jälkeen. Singalla ryhmäydymme yleensä perjantai-iltaisin, istumalla alas ja vaihtamalla kuulumisia keskenämme. Koska työntekijöitä on verrattain vähän, kaikkien työpanosta tarvitaan, ja iloinen työntekijä on kaikista tuotteliain (Zelenski, 2008).

Viikkoanalyysi

Glide ja Picasso ovat kaikista suosituimmat kolmannen osapuolen kuvanlatauskirjastot Android kehittäjien kesken (Eklund, 2016). Molemmat kirjastot tarjoavat monia nopeita ja optimoituja ominaisuuksia. Pintaraapaisulla molemmat ovat hyvin samankaltaisia, mutta

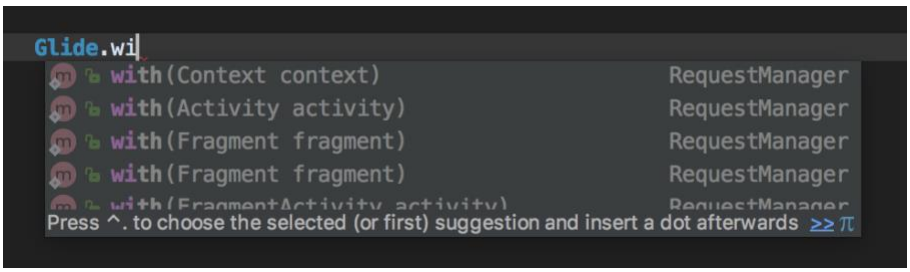
kun asiaa tutkii tarkemmin ne käsittelevät kuvien lataamista, välimuistia ja muistista lataamista eritavoin.

Kun vertailee Gliden versiota 3.5.2 ja Picasson versiota 2.5.2 toisiinsa voi huomata, että Gliden .jar-tiedoston koko on noin 3,5 kertaa suurempi kuin Picasson (Kuvio 1). Metodeja Picassolla on 849, kun taas Glidellä metodeja on 2678 (Kuvio 2).



Kuvio 1 ja 2. Picasson ja Gliden .jar-tiedosto koko kilotavuina ja metodien lukumäärät

Molemmilla kirjastoilla on samanlainen syntaksi, kun ladataan kuvia URLista ja näytetään niitä ImageViewtä – Androidin sisäistä kuvaelementtiä – apuna käyttäen. Molemmat kirjastot tukevat häivytysanimaatiota ja keskitettyä rajausta. Kirjastoilla voi myös lisätä sijaiskuvia, kun kuvaa ladataan tai jos kuva ei lataudukaan. Glide on myös luotu tukemaan aktiviteettien ja fragmenttien elinkaarta. Glide.with():lle voi syöttää siis kontekstin sijasta myös aktiviteetin tai fragmentin (Kuva 11) ja kirjasto integroituu saumattomasti aktiviteetin elinkaaren pyyntöihin kuten onPause() tai onResume().

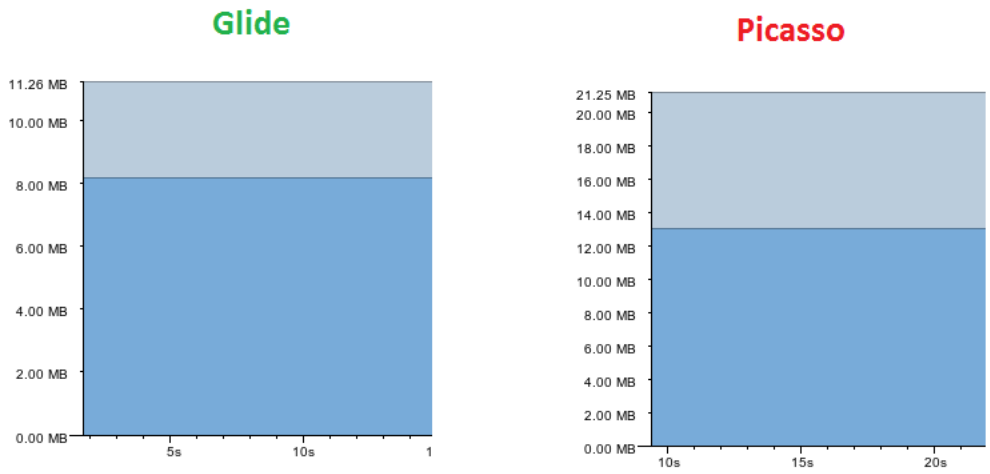


Kuva 11. Ehdotukset mitä Gliden metodille with voi syöttää.

Molemmat kirjastot lataavat kuvan URL:ista ja tallentavat levyn välimuistiin, mutta tavassa, jolla kirjastot tallettavat kuvat välimuistiin, on eroavaisuuksia. Picasso lataa kuvan ja pitää sen tallessa täysikokoisena välimuistissa (Peitek, 2015). Kun kutsumme kuvaa saamme sen täysikokoisena ja kuvan kokoa muutetaan reaaliaikaisesti sopimaan ImageViewiin. Glide taas lataa kuvan, muuttaa kuvakokoa ImageViewn mukaan ja

tallentaa tämän pienemmän kuvan levyn välimuistiin (Peitek, 2015). Eli jos lataat saman kuvan kahteen eri kokoiseen ImageViewiin, muistiin tallentuu kaksi versiota samasta kuvasta eri resoluutioilla. Tämä lisää välimuistin kokoa, mutta koska kuvat ovat pienempiä, ne latautuvat nopeammin välimuistista.

Glide käyttää kuvissa oletuksena RGB_565 konfiguraatiota, kun taas Picasso ARGB_8888 konfiguraatiota. ARGB_8888 antaa parhaan mahdollisen lopputuloksen, kun taas RGB_565 taas vie vähemmän tilaa. Joten jos haluaa tehdä reilun vertailun pitää GlideModulea muokata käyttämään ARGB_8888a. Jos Glideä ja Picassoa käyttää lataamaan kuvan voi huomata, että Glide vie vähemmän tilaa kuin Picasso (Kuva 12). Gliden tapa tallentaa kuva pienemmällä resoluutiolla on selkeästi tehokkaampi, mikä tulee aiheuttamaan vähemmän OOM-virheitä.



Kuva 12. Gliden ja Picasson muistinkäyttö.

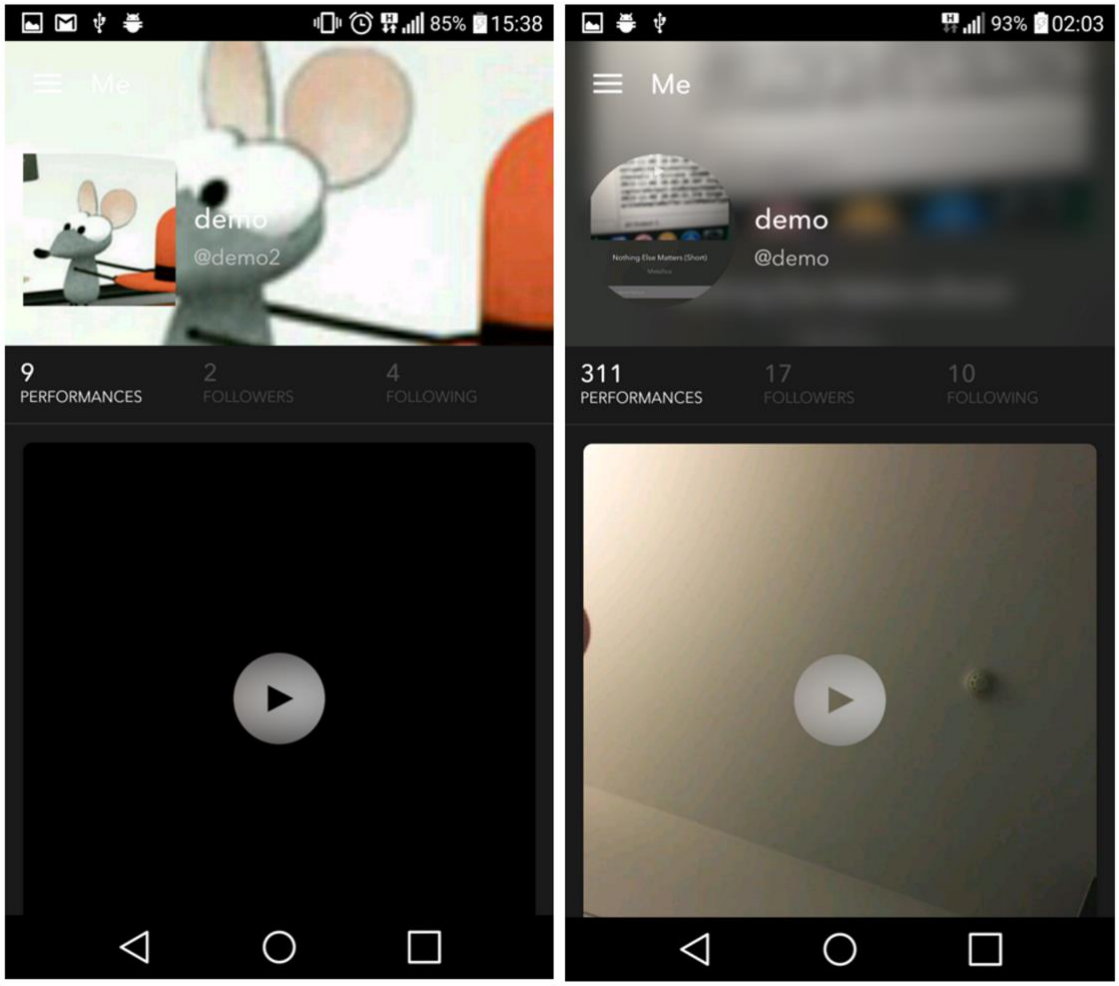
Picasso lataa kuvat aluksi nopeammin ja paremmalla resoluutiolla, kun taas Glide lataa välimuistista kuvat nopeammin ja päätyy oman sovelluksemme kannalta viemään loppujen lopuksi vähemmän tilaa. Koska halusimme kirjaston vaihdolla vähentää OOM virhetilanteita, oli Glide helppo valita.

3.2 Seurantaviikko 02

Sunnuntai 07.02.2016 - Maanantai 08.02.2016

Huomasimme sunnuntai-iltana, että Glidellä tehdyt transformoinnit eivät jostain syystä suoriutuneet. Tutkittuamme full stackimme kanssa pitkään mahdollista syytä huomasimme, että Gliden komentojen järjestys ei ollut ollut oikein kaikissa tiedostoissa.

Käytin aamuyöni käyden läpi kaikkia Glide-kutsuja. Jos transformointi yritetään tehdä ennen fitCenter- ja centerCrop –komentoja, transformointi ei tapahdu (Kuva 13).
Kun aamulla kävin läpi loppuja Gliden vaihtoon kuuluvia tehtäviä, huomasin, että Glidellä kohdentaminen oli myös erilaista Picassoon verrattuna (Kuva 14). Kun Picassolla haluat kohdentaa kuvan esimerkiksi esikatselukuvaa tehdessä asiaankuuluvaan view-elementiin, käytät fraasia ’this’. Glidellä on kuitenkin mahdollisuus tehdä enemmänkin kohdennetulle elementille. Glideen vaihto oli tässä vaiheessa valmis.



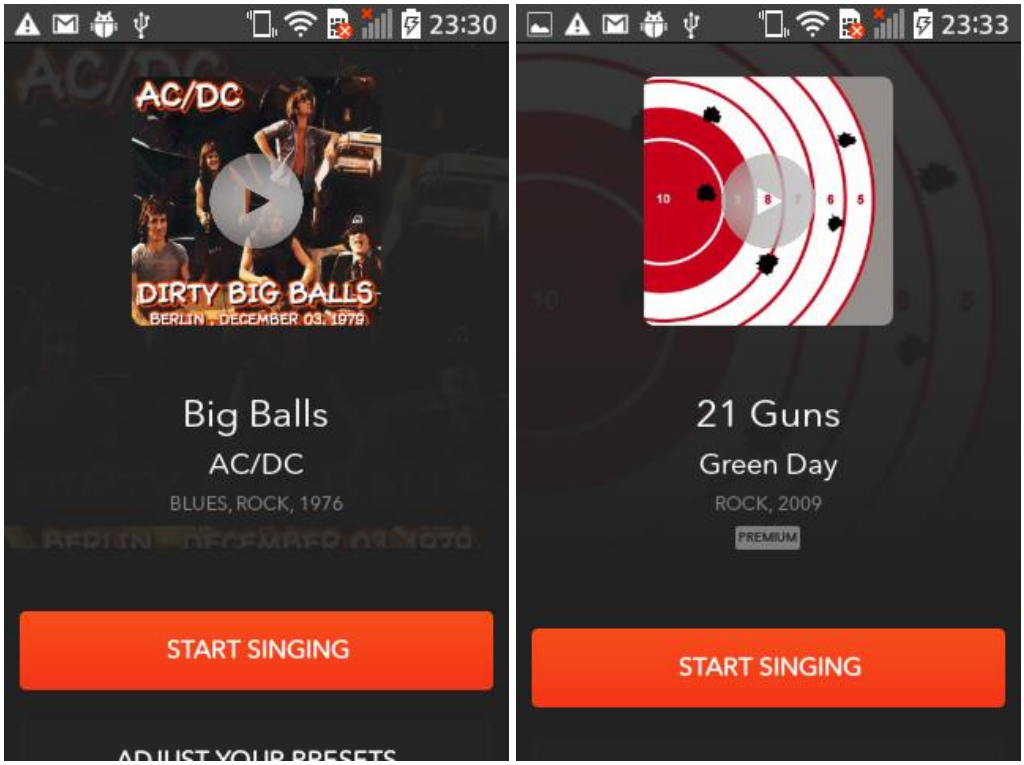
Kuva 13. Ennen ja jälkeen Glide-korjausten.

```
/*
Picasso.with(context).load(Url)
 .placeholder(R.drawable.placeholder_song_performance)
 .into(this);
*/
Glide.with(context)
        .load(Url)
        .asBitmap()
        .placeholder(R.drawable.placeholder_song_performance)
        .into(new ViewTarget<SingaVideoView, Bitmap>(SingaVideoView.this) {
            @Override
            public void onResourceReady(Bitmap resource, GlideAnimation anim) {
                this.view.setThumbnailImage(resource);
            }
        });
```

Kuva 14. Gliden ja Picasson kohdennusero.

