OULUN AMMATTIKORKEAKOULU

Henri Mäkelä

ANDROID-KEHITTÄJÄN PÄIVÄKIRJA

Ohjelmistoalan yrityksessä

Henri Mäkelä

ANDROID-KEHITTÄJÄN PÄIVÄKIRJA

Ohjelmistoalan yrityksessä

Henri Mäkelä
Opinnäytetyö
Kevät 2019
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

Tekijä: Henri Mäkelä
Opinnäytetyön nimi: Android-kehittäjän päiväkirja ohjelmistoalan yrityksessä
Työn ohjaaja: Ritva Virkkala
Työn valmistumislukukausi ja -vuosi: Kevät 2019 Sivumäärä: 59

Tämä päiväkirjamallinen opinnäytetyö kuvaa työtehtäviäni kymmenen viikon ajalta oululaisessa ohjelmistoalan yrityksessä. Työtehtävät ovat suurimmaksi osaksi Android-mobiilikehitystä asiakasprojektissa sekä aluksi yrityksen sisäisessä projektissa asiakasprojektin alkamista odotellessa. Koska kyseessä on asiakasprojekti, tarkempia yksityiskohtia sovelluksesta ja sen toiminnallisuuksista ei voitu jakaa.

Työn tavoitteena on kuvata, millaista on työskennellä Android-sovelluskehityksen parissa ja antaa yleiskuva Android-sovellusten rakenteesta sekä yleisimmistä suunnittelumalleista. Viikkoanalyysit kattavat teoriaa, joka on hyödyllistä aloittelevalle tai jo hieman kokemusta omaavalle Android-kehittäjälle.

Tietoperustana on käytetty suurimmaksi osaksi Googlen virallista Android-kehitys -dokumentaatiota sen ollessa luotettavin ja ajantasaisin lähde. Joissakin kohdissa on käytetty virallisen dokumentaation lisäksi blogikirjoituksia lähteinä. Ohjelmointiongelmien ratkaisemissa käytin myös ahkerasti StackOverflow-sivustoa, joka on verraton apu ongelmatilanteiden ratkaisemisessa.

Kymmenen viikon päiväkirjaraportoinnin ja viikkoanalyysien lopputuloksena saatiin kokonaiskuva Android-kehityksen pääpiirteistä, vaikkakin työn kuvan ollessa vanhan koodin refaktorointia ja ongelmien korjaamista ei tämä työ ole varsinaisesti ohje Android-sovelluksen luomiseen.

Asiasanat: Ohjelmointi, Android-sovelluskehitys, Java

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems, Web Application Development

---

Author: Henri Mäkelä
Title of thesis: A diary-based thesis of android developer
Supervisor(s): Ritva. Virkkala
Term and year when the thesis was submitted: Spring 2019 Number of pages: 59

This diary-style thesis was created to describe my work as an android developer on a software company located at Oulu, Finland. The tasks during these ten weeks are mostly android development for a customer's application and at first for company's internal application while waiting for the customer project to start. Because the work is for customer's application, some details about the application cannot be shared.

The idea of this thesis was to describe what it is like to work as an android developer and give a good overview about the android application structure and most common design patterns. The weekly analysis covers useful theory about android development for a beginner or an intermediate developer.

The background of the android development theory is mostly from the official android developer site by Google since it is the most trusted and most up-to-date source of information. Also, some blog posts are referenced for example in the design pattern part of the thesis.

The outcome of this work is a pretty good overview of android development principles and what it is like to work as a developer. Although the work itself was mostly refactoring old code and fixing bugs so this document is not exactly a guide on how to build an android application.

Keywords: Software development, Android development, Java

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on työtehtävieni kuvaileminen ja analysoiminen sovelluskehittäjänä ohjelmistoalan asiantuntijapalveluyrityksessä Oulussa kevättalvella 2019. Opinnäytetyössä käydään aluksi läpi lähtötilanne, eli mitä taitoja omasin jo ennestään, jotka mahdollisesti auttavat työssäni, sekä esitellään yritys ja työympäristö. Kirjoitin päiväkirjaa kymmenen viikon ajalta aikavälillä 22.1.2019 – 28.4.2019. Päiväkirja sisältää päivittäiset työtehtäväni, sekä viikoittaiset viikkoanalyysit, joissa käsitellään sovelluskehityksen teoriaa, mitä olen oppinut, sekä mitä ongelmia olen kohdannut.