Tiistai 09.02.2016

Tiistaina tarkkaavainen full stackimme, Juuso Koskenseppä, huomasi, että laulunäkymissä, joissa ei ollut premium-lappua, oli premium-lapun kokoinen aukko näkymässä (Kuvat 15 ja 16). Ajattelin ensin ongelman johtuvan siitä, että olen vahingossa asettanut premium-lapun visibilityn – eli näkyvyyden - invisibleksi gonen sijaan. Invisible muuntaa näkymän näkymättömäksi, gone poistaa näkymän käyttöliittymän ulkoasusta ja piirtää näkymän sen mukaan. Näin en ollut kuitenkaan tehnyt.



Kuva 15 ja 16. Normaali laulunäkymä ja premium laulunäkymä LG L40 –puhelimen ruudulta ennen korjausliikkeitä.

Aukko näkymässä oli kuitenkin premium-lapun kokoinen, josta päättelin, että TextViewlle ei voi asettaa toimivasti visibilityä vanhemmissa puhelimissa. Vaihtoehdoiksi jäikin siis luoda uusi LinearLayout elementti, jonka sisälle sisällyttää premium-lappu, tai vaihtoehtoisesti muuttaa teksti ja tekstin tausta premium-ominaisuuden mukaan (Kuva 17). Valitsin jälkimmäisen vaihtoehdon, koska xml:n kuormittaminen periaatteessa ylimääräisillä elementeillä hidastaa näkymien piirtoa (McAnlis, 2015). Tämä ratkaisi ongelman.

```
premiumLabel.setText(song.getIsPremium() ? "PREMIUM" : " ");
premiumLabel.setBackgroundResource(song.getIsPremium() ? R.drawable.bg_premium : R.drawable.transparent_button);
// premiumLabel.setVisibility(song.getIsPremium() ? View.VISIBLE : View.GONE);
```

Kuva 17. Ratkaisu TextViewn näkyvyysongelmaan.

Torstai 11.02.2016

Koska en päässyt keskiviikkona weekly meetiin, tein uuden version Android roadmapista torstaina toimistolla (Taulukko 2). Julkaisin myös Glidellä toimivan version sovelluksestamme Google Play Storeen. Jaoin työtehtävät seuraavalle viikolle full stackimme Juuso Koskensepän kanssa siten, että hän hoitaisi ilmoituksien ryhmittämisen ja minä hoitaisin playlistit eli laululistat niin pitkälle kuin saisin. Olin suunnitellut, että koska Juuso oli ennenkin tehnyt laululistan tyylisiä ratkaisuja olisi hyvä oman oppimiseni kannalta, jos saisin tehdä laululistaominaisuutta hänen kanssaan parikoodauksena, kunhan hän olisi valmis ilmoitusten vaatimien muutosten kanssa.

vko	**4 - 6**	**7 - 8**	**9 - 10**
Android	Glide optimointi	Ilmoitukset Playlist	Playlist Lokalisaatio

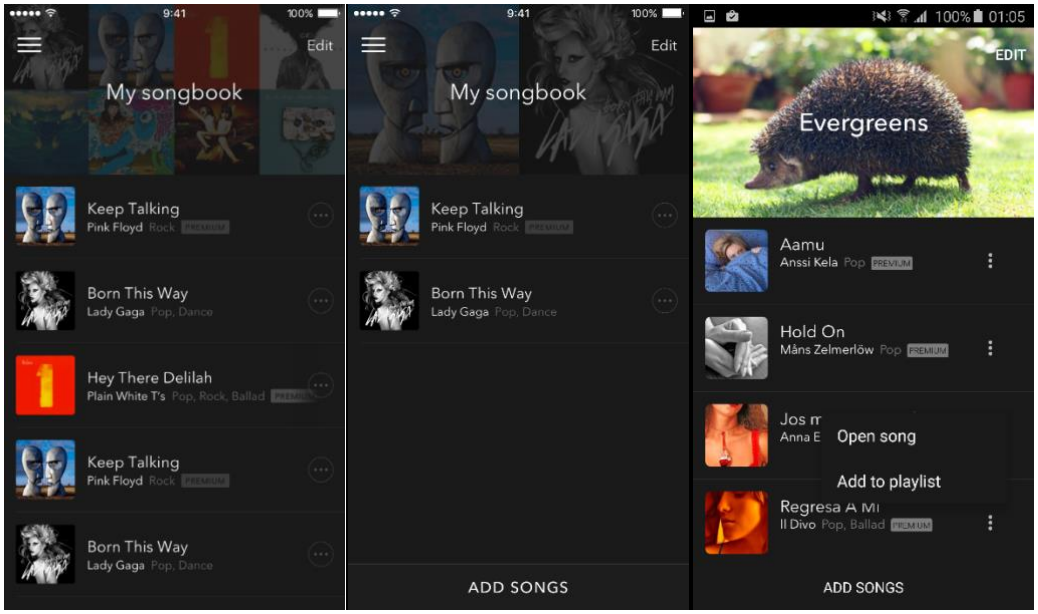
Taulukko 2. Singan roadmap 11.02.2016

Tein illalla myös julkaisun Play-kauppaan, koska Juuso oli saanut optimoitua Gliden ja parannellut NavigationDrawerin toiminnallisuutta.

Perjantai 12.02.2016

Pyrin aloittamaan käyttäjien playlistien mahdollistavat näkymät. Lähdin pienistä ja helpoista elementeistä liikkeelle. Että laulun voisi lisätä playlistiin on lista-alkiossa oltava

ActionMenu, josta voi valita avataanko laulunäkymä vai lisätäänkö laulu playlistiin. Toiminnallisuutta en vielä sisällyttänyt ActionMenuun. Kuten aikaisemminkin Androidin kanssa, päätin että käytämme Androidin omaa ActionMenu-ikonia, joka tulee olemaan loppukäyttäjälle selvempi ja yhtenäisempi muiden Android-kokemusten kanssa. Vaikka kustomoitu ikoni olisi omalla tavallaan hyvinkin hieno, käyttökokemuksen yhtenäisyys on omasta mielestäni syy siihen, että Android-sovellusten kehittämiseen tarjotaan niin paljon valmista sisältöä. Olemme käyttäneet aikaisemmin ActionMenua myös RecordingViewssämme, ja voimme siten sanoa, että tämä tuo käyttäjälle yhtenäisemmän käyttökokemuksen.



Kuva 18, 19 ja 20. Playlistin designit mistä lähdin rakentamaan toteutusta, ja ActionMenu, Edit-button ja SnackBar laululistassa.

Että käyttäjä pääsisi katsomaan omia laululistojaan, lisäsin NavigationDraweriin oman PlaylistsFragmentin, joka vie käyttäjän laululistojen listaukseen. Toistaiseksi en tehnyt vielä käyttäjälle omien laululistojen listausta, koska en ole vielä tehnyt laululistojen luontiakaan. Lisäsin PlaylistFragmentin toolbariin edit-buttonin, että playlistiä voi muokata. Lisäsin myös laulujen lisäystä varten SnackBarin alalaitaan kummittelemaan.

Viikkoanalyysi

Pienyrityksessä on todella tärkeää, että jokainen tekee oman osansa työtaakasta. Tästä syystä on hyvä pitää työntekijät onnellisina ja pitää yllä tiivistä ryhmäntunnetta (Van Der Vengt, 2005). Singalla ryhmäytymisen eteen joka perjantai vietämme töiden jälkeen aikaa yhdessä puhumalla edellisen viikon onnistumisista ja epäonnistumisista vapaasti.

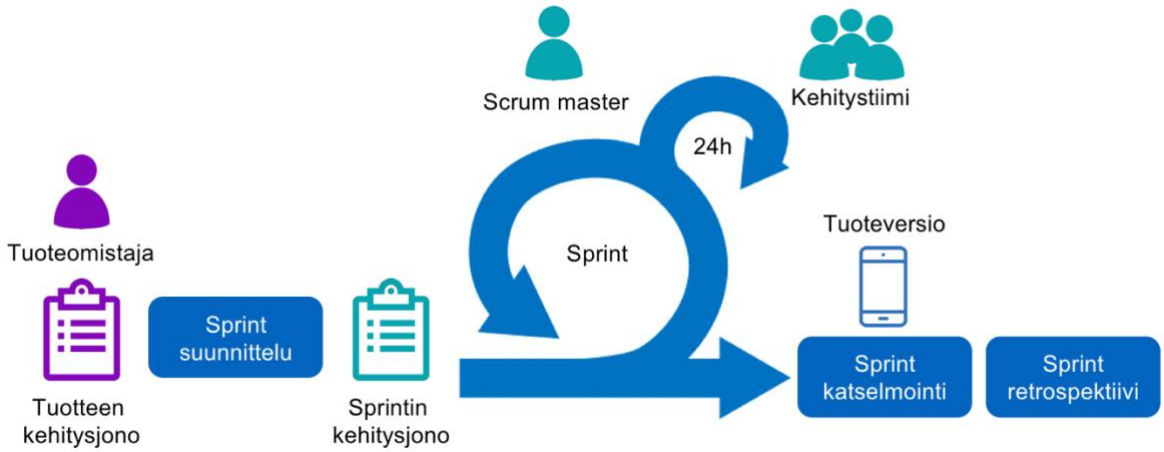
Työntekijät saavat myös tällöin kiitosta hyvin tehdyistä töistä. Perjantaina puhuimme työskentelymallimme muuttamisesta Scrumista kanbaniksi.

Ketterässä sovelluskehityksessä on monia viitekehyksiä, joita voidaan hyödyntää projektitöissä. Tällä hetkellä käytämme Scrumia, joka perustuu palauteluuppiin, jonka perusteella työntekoa ja työtehtäviä pystytään iteroimaan (Sutherland, 2018). Vaikka alun perin Hirotaka Takeuchi ja Ikujiro Nonaka kirjoittivat viitekehyksen ideasta jo vuonna 1989 (Takeuchi, 1986), Scrumin isinä pidetään laajalti Jeff Sutherlandia ja Ken Schwaberia, jotka kirjoittivat vuonna 1995 tutkielman (Sutherland, 1995) sekä monia kirjoja aiheesta. Viitekehykseen sisältyy erilaisia rooleja, tuotoksia ja tapahtumia. Scrumia tehdään sprinteissä, jotka ovat maksimissaan 4 viikon mittaisia ajanjaksoja, jolloin uutta tuoteversiota kehitetään.

Roolit Scrumissa ovat tuoteomistaja, scrum master ja kehitystiimi (Kuva 21). Yhteen niputettuina nämä muodostavat scrum tiimin. Tuoteomistaja on yleensä asiakkaan edustaja, jolla on valtuudet ja tieto, jonka mukaan priorisoida projekti. Hän on suorassa yhteydessä muuhun scrum tiimiin ja muihin sidosryhmiin. Singalla tuoteomistajana toimii Singan toimitusjohtaja Atte Hujanen, joka on yhteydessä kehitystiimiin. Hänellä on visio mitä eri ominaisuuksia hän haluaa sovellukseen ja tuoteomistajana hänen vastuullaan on tuotteen kehitysjonon ylläpito ja priorisointi. Tuotteen kehitysjono on Scrumin mukainen tuotos, jossa ovat kaikki tuotteen vaatimukset pienistä virheenkorjauksista suuriin kokonaisuuksiin. Tuotteen kehittäjät eli designer ja ohjelmistokehittäjät ovat osa tuotteen kehitystiimiä, jonka odotetaan luovan valmiita tuoteversioita. Valmis tässä kohtaa tarkoittaa sovittua valmiin määritelmää. Kehitystiimi on myös vastuussa tuotteen laadusta. Scrum master voi olla joko kehitystiimin ulkopuolinen tai sisäinen jäsen. Scrum masterin vastuulla on fasilitoida scrumia – eli pitää huolta scrumin mukaisista tapaamisista ja tuotoksista – sekä poistaa esteitä, joita projektityön eteen ilmaantuu. Singalla scrum masterina toimii selainversion pääkehittäjä Jani Nyman, joka pitää kyllä huolen scrumin fasilitoinnista, mutta joka ei työtehtäviensä laajuuden takia ehdi esteiden poistamiseen. Tämä puolestaan aiheutti suurimman osan ongelmista Singalla Scrumin kanssa. Miten poistamme esteitä, kun ne tulevat esiin?

Kuva 21. Scrum tiimin roolit visuaalisesti demonstroituna

Scrumin tuotoksia taas ovat tuoteomistajan vastuulla oleva tuotteenkehitysjono, sprintin kehitysjono ja tuoteversio. Tuotteen kehitysjonossa ovat kaikki tehtävät mitä tuotteeseen halutaan kehittää. Tästä tehtävälistasta ja sen priorisoinnista vastaa tuoteomistaja. Sprintin kehitysjonossa taas ovat kaikki sprintille valitut tehtävät. Tästä tehtävälistasta taas kehitystiimi on vastuussa. Sprintin jälkeen kehitystiimi on luonut tuoteversion, joka on toimiva ja mahdollisesti julkaistava versio tuotteesta. Tuotteen kehitysjono Singalla löytyy Trellosta, joka on selainpohjainen projektinhallintatyökalu. Sprintille valitut tehtävät yleensä kirjoitettiin ylös valkotaululle roadmapiin. Tehtävien haasteellisuuden arvioiminen on kuitenkin osoittautunut jokaiselle kehitystiimin jäsenelle vaikeaksi ja hyvin usein sprintin jälkeiset tuoteversiot eivät olleet valmiita. Suurin osa sprinteistä on ollut tästä syystä epäonnistuneita, mikä on aiheuttanut motivaation laskemista kehitystiimin sisällä.



Kuva 22. Scrumin visualisaatio

Scrumin keskeisiin tapahtumiin kuuluvat sprint, sprintin suunnittelupalaveri, päiväpalaveri, sprintin katselmointi ja sprintin retrospektiivi. Sprintti alkaa sprintin suunnittelupalaverilla,

johon koko scrum tiimi osallistuu, ja jossa luodaan sprintin kehitysjono käymällä tuotteen kehitysjonoa läpi ja valitsemalla sieltä tärkeimmät tehtävät. Ennen sprintin loppua näitä valittuja tehtäviä sprintin kehitysjonosta pyritään kehittämään. Päivittäiseen scrumiin kuuluu noin 15 minuutin päiväpalaveri, johon koko kehitystiimi osallistuu. Koska päiväpalaverissa jokainen kertoo mitä on saanut aikaan ja mitä päivälle on suunnitellut tekevänsä, optimoi päiväpalaverin pitäminen tiimin yhteistyötä ja suorituskykyä tuomalla herkästi esiin esteitä työnteolle. Tällöin kun päivittäin ilmaistaan ongelmia pystyvät muut työntekijät auttamaan ongelmien kanssa. Singalla päiväpalaverit korvattiin daily-kanavalla Slackissa, joka on yrityksien sisäisiin tarpeisiin suunniteltu pikaviestintäsovellus. Koska kaikki kehitystiimin jäsenet eivät voineet olla paikalla joka päivä, on etä-daily paljon käytännöllisempi ratkaisu Singalla. Lisää läpinäkyvyyttä pyrimme myös tuomaan kehitysprosessiin pitämällä kerran viikossa viikkopalavereja, joissa edellisen viikon aikaansaannoksista keskustellaan ja seuraavan viikon työtehtäviä pohditaan. 4 viikon työrupeaman lopuksi pidämme sprint katselmoinnin, jossa käymme läpi mitä saimme sprintin aikana valmiiksi ja päivitämme tuotteidemme kehitysjonon ja keskustelemme, miten voisimme parantaa tuotettamme. Sprintin retrospektiivit pidetään yleensä sprintin päätteeksi. Retroissa pyritään parantamaan sprintin prosessia ja mietitään mitä konkreettisesti voidaan tehdä paremmin seuraavassa sprintissä. Sprint retrospektiivejä Singalla pidimme noin neljän sprintin välein. Retroissa ehdotetut muutokset kuitenkaan eivät koskaan päätyneet toteutumaan, koska kukaan ei ottanut muutoksien ajamista eteenpäin vastuulleen.

Ongelmana oli siis pääasiallisesti päätöksien loppuun saattaminen ja scrumin periaatteista kiinni pitäminen. Kellään ei ollut näennäisesti aikaa ryhtyä pitämään kiinni sovituista prosessimuutoksista eikä tämän tähden mitään ketään päätynyt ottamaan vastuuta prosessien parantamisesta. Myös työlle asetetut aikarajat stressasivat työntekijöitä ja aiheuttivat motivaationpuutetta. Mutta koska työprosessin kokonaisvaltainen muutos aiheuttaisi tässä kriittisessä kohtaa enemmän ongelmia kuin mitä tämä ratkaisisi päätimme olla muuttamatta työtapojamme.

3.3 Seurantaviikko 03

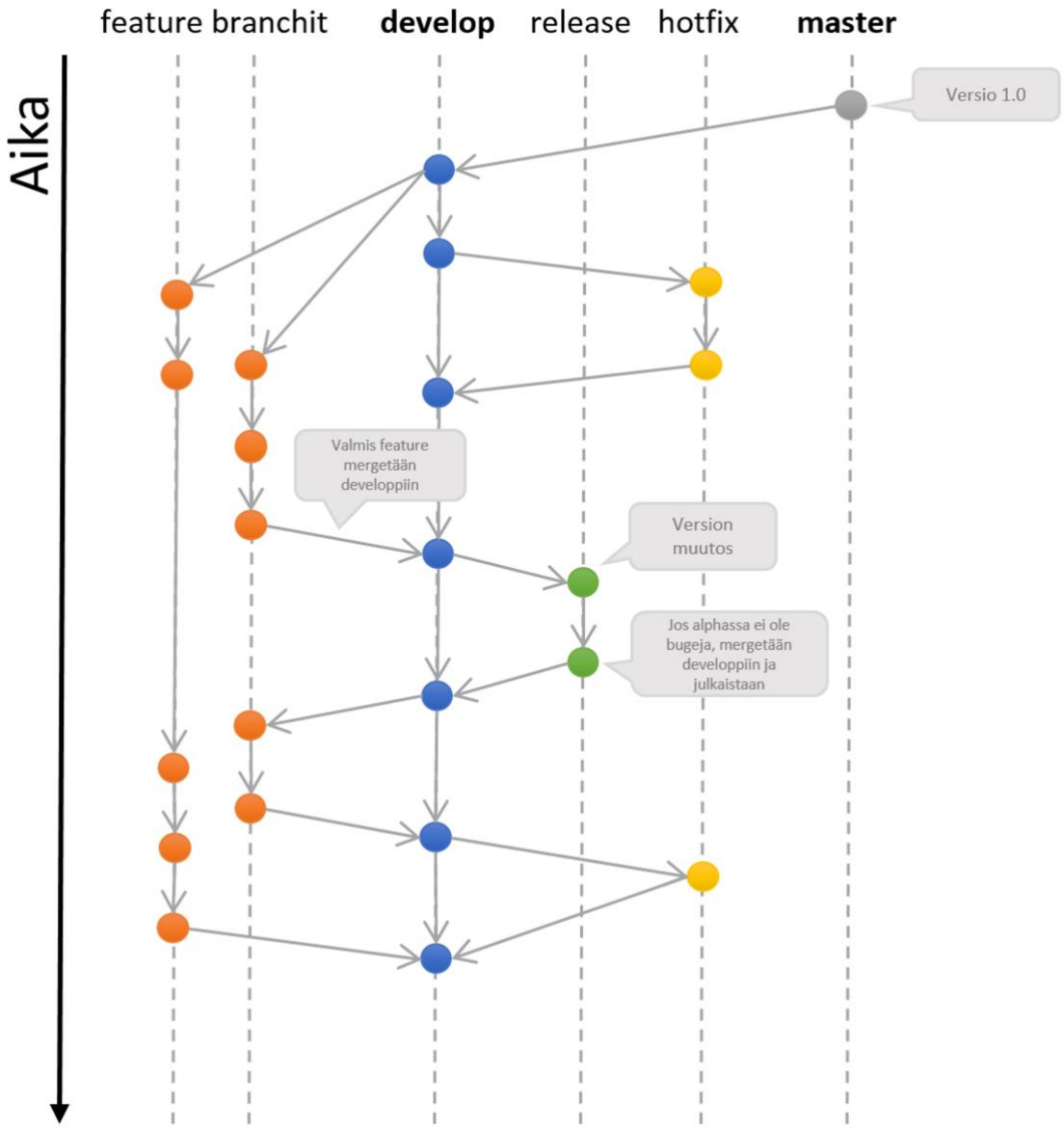
Maanantai 15.02.2016

Päätin käyttää maanantaini gitin selvittämiseen ja päivittämiseen, koska full stackimme Juuso Koskenseppä oli myös tekemässä töitä Android-sovelluksemme eteen ja olimme kohdanneet jo pari kyseenalaista merge conflictia. Koska en pariin kuukauteen ollut tehnyt yhteistyötä kenenkään kanssa, olin kehittänyt monia eri toiminnallisuuksia samassa

kehityshaarassa samanaikaisesti enkä ollut voinut keskeneräisyyksien takia mergetä valmiita toimintoja develop-haaraan.

Kun aloitin työt Singalla, minulle selitettiin, että master-haarassa pidetään vanhaa stabiilia koodia tallessa. Eli pääkehityshaarana toimisi develop, johon valmiita featureita ja korjauksia pushataan. Developia testataan ja siitä haarautuvat stabiilit release versiot, jotka menevät tuotantoon Googlen PlayStoreen (Kuva 23).

Olin kuitenkin alkanut tehdä kaikenlaisia featureita ja korjauksia feature/user-playlists haaraan omaa huolimattomuuttani. Piilotin playlisteihin tekemäni muutokset ja pushasin user-playlists-haaran developpiin ilman kovin suuria merge conflicteja. Lopuksi, tein uuden haaran lokalisaatiotoiminnallisuudelle.



Kuva 23. Esimerkki Singan git flowsta, havainnollistettuna.

Tiistai 16.02.2016

Tiistain vietin virheenkorjausten parissa. Full stackimme Juuso Koskenseppä oli tutkiessaan kaatumisia Crashalyticsin kautta huomannut, että suurin osa kaatumisista aiheutui yhdestä RuntimeExceptionistä (Kuva 24). Juuso arveli, että kyse on Toastista, joka ei ole UI-säikeen sisällä.

Kuva 24. Fabricin antama virheviesti.

En ollut koskaan itse saanut tätä virhettä aikaiseksi, joten en tiennyt miten testata oliko ongelma ratkaistu. Tein kuitenkin työtä käskettyä ja kävin läpi kaikki 92 Toastia, ja varmistin, että kaikki ovat UI-säikeen sisällä. Referenssejä löytyi yhteensä 16, jotka sopivat kuvaukseen. Mikään niistä ei ollut rivillä 271 tai 305, mutta siirsin ne silti (Kuva 25). Koska en saanut kyseistä virhettä aikaan millään testipuhelimista, julkaisin version ja sormet ristissä toivoin, että saisimme tietää, oliko virheilmoituksessa tästä kyse.

```
private void showError(final String message) {
    final Context c = this;
    this.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(c, message, Toast.LENGTH_SHORT).show();
        }
    });
}
```

Kuva 25. Näin ajamme Toastin UI-säikeellä.

Keskiviikko 17.02.2016

Weekly meetissä kävi ilmi, että koska Android-puoli oli päässyt aikatauluun kiinni, full stackimme Juuso Koskenseppä alkaisi tehdä iOS-, AppleTV- ja Pro- tuotteisiimme

playeriä. Olin aikaisemmin suunnitellut, että Juuson kanssa olisimme parikoodanneet kivasti playlistejä, mutta valitettavasti se ei ole mahdollista Juuson siirtyessä tärkeämpiin tehtäviin.

Juuso oli saanut tehtyä ilmoitusten ryhmittämisen loppuun asti, joten testasin ilmoituksia manuaalisesti ja gui-testaustyökalu Monkeyn avulla. Monkey lähettää kosketustapahtumia sovellukseen. Simppeleimmissä komennoissa on määriteltävä vain sovelluksen pakettinimi ja kosketuksien haluttu määrä (Kuva 26).

```
C:\Users\katri\AppData\Local\Android\sdk\platform-tools>adb shell monkey -p com.singa.client 500
    // Injection Failed
Events injected: 500
## Network stats: elapsed time=5258ms (0ms mobile, 0ms wifi, 5258ms not connected)
```

Kuva 26. Monkey testauksen tulos.

Menimme iltapäivällä ravitsemusliikkeeseen Apple TV -devaajamme Elias Mikkolan ja Product Managerimme Atte Valtosen keskustelemaan rekrytoinnista. Keskustelimme yhteisistä tuttavistamme ja keksimme heille mielekkäitä työtehtäviä Singalta. Kun vaihdoimme ravitsemusliikettä herra Valtosen kanssa, löysimme yhden yhteisistä tuttavistamme Antti Rummukaisen, jolle ehdotimme Singaa työharjoittelupaikaksi. Antti tulisi auttamaan web playerimme implementoinnissa, ja pisti siltä istumalta hakupaperit tulemaan.

Lauantai 20.02.2016

Lauantain teemana oli tutkia Androidin AV-synkronointiongelmaa. Freelancerimme Ville Valta oli löytänyt Superpowered-nimisen kolmannen osapuolen kirjaston, jota hän, full stackimme Juuso Koskensepän kanssa, alkoi implementoida.

Minulle oli aikaisemmin sanottu, että haluamme poistaa laululistauksestamme mainokset, koska ne aiheuttavat turhaa viivettä, eivätkä tuota tarpeeksi rahaa ollakseen aiheutuvan käyttäjäkokemuksen huonontumisen arvoisia. Tein tämän kommentoimalla adprovider-variantin käytön ListAdapteristamme. Ei sitä koskaan tiedä, jos haluamme alkaa mainostaa uudestaan.

Ville huomasi Playlists-näkymässä Gliden aiheuttaman transformointiongelman, joten aloin käydä kaikkia näkymiä läpi, ja korjaamaan mahdolliset Gliden järjestysvirheet.

CTOmme Tomi Pajunen oli päivän päätteeksi sitä mieltä, että emme tule käyttämään Superpoweredia sovelluksessamme, koska suljetun kirjaston implementointi kestäisi kauan, ja sen tuki todennäköisesti maksaisi valuuttaa.

Viikkoanalyysi

Monet mobiilisovellukset ovat riippuvaisia alhaisesta viiveestä äänitoiminnallisuuksissa. Trendinä näyttävimmät interaktiiviset ääntä käyttävät sovellukset eivät ole siirtyneet App Storen puolelta Play Storeen. Tämä johtuu Androidin 10 millisekunnin ongelmasta, joka on vaikea tekninen haaste, jolla on suuret seuraamukset. Ongelman nimi tulee ihmiskorvan havaitsemasta viiveestä puheessa. Kun viive äänen kuulemiseen on yli 10 millisekuntia, ihminen voi havaita ongelman (Nokia Siemens Networks, 2009). Koska heikentynyt äänenlaatu varmasti aiheuttaisi negatiivista palautetta ja tekisi lommon muuten puhtaaseen brandiin, moni sovelluskehittäjä on päättänyt jättää Androidin omaan rauhaansa. Tässä kontekstissa kokonaisviiveellä tarkoitetaan sitä aikaa mikä kuluu, kun mobiililaite saa äänisyötteen, käy läpi tarvittavan prosessoinnin ja tuottaa sitten tämän äänen kuultavaksi.