Työn aihe sopi hyvin omaan tilanteeseeni aloittelevana sovelluskehittäjänä asiakasprojektissa. Kuvaamalla päivittäisiä työtehtäviäni lukijat, esimerkiksi toiset samassa vaiheessa olevat opiskelijat saavat käsityksen siitä, mitä sovelluskehitys ohjelmistoalan yrityksessä pitää sisällään. Tämän lisäksi Android-sovelluskehityksestä kiinnostuneet henkilöt saavat yleiskuvan Android-kehitysympäristöstä.

1.1 Käsitteet

Android Studio - Kehitysympäristö Android-sovellusten luomiseen.

API - Application Programming Interface. Mahdollistaa kanssakäymisen toisen sovelluksen tai sovelluksen osan kanssa.

APK - Android Application Package - Tiedosto, johon koko sovellus on pakattu sen asentamista varten älypuhelimelle.

Backend – Sovelluksen ”takaosa”, jota käyttäjä ei näe ja jossa esimerkiksi tietokantatoiminnot tapahtuvat.

Bugi - Virhe ohjelman lähdekoodissa.

Client – Sovelluksen osa, jota käyttäjä käyttää.

Debuggaus - Virheen syyn etsiminen lähdekoodista.

Git – versionhallintatyökalu.

Gradle - Koodin käännöstyökalu.

JNI (Java Native Interface) - Rajapinta natiivikoodin ja Javan välille.

Kotlin – Googlen kehittämä ohjelmointikieli Android-sovellusten luomiseen.

NodeJS – JavaScriptiä, joka pyörii palvelimella selaimen sijasta.

Refaktorointi - Sovelluksen koodin uudelleenjärjestämistä ja kirjoittamista paremman arkkitehtuurin ja ylläpidettävyyden vuoksi.

Repositorio - Paikka, jossa koodia säilytetään versionhallinnassa.

REST-rajapinta – sovellusrajapinta, joka tarjoilee ja vastaanottaa dataa JSON-formaatissa.

UI - User Interface eli käyttöliittymä.

UX - User Experience eli käyttäjäkokemus.

VPN - Virtual Private Network.

1.2 Yrityksen ja työympäristön esittely

Yritys on vuonna 2006 perustettu ohjelmistoalan konsultointiin keskittyvä yritys, jossa on noin 30 työntekijää. Yrityksen työntekijöistä suurin osa on sovelluskehittäjiä ja -arkkitehtejä. Projektitiimissäni on minun lisäkseni kolme muuta henkilöä, joilla on pitkä tausta mobiilisovelluksien ja -ratkaisujen kehittämisestä. Tämä helpottaa työntekoani, kun osaavaa apua on saatavilla.

Sidosryhmäni työpaikalla sisältävät projektitiimini lisäksi myyjät sekä liiketoiminnan kehittäjät ja esimieheni. Ulkoisiin sidosryhmiin kuuluu asiakas.

Projektitiimissäni Android-vastaavana on työkaverini, jonka kanssa päätämme, missä järjestyksessä teemme asioita. Kommunikointi on helppoa meidän ollessa samassa tilassa päivittäin sekä päivittäisten pikaraporttien ansiosta, jotka olemme automatisoineet Slack-botille. Asiakkaan kontaktihenkilöt ovat meidän kanssamme jaetussa Slack-keskustelussa, jonka kautta kommunikointi asiakkaan päähän on vaivatonta. Tämän lisäksi järjestämme joka toisella viikolla noin puoli tuntia kestävän palaverin asiakkaan kanssa toimistollamme.

2 ALKUTILANNE

2.1 Omat taidot

Olin kesän 2018 harjoittelussa kyseisessä yrityksessä, jolloin tein mobiilisovelluskehitystä Androidille. Harjoittelun projektina minulla oli yrityksen oman sovelluksen kehittäminen Androidille. Sovelluksessa päädyttiin käyttämään Unity-pelimoottoria 3D-ympäristön luomiseen, joten se toi lisähaasteita työhön. Sovimme harjoittelun alussa, että ohjelmointikieli, jolla sovellusta lähdetään luomaan Androidille, on Kotlin perinteisemmän Javan sijaan.

Harjoitteluni onnistui hyvin ja opin paljon uusia asioita, kuten tiimityöskentelyä, versionhallintaa, sovellusarkkitehtuuria ja Kotlin-ohjelmointikieltä. Suurimmat haasteet harjoittelussani olivat Unityn 3D-näkymän upottaminen Android-ympäristöön sekä Unityn ja Androidin välinen kommunikointi.