10 millisekunnin ongelma aiheuttaa suuria suruja monella alueella. Ääniefekti ja virtuaalisia instrumentti sovelluksia käyttävät esiintyjät olisivat puolikkaan biitin myöhässä muista. Pelien ääniefektit tulevat viiveellä ja ääni kuulostaa siksi häiritsevän erilliseltä ulkoasuun verrattuna. Singassa ongelma esiintyi ikävänä viiveenä, kun laulaja yritti kuunnella itseään kuulokkeilla samaan aikaan laulaessaan. Myös videoissa ääni kuului häiritsevällä latenssilla.

Äänen täytyy kulkea monen tason läpi Android-laitteella (Vlaskovits, 2015) (Kuvio 3). Ensiksi on monia eri analogisia komponentteja, kuten sisäänrakennetun mikrofonin esivahvistin, jotka eivät aiheuta yli millisekunnin viivettä. Äänisiru mittaa sisään tulevan äänijonon ennalta määrätyin aikavälein ja konvertoi nämä mittatulokset numeroksi. Tätä aikaväliä kutsutaan sämpläystahdiksi ja se mitataan hertseissä. Esimerkiksi jos sämpläystahti on 48 000 hertsiä tarkoittaa tämä, että ääninäytettä mitataan 48 000 kertaa sekunnissa.

Kuvio 3. Ääninäytteen kulkureitti Android-laitteen sisällä.

Konversio analogisesta digitaaliseksi sisältää yleensä ylisämpläys filtterin, joten tämän tason ajatellaan yleisesti vievän noin yhden millisekunnin. Tämän tason jälkeen digitaalinen ääni kulkeutuu eteenpäin lohkottuina kokonaisuuksina, joita kutsutaan yleisesti puskureiksi.

Äänisirulla on monta tehtävää, se hoitaa muutokset analogisesta digitaaliseksi ja toisin päin, yhdistelee ja vaihtelee eri tulosteiden ja syötteiden välillä, säätelee äänenvoimakkuutta ja niin edelleen. Äänisiru myös ryhmittää erilliset ääninäytteet puskureiksi ja siirtää ne käyttöjärjestelmän käytettäväksi. Äänisiru on yhdistettynä suorittimeen USBn, PCIn tai muunlaisen väylän kautta. Jokaisella väylällä on oma siirtoviiveensä riippuen väylän sisäisten puskurien ko'oista ja määristä. Viive tällä tasolla vaihtelee yhden ja kuuden millisekunnin välillä, riippuen siitä käyttääkö äänisiru järjestelmän sisäistä väylää vai USB äänikorttia konservatiivisilla asetuksilla.

Ääniajuri ottaa vastaan tulevan audion rengaspuskuriin väyläpuskurin kokoisina palasina käyttäen sirun natiivia sämpläystahtia, joka yleensä on edellä mainittu 48 000 hertsiä. Rengaspuskuri on hyvin tärkeä, koska se tasoittaa väyläsiirron karkeutta ja tavallaan yhdistää väyläsiirron puskurikoon käyttöjärjestelmän äänipinon puskurikokoon. Rengaspuskurin datan käyttö tapahtuu käyttöjärjestelmän äänipinossa, joka luontaisesti

lisää viivettä. Koska Android pohjautuu Linux-käyttöjärjestelmään, monet Android laitteet käyttävät suosituinta Linuxin äänirajapintaa - ALSAa. Rengaspuskuri jaetaan moneen periodiin. Esimerkiksi, periodien koko on 480 ääninäytettä, periodien määrä on kaksi, joten rengaspuskurin koko on 480 kertaa kaksi eli 960 ääninäytettä. Sillä aikaa kun yksi periodi vastaanottaa äänisyöteen, äänipino lukee ja prosessoi toista pinoa. Viive vie tässä esimerkkitapauksessa yhden periodin aikana, 480 ääninäytteen prosessointiin, 48 000 hertsin sämpläystahdilla, 10 millisekuntia. Tämä taso siis lisää käytettyjen periodien verran viivettä, ja periodeja saattaa jopa olla kaksi tai enemmän.

HAL eli laitteiston abstrahointi taso toimii Android media serverin ja Linuxin ääniajurin välikätenä. Tämän tason implementointi on täysin kiinni puhelimen valmistajasta, koska Android alustana antaa laitteiden toimittajille vapaat kädet tässä kohtaa koodin suhteen. Kommunikaatio media serverin kanssa käyttää ennalta määrättyä rakennetta; media serveri lataa laitteen HAL:n ja pyytää lupaa luoda syöte ja tuloste jonot, jotka voidaan luoda vapaavalintaisilla muuttujilla, kuten esimerkiksi halutulla sämpläys tahdilla tai puskurin koolla. Hyvän HAL implementaation ei pitäisi luoda lisää viivettä.

Androidin media serveri koostuu kahdesta palvelusta – AudioPolicystä ja AudioFlingeristä. AudioPolicy-palvelu käsittelee audio sessioita ja lupia, kuten mikrofonin käyttölupaa. AudioFlinger puolestaan käsittelee digitaalista äänijonoa. AudioFlinger luo RecordThread-säikeen, joka toimii välikätenä sovelluksen ja ääniajurin välillä. Tämä säie hankkii syötteessä seuraavan äänipuskurin rengaspuskurilta HAL:ia apunaan käyttäen, muuttaa puskurin sämpläys tahtia, jos sovellus sitä vaatii ja tekee lisäpuskurointia, jos sovellus vastaanottaa eri puskuri kokoa kuin mitä laitteisto sisäisesti tarjoaa. Käyttämällä Androidin natiivia koodia hyväkseen pystyy sovellus luomaan äänipuskurijonon laitteiston sämpläystahdilla ja periodi koolla AudioFlingerin avulla, jos käyttöjärjestelmä sen sallii. Viive tällä tasolla on parhaassa tapauksessa yhden periodin mittainen.

Androidin pääprosessien välisen kommunikointijärjestelmän jaettua muistia käytetään välittämään äänipuskureita AudioFlingerin ja sovelluksen välillä. Tämän jälkeen AudioRecord-kirjasto implementoi sovellukseen äänisyötteen (Android Developers Documentation, 2018). Tämä kirjasto ajaa säännöllisesti säikeen, joka hakee AudioFlingeriltä uuden puskurin. Jos sovelluksen kehittäjä ei päätä lisätä buffereita, tämän tason ei pitäisi aiheuttaa ylimääräistä viivettä.

Koska syöte ja tuloste säikeet eivät ole samat, sovelluksen on implementoitava rengaspuskuri säikeiden välille. Puskurin koko on vähintäänkin kahden periodia – yksi

äänisyötteelle ja yksi äänitulosteelle.

Tästä alkaakin äänitulosteen matka, joka on hyvin samanlainen kuin äänisyötteen. Yksi hiuksen hienoista eroista on se, että RecordThread-säikeen sijasta syötteen välikätenä sovelluksesta ääniajuriin toimii PlaybackThread. Joten jos Android sovellus haluaa toistaa käyttäjälle tämän omaa laulua, tulee käyttäjän äänisyöte käymään monta eri tasoa läpi päästäkseen sovellukseen ja tämän jälkeen, että sama ääni päästään toistamaan käyttäjälle samat käyttöjärjestelmän sisäiset tasot on käytävä uudestaan läpi. Tätä prosessia tutkimalla pääsin itse enemmän jyvälle Androidin kymmenen millisekunnin ongelmasta ja haasteista mitä tämän ongelman ratkaisemiseen liittyy.

3.4 Seurantaviikko 04

Maanantai 22.02.2016

Päivän aluksi tein julkaisun Google Playhin. Sitten otin tavoitteekseni aloittaa sovelluksen lokalisaatiota. Ostamme käännöksemme Transfluent-nimisen yrityksen kautta. Koska web puolemme käyttää .po-käännöstiedostoja tutkin mahdollisuutta käyttää myös .po-tiedostoja androidin sisäisen strings.xml:n sijasta. Tämä kuitenkin osoittautui turhaksi vaivaksi, koska Transfluent tukee XML-formaattia. Web-sovelluksemme pääkehittäjä Jani Nyman hankki minulle Transfluent-tilin, ja näytti, miten käännöksiä tilataan. Ensimmäisenä tilasimme venäjän ja espanjan kielen käännökset, että voimme tutkia kuinka paljon ongelmia erikoismerkit ja kyrilliset aakkoset aiheuttavat GUIssämme.

Tutustuessani .po-tiedostoihin huomasin, että web-käännöksistä löytyi samoja sanoja mitä Android-sovellukseen tarvitaan, joten lisäsin suomeksi jo käännetyt sanat suomenkieliseen versioon strings.xml tiedostosta. Android Studion sisäänrakennettu Translation Editor teki tästä syötöstä erityisen helppoa.

Keskustelin myös samassa koulutusohjelmassa olevan Arto-Pekka Hännisen kanssa, mitä mieltä hän olisi työharjoittelusta Singalla webpuolen logiikan implementointihommissa. Olin aikaisemmin viikolla selvittänyt web puolen pääkehittäjältämme Jani Nymanilta, mitkä olisivat tärkeimmät asiat osata, jos tulee meille töihin, ja jaoin tämän tiedon Arto-Pekan kanssa. Hän alkoi kirjoittaa siltä istumalta työhakemustaan.

Tiistai 23.02.2016

Huomasin, että osa sovelluksessa ilmenevästä tekstistä ei oltu lokalisoitu. Lähdin sitten lokalisoimaan eli hakemaan strings.xmlän kautta kaikki mahdolliset stringit. Asetukset, Ilmoitukset, ProfileTabs, SearchTab ja RecordingView olivat jääneet epähuomiossa kääntämättä. Lisäsin myös kontekstin haun SingaApplicationiin että pystyin lisäämään Notification-objektiin lokalisoituja stringejä. Objekteissa harvoin on määritelty kontekstia, joka on resurssien haun yhteydessä välttämätön (Kuva 27).

```
} else if (verb.equalsIgnoreCase("like") && actor instanceof Singer && object instanceof Recording) {
    if (object.getId() == SingaApplication.getInstance().getContentManager().getAuthUser().getId()) {
        text = ((Singer) actor).getUsername(true) + " " + SingaApplication.getContext().getResources().getString(R.string.liked_your_perf);
    } else {
        text = ((Singer) actor).getUsername(true) + " " + "liked a performance by" + " " + ((Recording) object).getSinger().getUsername(true) + ".";
    }
```

Kuva 27. Esimerkki miten haetaan strings.xml tiedostosta stringejä objektin sisällä, jossa ei ole erikseen määriteltyä kontekstia.

Keskiviikko 24.02.2016

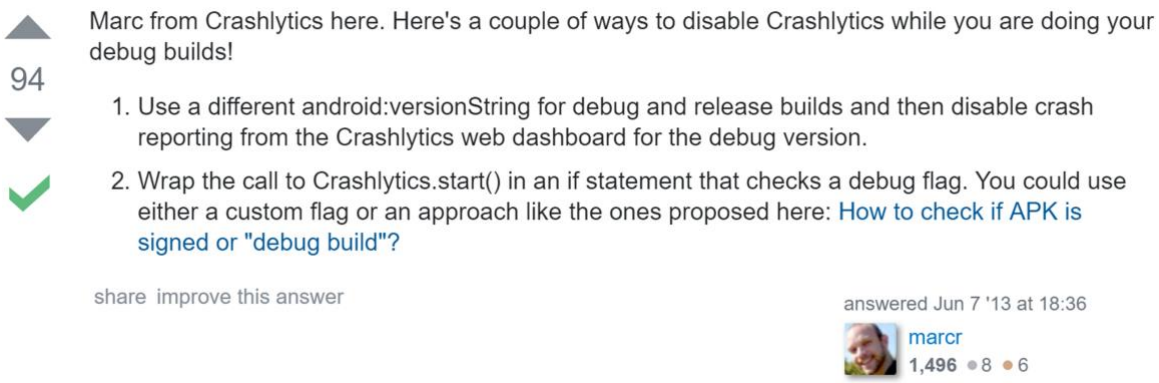
Huomasin että Genret, Search headerit ja LanguageGenerator olivat jääneet ilman stringejä. Hoidin genrejen käännökset yksinkertaisella switch-casella (Kuva 28). Muut lokalisaatiot pystyin tekemään normaalisti string-resursseja apuna käyttäen.

```
public void setNameValue(String nameFromJson){
    switch (nameFromJson){
        case "Alternative":
            nameValue = SingaApplication.getContext().getResources().getString(R.string.alternative);
            break;
        case "Ballad":
            nameValue = SingaApplication.getContext().getResources().getString(R.string.ballad);
            break;
        case "Blues":
            nameValue = SingaApplication.getContext().getResources().getString(R.string.blues);
            break;
        case "Country":
            nameValue = SingaApplication.getContext().getResources().getString(R.string.country);
            break;
```

Kuva 28. Perus switch-case, joka jsonista parsitun stringin perusteella valitsee käytettävän stringin.

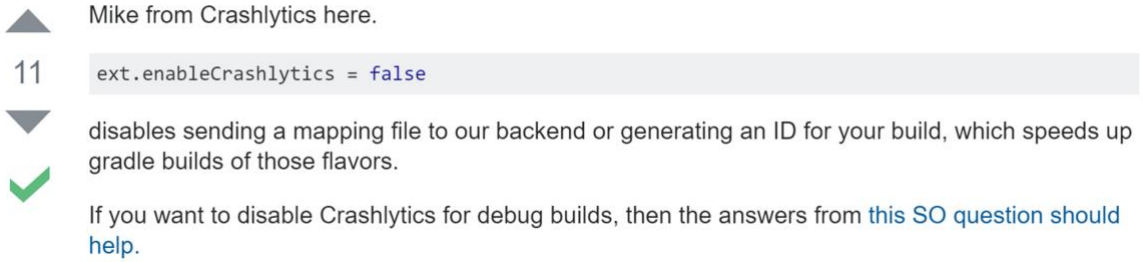
Torstai 25.02.2016

Aamulla tullessani toimistolle full stackimme Juuso Koskenseppä mainitsi, kuinka paljon häntä häiritsee debugatessa Crashalyticsin turhat kaatumisilmoitukset. Yhdyin Juuson mielipiteeseen, joten aloin tutkia, josko debugatessa Crashalyticsin saisi disabloitua. Löysinkin heti vastauksen stackoverflowsta (Kuva 29).



Kuva 29. Crashalyticsin Marcin vastaus ongelmaamme luettavissa http://stackoverflow.com/a/16990788 Luettu:18.5.2016

Valitettavasti tämä vastaus oli väärin. Olimme jo tehneet kuten Marc ohjeisti, mutta Crashalytics ei lopettanut kaatumisilmoitusten lähettämistä. Jatkoin siis stackoverflown kaluamista, ja löysin toisen mahdollisen vastauksen (Kuva 30).



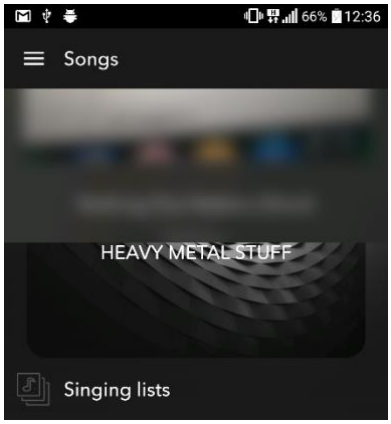
Kuva 30. Crashalyticsin Miken vastaus ongelmaamme luettavissa http://stackoverflow.com/a/28357998 Luettu: 18.5.2016

Valitettavasti, kun edellä mainitun vastauksen koodinpätkän lisää gradlen debug buildiin, Crashalytics estää sovelluksen buildaamisen ja kaataa buildausyritykset. Myöhemmin huomasimme, että sovelluksen applikaatiolayerissä täytyy vielä erikseen estää Crashalyticsin alustus sovelluksen debug-versiossa.

Lauantai 27.02.2016

Freelancerimme Ville Vallan kanssa päätimme tutkia sovelluksemme suurimpia bugeja. Ville osasi heti sanoa, että suurin kaatumisten aiheuttajamme johtui Toastista, jota yritettiin luoda ennen kuin UI-säiettä oli edes luotu. Kyseinen Toast halusi ilmoittaa ContentManagerissa, että laite ei ole muodostanut internetyhteyttä. Tästä syystä siirsimme yhteyden tarkistuksen toasteineen MainActivityyn.

Toinen korjaamamme virhe oli applikaation avautuessa, MainActivityssä vilkahtava profiilinäkymän taustakuva karusellin päällä (Kuva 31). Tämä johtui siitä että toolbar_main.xml:ässä oli profilecontentin visibility vaihdettu näkyväksi ja siksi piirtyi ennen näkymättömäksi muuttumista hitaimmilla laitteilla. Vaihdoimme siis vain visibilityn takaisin näkymättömäksi xml-tiedostoon, ja ongelma oli korjattu (Kuva 32).




```
<LinearLayout
    android:id="@+id/profilecontent"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="?attr/actionBarSize"
    android:minHeight="?attr/actionBarSize"
    android:orientation="horizontal"
    android:paddingBottom="18dp"
    android:paddingLeft="12dp"
    android:paddingRight="12dp"
    android:paddingTop="18dp"
    android:visibility="gone"
    app:layout_collapseMode="none">
```

Kuva 31 ja 32. Kuva UI-virhetilanteesta ja sen korjauksesta.

Viikkoanalyysi

Olen viimeisen parin viikon aikana tehnyt paljon rekrytointia, koska olen päättänyt hankkia Singalle kesätyöharjoittelijoita. Oikean tiimin kasaaminen on todella tärkeää startupille, koska palkkausresurssit ovat rajalliset. Tämän lisäksi olen tutkinut hiukan Androidin säikeitä ja prosesseja.

Kun jokin sovelluksen komponentti käynnistyy ja sovelluksella ei ole muita käynnissä olevia komponentteja, Android järjestelmä sykäyttää liikkeelle uuden Linux prosessin sovellusta varten yhdellä suoritettavalla säikeellä (McAnlis, 2016). Oletuksena kaikki saman sovelluksen komponentit ajetaan samassa prosessissa ja säikeessä, jota kutsutaan pääsäikeeksi. Jos sovelluksen komponentti käynnistyy ja sovelluksella on jo prosessi käynnissä, käyttää komponentti jo käynnissä olevaa prosessia ja säiettä. Sovelluksen kehittäjä pystyy kuitenkin luomaan erilaisia komponentteja ja ajamaan niitä eri prosesseissa ja luomaan lisäsäikeitä mille tahansa prosessille.

Pääsäie on todella tärkeä, koska se hallitsee tapahtumien lähettämistä oikeille käyttöliittimän pienoisohjelmille, esimerkkinä käyttöliittymän piirtotapahtumat. Tällä säikeellä tapahtuvat melkein kaikki sovelluksen vuorovaikutukset Android UI toolkitin kanssa – toolkitiin sisältyvät kaikki android.widget ja android.view paketit (Android Developers Documentation 2018). Tästä johtuen joskus pääsäiettä kutsutaan UI-

säikeeksi. Kaikki komponentit, jotka ajetaan samassa prosessissa, saavat alkunsa UI-säikeessä, ja järjestelmän lähettämät kutsut näille komponenteille suorittuvat tällä säikeellä. Tästä syystä kaikki metodit, jotka vastaavat järjestelmän callbackeihin – kuten onKeyDown() kun halutaan saada käyttäjän kosketuksista tietoa tai sovelluksen elinkaaren callback-metodit – suoritetaan aina prosessin UI-säikeellä. Esimerkiksi kun käyttäjä painaa nappia ruudulla, sovelluksen UI-säie lähettää kosketustapahtuman pienoisohjelmalle, joka vuorostaan muuttaa napin painettuun tilaan ja lähettää mitätöimispyynnön tapahtumajonoon. UI-säie poistaa pyynnön jonosta ja ilmoittaa pienoisohjelmalle, että sen pitää uudelleen piirtää itsensä ruudulle.

Kun sovellus tekee intensiivistä työtä tällaisen vuorovaikutuksen vastatoimena, yhden säikeen malli saattaa aiheuttaa ongelmia suorituskyvyssä. Esimerkiksi jos sovelluksessa kaiken tekee UI-säikeellä, pitkät operaatiot kuten tietokanta- ja verkkokutsut estävät kokonaan käyttöliittymän toiminnan. Kun säie on estynyt, mikään tapahtuma ei välity eteenpäin, eivät edes käyttöliittymän piirtymistapahtumat. Käyttäjälle käyttöliittymä jäätyy kokonaan. Pahimmillaan jos UI-säie on estynyt yli 5 sekunnin ajan, käyttäjä saa ”application not responding” dialogin eteensä. Yleensä tässä vaiheessa käyttäjä voi jopa päättää poistaa sovelluksen. Android UI toolkitiä vuorostaan pitää manipuloida UI-säikeessä.

Jos operaatioita, jotka eivät ole välittömiä, pitää suorittaa, on tärkeää tehdä ne joko tausta- tai työsäikeellä. Pitää myös muistaa, että UI-päivityksiä ei voi tehdä muulla kuin UI-säikeellä. Singalla käytämme hyvin usein Activity.runonUiThread(Runnable) metodia, mutta View.post(Runnable) ja View.postDelayed(Runnable, long) ovat myös hyvin tärkeitä apumetodeja, joilla päästään muiden säikeiden puolelta kiinni UI-säikeeseen (Kuva 33).

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // processBitMap vie mahdollisesti paljon aikaa
            final Bitmap bitmap = processBitMap( s: "image.png");
            // käytetään View.post(Runnable) metodia että saadaan
            // UI päivitettyä UI-säikeessä
            imgView.post(new Runnable() {
                public void run() {
                    imgView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Kuva 33. Säikeille turvallinen metodi. Taustaoperaatio processBitMap(String) tehdään eri säikeellä, kun taas ImageView:n setBitmap(Bitmap) suoritetaan UI-säikeellä.

Kun operaatiot monimutkaistuvat, tulee tällaisen koodin ylläpitämisestä vaikeampaa (Kuva 33). Todennäköisesti paras vaihtoehto on laajentaa AsyncTask-luokkaa, joka yksinkertaistaa työsäikeiden suorittamisen, kun näiden täytyy olla yhteydessä käyttöliittymään.

AsyncTask tekee asynkronisia tehtäviä käyttöliittymässäsi. Se suorittaa taustaoperaatiot työsäikeellä ja sitten näyttää tulokset UI-säikeellä, ilman että erillistä säikeiden käsittelyä (Göransson 2014). Asynkronisia tehtävät vastaanottavat kolmen tyyppisiä parametrejä – Params, Progress ja Result – ja ne suoriutuvat neljässä osassa – onPreExecute, doInBackground, onProgressUpdate ja onPostExecute. AsyncTaskia käytetään aliluokkana, joka ohittaa yhden tai useamman metodin (Kuva 34). Kun olet luonut aliluokan, voit suorittaa AsyncTaskin kutsumalla execute-metodia (Kuva 35).

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Jos käyttäjä peruuttaa tapahtuman
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }


    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Kuva 34. Esimerkki kuinka AsyncTaskia voisi käyttää hyväkseen ohittamalla doInBackgroundin, onProgressUpdaten ja onPostExecuten.

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Kuva 35. Esimerkki asynkronisen tehtävän kutsumisesta.

AsyncTask käyttää kolmenlaisia geneerisiä tyyppejä. Paramsit ovat parametrejä, joita lähetetään tehtävälle sitä suorittaessa, ensimmäisessä esimerkissä doInBackground ottaa vastaan URLeja ja toisessa esimerkissä AsyncTaskille syötetään URLeja (Kuva 34 ja 35). Progressit ovat taustalaskennan valmiusluku, esimerkissä onProgressUpdate näyttää edistymisen Integerinä eli prosentteina (Kuva 34). Resultit taas edustavat taustalaskennan tuloksen tyyppiä. Aina näitä kaikkia ei tosin tarvita, jolloin luodessa AsyncTaskia käyttämättömät parametrit voi merkata Voidiksi (Kuva 36).

```
private class MyTask extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... voids) {
        return null;
    }
}
```

Kuva 36. Asynctask ilman Params, Progress tai Result parametrien käyttöä.

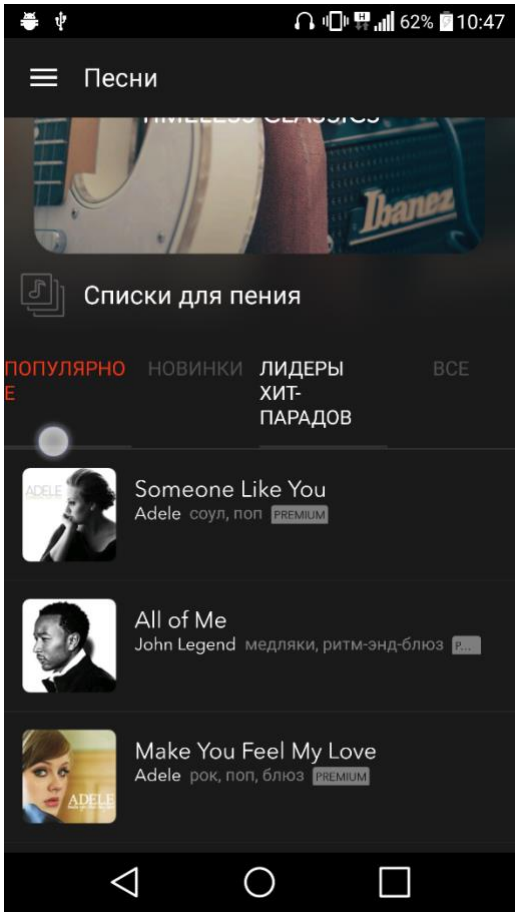
onPreExecute()-metodi käynnistyy UI-säikeellä ennen kuin AsyncTask suoritetaan. Tässä osassa tehtävää voidaan esimerkiksi tuoda tehtävän edistymispalkki näkyviin tai tehdä muita asioita ennen kuin AsyncTask alkaa suoriutumaan. doInBackground(Params…)-metodia kutsutaan taustasäikeellä heti kun onPreExecute() on tehnyt taikansa. Tässä osassa tehtävää suoritetaan pitkään kestäviä kommervenkkeja. AsyncTaskin saamat parametrit – Params - siirtyvät tälle metodille. onProgressUpdate(Progress…)-metodi taas kutsuu UI-säikeellä publishProgress(Progress…)-metodia. Tätä osaa tehtävästä yleensä käytetään näyttämään käyttäjälle tehtävän edistyminen reaaliajassa, esimerkiksi animoimalla edistymispalkkia. Kun AsyncTask on valmis, suorittaa se UI-säikeellä onPostExecute(Results)-metodin. Taustalaskennan tulos yleensä siirretään tälle metodille parametrina.

Singalla käytämme esimerkiksi kuvien ja tiedostojen lataukseen asynkronisia tehtäviä, koska ne soveltuvat siihen tehtävään täydellisesti. AsyncTaskit ovat valtavan hyvä työkalu olla Android kehittäjän takataskussa. Nykyään suurin osa sovelluksista hakee tietoa verkkokutsuilla tai lähettää verkkoon tiedostoja ja asynkroniset tehtävät tekevät tästä palan kakkua jumittamatta sovellusta.

3.5 Seurantaviikko 05

Tiistai 01.03.2016

Päätin parin seuraavan päivän aikana keskittyä optimoimaan venäjänkielisiä käännöksiä, koska ne rikkoivat mukavasti designin (Kuva 37). Kävin myös pitkän keskustelun venäjän kielen kääntäjän kanssa venäjän käännöksistä, käyden kaiken mahdollisen lävitse hänen kanssaan (Kuva 38). Käytämme kääntäjinämme Transfluentia, ja onneksemme heidän kääntäjänsä ovat nopeita vastaamaan asiakaspalautteeseen. He myös vastaavat palvelunsa kautta kysymyksiin hyvällä tahdilla ja tarmolla.



No notifications. - Нет уведомлений

notifications =>

1,21,31,41..91 = уведомление
2..4,22..24,32..34,92..94 = уведомления
25..30,35..40,45..50,.. 95..100 = уведомлений

recordings. =>

1,21,31,41..91 = запись
2..4,22..24,32..34,92..94 = записи
25..30,35..40,45..50,.. 95..100 = записей

added => (username) added (1-100 recordings.)
добавил/-а

but

Over (5-100 new songs added!

In this case, added = добавлено

(started following) users =>

1,21,31,41..91 = пользователя
2..4,22..24,32..34,92..94 = пользователей
25..30,35..40,45..50,.. 95..100 = пользователя

Kuva 37 ja 38. Muuttamaton venäjän käännösversio ja lyhyt ote venäjän käännöskukkasia.

Keskiviikko 02.03.2016

Aamun roadmap-sessiossa revisiomme Androidin aikajanaa. En ollut päässyt vielä ollenkaan aloittamaan playlistejä, koska lokalisaatio oli vienyt paljon huomiota. Pistin todella paljon aikaa lokalisaatioille ja playlisteille, että aikaa jäisi paukuttaa kumpikin finaaliin aikataulussa (Taulukko 3).

vko	9-12	13-15	15->

Android	Playlist Lokalisaatio	Soittojono	Chromecast

Taulukko 3. Singan roadmap 02.03.2016

Roadmap-session jälkeen kävin sovelluksen läpi ja keskustelin lisää kääntäjän kanssa vaihtoehtoisista käännöksistä. Onneksemme kääntäjämme osasi antaa esimerkiksi sanalle top synonyymin, joka ei ollut 3 sanaa pitkä (Kuva 38). Suoritin venäjän käännösten aiheuttamat muutokset ja pääsin tavoitteeseeni, jonka olin tiistaina itselleni antanut.

Perjantai 04.03.2016

Asetin tavoitteekseni saada laululistoja vähän eteenpäin. Kääntäjät eivät valitettavasti tee töitä viikonloppuisin, joten päätin tehdä lokalisaatiotilausten maanantaina ja työstää laululistoja.
Aloitin työpäiväni lisäämällä Playlist-luokkaan song_count attribuutin, joka haetaan ObjectParseFactorylla, joka hakee jsonista attribuutit ja tallettaa ne omaan objektiinsa (Kuva 39). Haluan antaa laulujen lukumäärän, kun luettelemme käyttäjän omat laululistat (Kuva 40).

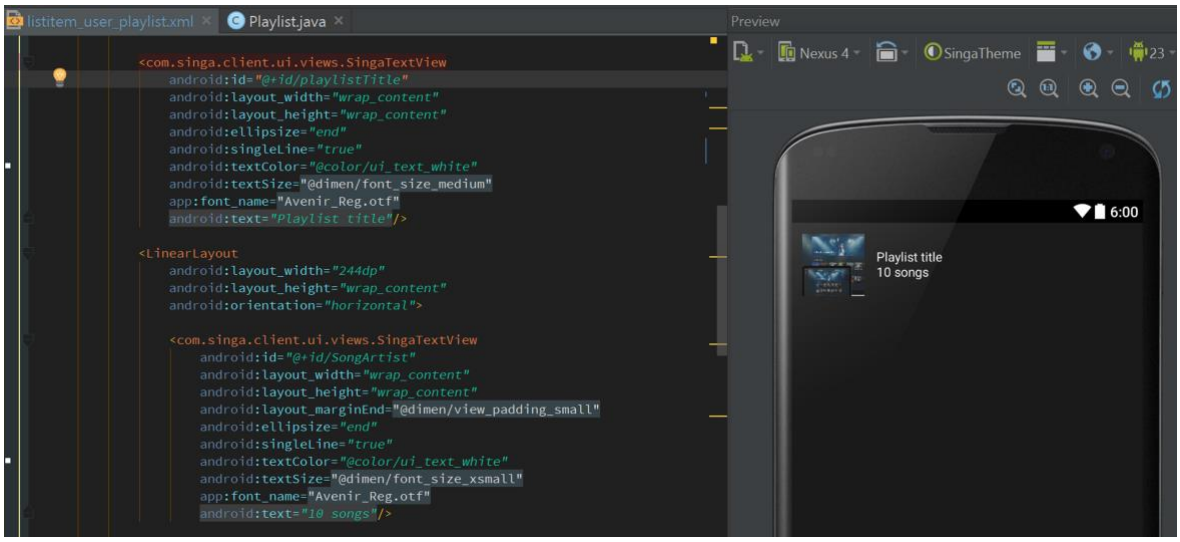
```
private static Playlist parsePlaylist(JSONObject json) throws JSONException {
    Playlist result = new Playlist();

    result.setId(json.getInt("id"));
    result.setTitle(json.getString("title"));
    result.setCreated(new DateTime(json.getString("created")));
    result.setIndex(json.getInt("index"));
    result.setUser(parseSinger(json.getJSONObject("user")));
    result.setSong_count(json.getInt("song_count"));

    if (json.has("cover") && !json.isNull("cover"))
        result.setCover(parseImage(json.getJSONObject("cover")));
    if (json.has("background") && !json.isNull("background"))
        result.setBackground(parseImage(json.getJSONObject("background")));
    if (json.has("image") && !json.isNull("image"))
        result.setImage(parseImage(json.getJSONObject("image")));

    return result;
}
```

Kuva 39. ObjectParseFactory:ssä jsonista parsitaan Playlistin tiedot ja ne tallennetaan Playlist-objektiin.



Kuva 40. Käyttäjän playlistien listauksen xml-tiedosto ja graafinen preview.

Illalla menimme Oppexin toimistolle Reaktor Venturesin järjestämään iltaan, jossa pääsimme tapaamaan muita Reaktorin sijoituskohteita. Pääsin esimerkiksi SportSetterin Lead Software Developerin ja HubChatin Senior Mobile Developerin kanssa juttusille ja puhuimme erilaisista mobiiliratkaisuista, joita startupit olivat tehneet. Hyvin usean iOS devaajan kanssa sain käsityksen, että mobiilipresenssi on nousevissa määrin tärkeää peliyritysten lisäksi myös palvelualoilla ja muissa kaupanalan yrityksissä.

Lauantai 05.03.2016

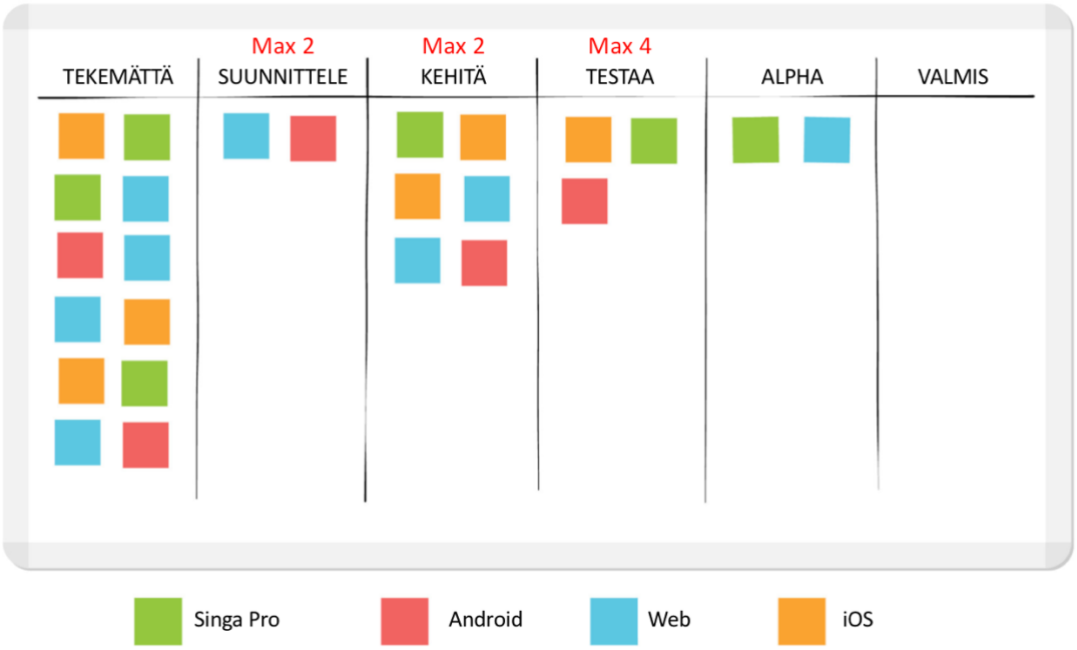
En odottanut tekeväni lauantaina töitä, mutta sain tiedon Software Developeriltamme Jani Nymanilta, että rekisteriselosteemme, jota sivullamme kutsutaan tietosuojaksi, ja käyttöehtojemme URLit olivat vaihtuneet. Henkilötietolaissa toisen luvun, kymmenennessä pykälässä sanotaan että ”Rekisterinpitäjän on pidettävä rekisteriseloste jokaisen saatavilla.” eli meidän on pidettävä lain mukaan asiakkaillemme käsillä näitä tietoja. Edellinen alpha-versiomme Singasta oli ollut viikon testauksessa ilman suurempia kaatumisia ja päätin vaihtaa URLit ja laittaa alphan tuotantoon.
Kävin myös läpi työharjoittelijatarpeitamme Web Front End Developerimme, Konsta Tzoulaksen kanssa. Hän informoi minua ystävästään, joka saattaisi olla kiinnostunut tekemään töitä Singalle. Valitettavasti tältä ehdokkaalta ei löytynyt LinkedIn profiilia, joten en voinut tarkemmin tutkia hänen mahdollisuuksiaan päästä Singalle töihin, mutta kerroin kuinka meiltä voisi hakea töitä.

Viikkoanalyysi

Uusien roadmappien takia meillä oli taas puhetta työskentelyviitekehyksen vaihtamisesta. Hyvin keskeisesti meillä puhuttiin Kanbanista, joka pohjautuu Toyota Production System järjestelmään, joka puolestaan on saanut inspiraationsa supermarketeista (Anderson 2010). Kaupoissa hyllyt täytetään kysynnän mukaan, ei tavarantoimittajan tarjonnan mukaan. Koska varastotasot vastaavat menekkiä, supermarkettien varastonhallinnan tehokkuus kasvaa vähentäen ylijäämän aiheuttamia varastointikuluja. Kuluttaja löytää silti aina tarvitsemansa tavaran kaupan hyllyltä. Kun Toyota alkoi käyttää samaa järjestelmää omissa tehtaissaan, tavoitteena oli saada inventaario tasoihin varsinaisen materiaalien kulutuksen kanssa. Että kaikilla työntekijöillä olisi ajantasainen käsitys kapasiteetista, tiimien välillä liikkui kortti - japaniksi kanban. Esimerkiksi kun tuotantolinjan materiaalilaatikko on tyhjentynyt, lähetetään varastolle kanban, jossa kerrotaan mitä materiaalia tarvitaan ja tarkalleen paljonko sitä tarvitaan. Varastolla olisi uusi materiaalilaatikko valmiina odottamassa, jonka he lähettävät tuotantolinjalle ja vuorostaan he lähettävät kanbanin tavaranhankkijalle, joka vuorostaan lähettää materiaalilaatikon varastolle. Vaikka kommunikaatiovälineet ovat kehittyneet Kanbanin keksimisen jälkeen, Just-In-Time-tuotannon periaate on edelleen Kanbanin keskiössä.

Ketterässä ohjelmistokehityksessä voidaan käyttää Just-In-Time-tuotannon periaatteita tasaamalla työn alla olevat tehtävät ja tiimin kapasiteetti. Näin tiimi pystyy suunnittelemaan työtään joustavammin ja tuottamaan tulosta nopeammin, samalla työn painopisteet selkenevät ja työskentelystä tulee läpinäkyvämpää (Björkholm 2014). Kanban on saanut todennäköisesti siksi suosiota ohjelmistokehityspiireissä, koska se on helppo ja nopea ottaa käyttöön, kun ymmärtää perusperiaatteet. Kaikki mitä softatiimit tarvitsevat alkaakseen käyttää Kanbania ovat taulu ja nippu tauluun kiinnitettäviä kortteja, mitkä yhtä hyvin voivat olla virtuaalisia.

Kanban-tiimien työ pyörii Kanban-taulun ympärillä, millä pyritään visualisoimaan työtä ja optimoimaan työtahti tiimille sopivaksi. Taulun avulla on helppo luoda työlle standardit ja samalla pullonkaulat on helppo identifioida ja ratkaista. Perus Kanban-taulussa on kolme osaa johon työtehtävät sijoittuvat – tekemättä, työn alla ja valmis. Riippuen tiimin koosta, rakenteesta ja tavoitteista taulun osia voi lisätä ja muuttaa tiimin omaan prosessiin sopiviksi. Näillä osioilla on myös rajat, kuinka monta työtehtävää saa olla työn alla samaan aikaan. Esimerkiksi jos tiimissä on vain yksi taustajärjestelmien osaaja, voidaan tiimin kesken sopia, että hänellä ei voi olla työn alla kuin maksimissaan kaksi työtehtävää saman aikaisesti (Kuva 41).



Kuva 41. Esimerkki Kanban-taulusta, millaista Singalla voitaisiin käyttää.

Jokaista työtehtävää edustaa yksi lappu Kanban-taululla, mikä tekee työtehtävien edistymisen seurannasta helppoa. Lappuun on kirjoitettu ylös kaikki tärkeä tieto työtehtävästä, kuten kuka on vastuussa työtehtävästä, lyhyt kuvaus tehtävän sisällöstä, milloin työnteko on aloitettu ja niin edelleen (Kuva 42). Kun kaikki tiimin jäsenet näkevät työhön liittyvät yksityiskohdat, tämä varmistaa keskittymisen tärkeisiin asioihin, työn jäljitettävyyden ja mahdollistaa riippuvuuksien ja esteiden nopean tunnistamisen.

Kuva 42. Esimerkki Kanban-lapusta.

Kanban-tiimi keskittyy tekeillä olevaan työhön. Kun tiimi saa yhden työtehtävän tehtyä, he ottavat seuraavan työtehtävän itselleen taululta tehtäväksi. Tehtäviä voi priorisoida vapaasti ennen kuin niitä aletaan työstämään, koska nykyisten työtehtävien ulkopuoliset tehtävät eivät vaikuta tiimiin. Kunhan kaikista tärkeimmät tehtävät ovat priorisoituna ensimmäiseksi, pystyy kehitystiimi tekemään parasta mahdollista tulosta (Kniberg 2009).

Scrumin tapaisia sprinttejä ei siis käytetä, vaan työtä tehdään jatkuvalla syötöllä.

Kanbanissa mitataan tehtäviin kulutettua aikaa, siitä hetkestä, kun työtehtävä aloitetaan, siihen asti, kunnes se on tuotannossa. Kun tehtäviin kulutettua aikaa alkaa mittaamaan, pystyy tiimi tulevaisuudessa arvioimaan paremmin kuinka kauan samankaltaisiin tehtäviin tulee kulumaan aikaa. Myös päällekkäiset osaamiset vähentävät aikaa mikä kuluu työtehtäviin. Jos vain yhdellä henkilöllä tiimissä on tarvittavat taidot tietyn ongelman ratkomiseen, ei hän pysty saamaan tukea keneltäkään muulta tiimin jäseneltä ja tästä voi helposti muodostua pullonkaula. Kanbanissa on koko tiimin vastuulla varmistaa, että työt sujuvat kitkattomasti. Esimerkiksi taustajärjestelmien herrasmiehet voivat myös testata käyttöliittymää, jos vain suinkaan ehtivät.

Monien asioiden tekeminen samanaikaisesti on myrkkyä tehokkuudella. Mitä enemmän palloja on ilmassa samanaikaisesti, sitä hankalampi tehtäviä on saada valmiiksi. Tämän vuoksi yksi Kanbanin perusperiaatteista on rajata yhtä aikaa työn alla olevien tehtävien määrää. Työn alla olevien tehtävien määrän rajoitus korostaa pullonkauloja ja ongelmia tiimin keskittymisessä, jäsenissä tai osaamisessa. Singalla varsinkin taustajärjestelmien asiantuntijamme ja web-sovelluksemme pääkehittäjä ovat kummatkin monen tehtävän loukussa ja Kanbanin samanaikaisten työtehtävien rajaus olisi heille todennäköisesti suuri helpotus. Päädyimme kuitenkin jälleen kerran jättämään agiilin viitekehyksen vaihdon tulevaisuuteen.

3.6 Seurantaviikko 06

Maanantai 07.03.2016

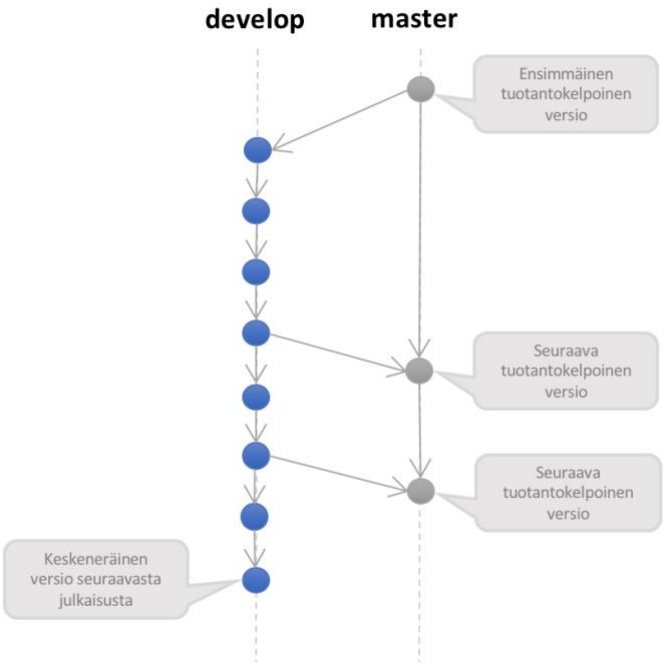
Aloitin työpäiväni lataamalla kaikki tähän mennessä tehdyt käännökset Transfluentista. Kävin valmiit .po-käännöstiedostot läpi rivi riviltä ja lisäsin tarvittavat käännökset Android Studion Translation Editorilla kuhunkin kieleen. Poistin myös jo saadut tiedot, käännettävät rivit sisältävästä tiedostosta, jonka lähetin eteenpäin kääntäjille.

Perjantai 11.03.2016

Päivän päätteeksi kävin iOS-kehittäjämme Antti Oinaalan kanssa läpi, miten hän oli suunnitellut toteuttavansa Playlistit, ja keskustelimme siitä, kuinka yhteneväiseen logiikkaan voisimme päästä Androidilla ja iOS:llä. Antti oli päässyt jo todella pitkälle logiikan implementoinnissa, kun taas itse en vielä ole päässyt kunnolla alkuun.

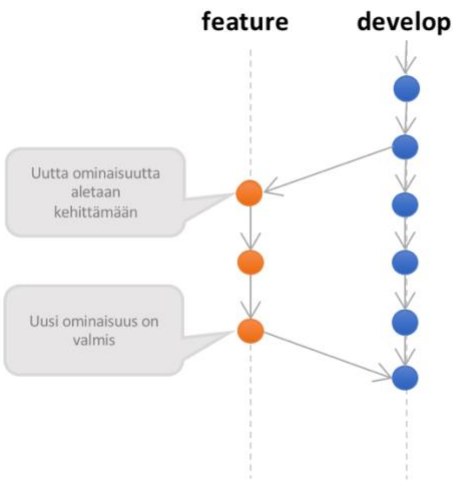
Viikkoanalyysi

Viime aikoina olen käyttänyt paljon aikaa Singan versionhallinnan organisoimiseen. Haluamme käyttää hyvin universaalia git flowta, jotta yhteistyö helpottuisi ja uusien työntekijöiden olisi siten helpompi päästä mukaan tekemiseen (Westby, 2015). Keskeisimmät repositoryt ovat master ja develop, jotka kehittyvät ja muuttuvat koko ajan (Kuva 43). Develop ja master repositoryt eivät myöskään koskaan poistu käytöstä (Santacroce 2017). Jaetun git-hakemiston masterissa on aina julkaisuvalmis versio lähdekoodista. Jaetun git-hakemiston developissa lähdekoodi on tilassa, jossa viimeisimmät tehdyt muutokset ovat ennen seuraavaa julkaisua. Kun develop-haara on stabiili ja julkaisuvalmis, kaikki muutokset yhdistetään merge-komennolla masteriin. Jos Singalla alettaisiin käyttämään automaattista tuotantojulkaisu skriptaa, olisi todella tärkeää, että masterissa on täysin tuotantokelpoinen stabiili versio sovelluksesta.



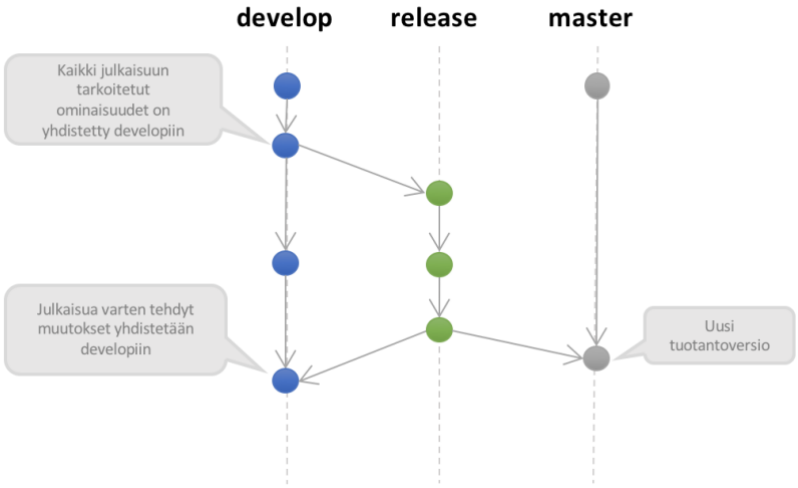
Kuva 43. Develop- ja master-kehityshaarojen suhde visualisoituna

Hyödyntämämme kehitysmalli käyttää apurepositoryjä, jotka helpottavat ominaisuuksien kehityksen seurantaa, valmistavat tuotanto julkaisuja ja avustavat tuotannossa ilmenneiden ongelmien nopeassa korjaamisessa (Driessen 2010). Apurepositoryjä on kolmen tyyppisiä; feature, release ja hotfix. Nämä ovat perus Git kehityshaaroja ja niiden nimet määräytyvät vain siitä, miten niitä käytetään. Jokaisella kehityshaaralla on omat sääntönsä mistä haarasta ne saavat alkunsa ja mihin ne yhdistetään merge-komennolla.



Kuva 44. Feature- ja develop-kehityshaarojen suhde visualisoituna

Feature-kehityshaara saa alkunsa developista ja yhdistyy developiin (Kuva 44). Featureksi nimetyssä kehityshaarassa kehitetään uusia ominaisuuksia tuleviin julkaisuihin. Tämä kehityshaara on olemassa niin kauan kuin tätä tiettyä ominaisuutta kehitetään. Kun ominaisuus on valmis, kehityshaara yhdistetään developiin tai poistetaan jos ominaisuus osoittautuu turhaksi.

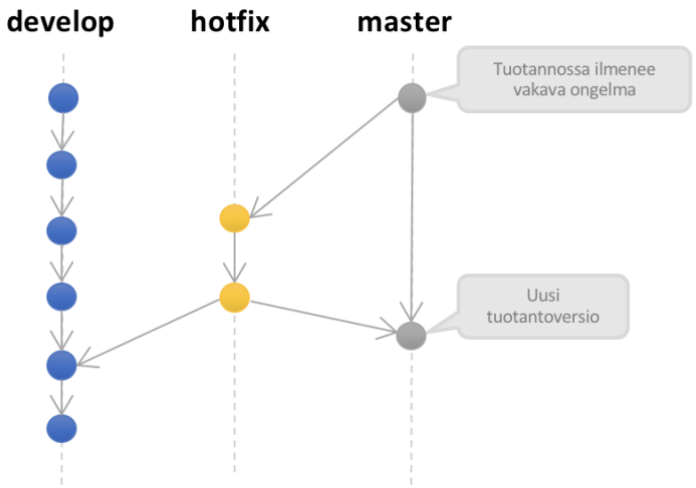


Kuva 45. Release-, develop- ja master-kehityshaarojen suhde visualisoituna

Release-kehityshaara saa alkunsa develop-kehityshaarasta, kun develop on liki julkaisuvalmis (Kuva 45). Vähintään kaikki ominaisuudet seuraavaa julkaisua varten ovat yhdistetty merge-komennolla developiin ennen releasen luomista. Kun release-kehityshaaraan on tehty kaikki tarvittavat muutokset, se yhdistyy merge-komennolla developiin ja masteriin. Releasen tarkoitus on tukea uuden tuotantojulkaisun tekoa. Tässä haarassa voi vielä tehdä viimeistelyjä ominaisuuteen ja valmistella sovelluksen metadata

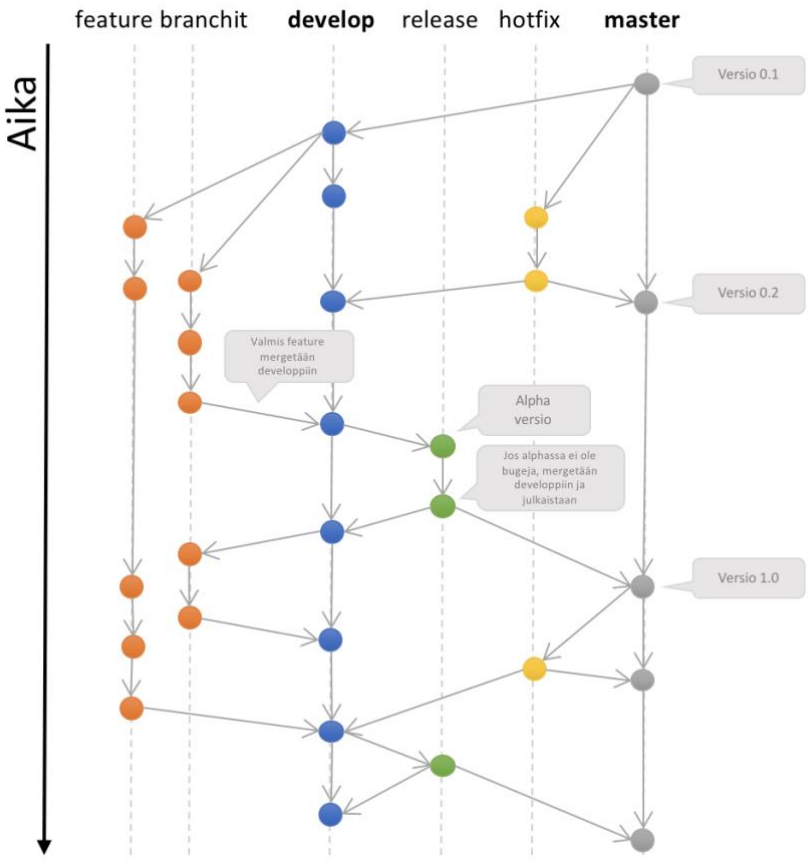
julkaisua varten, esimerkiksi versionumeron muutoksella. Kun tämän tekee erikseen release-kehityshaarassa, pystyy develop sillä välin ottamaan vastaan seuraavia ominaisuuksia tulevaa julkaisua varten.

Hotfix-kehityshaara on hyvin samankaltainen releasen kanssa, siinä mielessä, että ne tukevat uuden tuotantojulkaisun tekoa, vaikka ovatkin suunnittelemattomia. Hotfix-kehityshaara luodaan, kun tuotannossa on havaittu vakava ongelma, joka pitää korjata välittömästi. Masterista alkunsa saava kehityshaara yhdistetään merge-komennolla lopuksi master- ja develop-kehityshaaroihin (Kuva 46). Ainoa poikkeus näihin sääntöihin on, kun release-kehityshaara on jo luotuna. Silloin hotfix yhdistetään release-haaraan. On hyvin tärkeä luoda oma kehityshaara ongelman ratkaisemiseksi, että develop-haaran kehitystä pystytään jatkamaan ongelmitta.



Kuva 46. Hotfix-, develop- ja master-kehityshaarojen suhde visualisoituna

Singalla emme siis olleet kaukana ideaalista git flowsta (Kuva 23). Julkaisuputkemme parantamiseksi kuitenkin päätimme muuttaa hieman git malliamme (Kuva 47). Muutos ei ole maailman suurin, mutta jotta yhteistyö sujuisi mahdollisimman hyvin ja ongelmitta näimme tärkeäksi aktivoida master-kehityshaaramme käytön.



Kuva 47. Singan uusi git flow

4 Pohdinta ja päätelmät

Työnkuvani yrityksessä selkeentyi opinnäytetyöni raportoinnin aikana. Vaikka töihini pääasiallisesti olisi kuulunut sovelluksen muokkaaminen Chromecastille sopivaksi huomasin, kuinka suurin osa ajastani meni sovelluksen jatkokehitykseen ja ylläpitoon. Työtehtäväni myös laajentuivat rekrytointiin ja opinnäytetyön kirjoittamisen loputtua myös asiakaspalautteeseen vastaamisiin. En ollut aikaisemmin hahmottanut kuinka paljon luin työpäiväni aikana informatiivisia artikkeleita ja kuinka syvälle joihinkin ongelmiin paneuduin työni ohella. Kehityin myös työtiimin jäsenenä opinnäytetyön tuoman itsetutkiskelun myötä.

Lähtötilanteessa tiesin mitä työtehtäväni Android sovelluskehittäjänä pitivät sisällään. Työn raportoinnin aikana on tosin selvinnyt, kuinka laaja työnkuva startupissa työskentely on pelkän sovelluskehityksen lisäksi. Työ on hyvin ihmisläheistä ja vaatii paljon mukautumiskykyä ja lähtöä omalta mukavuusalueelta. Yhtenä päivänä saatat joutua messuille tunnin varoitusajalla, seuraavana päivänä et liiku työpisteeltäsi, koska teet syvä luotaavaa tutkimusta asiaan mistä et ollut aikaisemmin tiennyt mitään, ja sitä seuraavana päivänä istutte pitkässä palaverissa ja pyritte yhdessä ratkaisemaan jonkin yrityksen ongelmista. Koodaajan työ startupissa sisältää paljon enemmän kuin koodin kirjoittamista ja designien orjallista seuraamista. Tärkeimpänä työtehtävänäni näen kuitenkin edelleen hyvän käyttökokemuksen luomisen loppukäyttäjälle. Sovelluksen käyttämisen pitää tuntua luonnolliselta, ja sovelluksen tärkeimpien osien täytyy olla avoimesti esillä.

Raportointijakson aikana työvälineeni eivät ole muuttuneet. Työskentelen edelleen käyttäen Android Studiota ja Sourcetreetä. Olen kehittynyt Android Studion käytössä sekä teknisesti työskentelynopeuteni puolesta että tietotason kannalta. Työskennellessä olen oppinut käyttämään hyväksi erilaisia tekniikoita ja menetelmiä, esimerkiksi miten Android Studiossa sovelluksen visuaalisen käyttöliittymän ominaisuuksia pystyy parilla xml rivillä esikatselemaan käyttäen tools-lisämääreellä tai miten Sourcetreellä ja Bitbucketilla luodaan pull requesti. Näitä osia työstäni en osannut ennen kuin olin aloittanut opinnäytetyöni.

Opinnäytetyöni alussa pystyin työskentelemään hyvin näiden työkalujen kanssa, mutta nyt koen ymmärtäväni paremmin, miksi asioita tehdään alan standardien mukaisella tavalla sekä miten käyttämäni menetelmät vaikuttavat ryhmänä työskentelyyn. Hyvä versionhallinta hyvin pitkälti informoi ongelmanratkaisua, jos tiedetään missä versiossa sovellusta ongelma on ilmaantunut. Kun kommentit ovat kuvaavia, pystyvät muut helpommin ymmärtämään commitin sisältöä. Olen oppinut myös parantamaan

suorituskykyä listaelementtien taustavärin muutoksella läpinäkyvästä taustan kanssa samaksi väriksi.

Sopeutuessani työpaikkaan kommunikointi tiimien jäsenten välillä on kehittynyt. Pystyn suuntaamaan kysymykseni ja kommenttini oikealle taholle ja esittämään kehitysehdotuksia niin että tulen ymmärretyksi. Tärkeimpänä kehityksen sarana pidän sitä, että olen oppinut ymmärtämään muiden työyhteisön jäsenten rooleja, toimenkuvia ja työtehtäviä. Heidän työnsä ymmärtäminen on ollut integraalia parhaan mahdollisen ratkaisun luomiseksi sovelluksessa, esimerkiksi mikä logiikka on parempi toteuttaa sovelluksessa ja mikä taustajärjestelmässä. Kehitys tällä alueella jatkuu edelleen. Taustajärjestelmiemme ohjelmointikielen Pythonin ja viitekehyksen Djangon opettelu olisi hyvä kehitysaskel kommunikaatiota ja sovelluksen ominaisuuksien suunnittelua varten. Haluaisin enemmän osallistua taustajärjestelmien kehittämiseen ja olen vapaa-ajallani opetellut Pythonin alkeita.

Ohjelmointitaitoni ovat myös kehittyneet selkeästi. Eri toiminnollisuuksien suunnittelu ja toteutus tuntuvat nykyään paljon helpommilta ja luonnollisemmilta. Visuaalisen puolen kehitys on minusta todella mielenkiintoista ja tuntuu että opin koko ajan uusia asioita. Esimerkiksi animaatiot ja siirtymät ovat suuria kokonaisuuksia mitä minun pitäisi vielä harjoitella paljon, mutta niiden oppiminen ei tunnu kovin suurelta esteeltä urani jatkumisen kannalta. Aion jatkossakin kehittää visuaalista silmääni ja pyrkiä mahdollisimman luontaisen tuntuiseen käyttökokemukseen.

Työn lopputuloksesta olen myös oppinut, että sovellus ei ole koskaan valmis. Palvelu minkä asiakkaalle tulee elämään ja mitä asiakkaalle siitä jää käteen on käyttökokemus. Kehityksessä käyttökokemuksen helppouden pitäisi olla päämääränä. Totta kai haluamme tarjota loppukäyttäjälle enemmän uusia toiminnollisuuksia koko ajan, mutta hyvin tärkeää on, että vanhatkin sovelluksen osat toimivat halutulla tavalla ja kehittyvät muun sovelluksen myötä. Jos käyttöliittymä on objektiivisesti monimutkainen ja käyttäjät eivät löydä tärkeisiin ominaisuuksiin täytyy muutoksia tehdä, että käyttäjät jatkavat sovelluksen käyttämistä.

Testaamisen merkitys on hyvin tärkeä arvioidessa käyttökokemusta. Luodessani jotain uutta ominaisuutta tai tehdessäni korjauksia sovellukseen testaan sen toimivuutta samalla kun teen muutoksia. Tämän kaltaisen testaamisen tarkoitus on virheiden löytäminen, mutta kaikkia virheitä ei itse aina tule huomanneeksi. Siksi on tärkeää testata sovellusta kehitystiimin ulkopuolisilla henkilöillä. Joskus annan sovelluksen testattavaksi designerillemme taikka toimitusjohtajallemme, koska he edustavat sovelluksen

käyttäjäryhmää. Kun toimitusjohtajamme äiti sattuu paikalle, pyrin myös saamaan häneltä kommentteja uusimpiin muutoksiin. Myös käyttäjäpalaute saattaa informoida sovelluksen kehityksen suuntaa.

Opinnäytetyön aikana työni raportoiminen on johtanut käyttökokemus suunnittelun sekä työtehtävieni osa-alueiden syvälliseen pohdintaan. Olen oppinut tämän seurauksena paljon työstämieni ominaisuuksien pohjalla olevaa teoriaa. Esimerkiksi olen saanut hyviä näkökulmia raportoinnin ja käyttämieni lähteiden kautta Scrumin mukaisiin työskentelytapoihin ja Androidin sisäisiin ääniongelmiin liittyen. Alan kirjallisuus ja asiantuntija-artikkelit ovat olleet hyödyksi käytännössä ja päätyneet syventämään tietouttani isojen konseptien osalta.

Lähteet

Talouselämä 11.11.2015. Uusi suomalaisyhtiö karaokemarkkinoille: Singa julkaisi sovelluksensa Slushissa. Luettavissa: http://www.talouselama.fi/kasvuyritykset/uusi-suomalaisyhtio-karaokemarkkinoille-singa-julkaisi-sovelluksensa-slushissa-6064020 Luettu: 2.2.2016

Zapata, BC 2013, Android Studio Application Development, Packt Publishing Ltd, Olton, Birmingham, GBR. Luettavissa: http://ezproxy.haaga-helia.fi:2077/lib/haagahelia/detail.action?docID=10783635&p00=android+studio Luettu: 3.5.2015

Meier, R 2012, Professional Android 4 Application Development, John Wiley & Sons Inc., Indianapolis, Indiana, US. Luettu: 4.12.2015

McAnlis, C 2015, Android Developers YouTube-kanava, video Double Layout Taxation [Android Performance Patterns Season 3 ep8] (100 Days of Google Dev), Julkaistu 1.9.2015. Katsottavissa: https://www.youtube.com/watch?v=dB3_vgS-Ugo Katsottu 2.2.2016.

Eklund, T 2016, Software engineer, Qvik Oy, Helsinki. Helsinki Android Meetup, ImageView + Url = How hard can it be? -esitys, 26.1.2016. Katsottavissa: https://github.com/ShionTatarian/ImageCompare

Peitek, N et al 2015, Picasso: Easy Image Loading on Android, Ruboss Technology Corp, Victoria, BC, Canada. Luettu: 26.12.2015

Peitek, N et al 2015, Glide: Customizable Image Loading on Android, Ruboss Technology Corp, Victoria, BC, Canada. Luettu: 26.1.2016

Zelenski J. et al 2008, The Happy-Productive Worker Thesis Revisited, Journal of Happiness Studies Volume 9, Issue 4, sivut 521-537. Luettavissa: http://link.springer.com/article/10.1007/s10902-008-9087-4 Luettu: 7.2.2016

Sutherland J. et al 2018, Suomenkielinen Scrum-opas, Lekman Consulting, Helsinki, Suomi, Luettavissa: https://lekman.fi/scrumguide/ Luettu: 29.3.2018

Takeuchi H. et al 1986, New New Product Development Game, Harvard Business Review. Luettavissa: https://cb.hbsp.harvard.edu/cbmp/product/86116-PDF-ENG Luettu: 10.2.2018

Sutherland J. et al 1995, Business object design and implementation, OOPSLA ’95 workshop proceedings, The University of Michigan, sivu 118. Luettu: 10.2.2018

Nokia Siemens Networks 2009, The impact of latency on application performance. Luettavissa: https://web.archive.org/web/20130801103723/http://www.nokiasiemensnetworks.com/system/files/document/LatencyWhitepaper.pdf Luettu: 12.3.2018

Vlaskovits P. et al 2015, Android Audio's 10 Millisecond Problem: The Android Audio Path Latency Explainer. Luettavissa: http://superpowered.com/androidaudiopathlatency Luettu: 25.4.2018

Android Developers Documentation, AudioRecord. Luettavissa: https://developer.android.com/reference/android/media/AudioRecord Päivitetty viimeksi: 17.4.2018. Luettu: 26.4.2018

McAnlis, C 2016, Android Developers YouTube-kanava, video Understanding Android Threading (Android Performance Patterns Season 5 ep2). Julkaistu 1.3.2016. Katsottavissa: https://www.youtube.com/watch?v=dB3_vgS-Ugo Katsottu 1.5.2018.

Android Developers Documentation, Processes and threads overview. Luettavissa: https://developer.android.com/reference/android/media/AudioRecord Päivitetty viimeksi: 23.4.2018. Luettu: 1.5.2018

Göransson, A 2014, Efficient Android Threading, O’Reilly Media, Sebastopol, CA, United States. Luettu: 1.5.2018

Anderson D 2010, Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press, Sequim, Washington, United States. Luettu: 20.12.2017

Björkholm T 2015, Kanban in 30 days, Packt Publishing Ltd, Olton, Birmingham, GBR. Luettu: 3.5.2015

Kniberg H 2009, Kanban and Scrum – Making the Most of Both, C4Media Inc., United States. Luettu: 28.4.2018

Westby EJH 2015, Git for Teams, O'Reilly Media, Inc. Sebastopol, CA, United States. Luettu: 26.4.2018

Driessen, V 05.01.2010, A successful Git branching model, Luettavissa: http://nvie.com/posts/a-successful-git-branching-model/ Luettu: 30.4.2018

Santacroce, F 2017, Git Essentials, Packt Publishing Ltd, Olton, Birmingham, GBR. Luettu: 3.5.2015