2.2 Projektin kuvaus

Aloitin työni neljä viikkoa ennen tämän opinnäytetyön kirjoituksen aloittamista. En voinut aloittaa opinnäytetyön kirjoitusta heti, koska minulla oli jäljellä vielä neljä viikkoa harjoittelua, joka piti suorittaa loppuun ennen opinnäytetyön aloittamista.

Työsuhteeni alkaessa minulle ei heti löytynyt sopivaa asiakasprojektia, joten työkseni muodostui yrityksen sisäisen mobiilisovellusprojektin aloittaminen ja kehittäminen siihen asti, kunnes asiakasprojekti alkaa. Yrityksen sisäiset projektit ovat tärkeysjärjestyksessä alempana kuin asiakkaille tehtävät palvelut, joten niitä tehdään vain silloin, kun odotellaan uusien asiakasprojektien alkamista.

Tämä mobiilisovellus tiivistettynä sisältää REST-rajapinnan, joka tarjoilee dataa tietokannasta mobiili-clientille, ja esittää dataa android-widgetissä. Aloitin clientin kehittämisen Androidista, koska se oli jo entuudestaan tuttua minulle ja työvälineeni soveltuivat paremmin siihen (Windows-kone ja Android-älypuhelin).

Aloitin projektin kehittämisen backendistä. Toive oli, että se rakennettaisiin käyttäen Serverless Frameworkia, jotta se voitaisiin ottaa käyttöön FaaS-palveluna (function as a service). Näitä palveluita tarjoaa muun muassa AWS, Microsoft Azure ja Google Cloud. Backendissä käytettyjä teknologioita olivat NodeJs, Serverless Framework ja MongoDB.

Kun olin saanut backend-prototyypin toimivaksi, siirryin Androidin puolelle. Sain clientin kohtalaisen nopeasti siihen vaiheeseen, että se pystyi kommunikoimaan kehittämäni backendin kanssa. Ainoa uusi asia Androidin puolelta minulle oli widgetien kehittäminen. Widget on osa sovellusta, joka voidaan kiinnittää kotinäkymään, josta se näyttää esimerkiksi tiivistettynä tärkeimmät tiedot sovelluksesta. Tässä sovelluksessa widgetin ideana oli päivittyä sopivin väliajoin ja tarkistaa, onko käyttäjän dataan tullut muutoksia.

Päiväkirjaraportoinnin alkaessa teen siis yhä tätä sisäistä projektia. Olen luonut backend-prototyypin ja clientin Androidille siihen vaiheeseen, että käyttäjä voi kirjautua sisään, tarkastella omaa dataansa ja lisätä widgetin puhelimen kotinäkymään, joka päivittyy automaattisesti tunnin välein tai käyttäjän toimesta.

Työni todennäköisesti muuttuu lähiaikoina asiakasprojektiin, jolloin ylläkuvaamani projekti siirtyy odottelemaan seuraavaa tekijää, jolla on aikaa tehdä sitä.

3 PÄIVÄKIRJARAPORTOINTI

3.1 Viikko 1

Maanantai 18.2.2019

Viikko alkoi suunnittelemalla, mitä sovelluksen ominaisuuksia on parannettavissa tai lisättävissä annetuilla tiedoilla. Tämä tuo ajoittain ongelmia kehitykseen, koska sovelluksella ei ole tarkkoja kehyksiä, eikä määritelmiä millainen sen tulisi olla, ainoastaan pääpiirteet.

Päätin, että alkuviikon työni on ottaa selvää eri tavoista, joilla JWT:n (autentikaatio-token) voi tallentaa käyttäjän puhelimeen, jotta se ei tuhoudu joka kerta, kun sovellus lopetetaan, eikä käyttäjän tarvitse joka kerta kirjautua sisään uudestaan.

Päädyin käyttämään Androidin Shared Preferences -ominaisuutta, joka mahdollistaa tiedon tallentamisen puhelimen muistiin avain-arvoparina. Tätä käytettäessä asetuksen tulee olla yksityisenä, jolloin ainoastaan kyseinen sovellus pääsee siihen käsiksi. Suuri osa päivästä meni kyseisen asian opiskeluun, mutta sain kuitenkin implementoitua sen jo osittain sovellukseen. Tehtäväksi jäi vielä tarkastaa, löytyykö tokeni puhelimesta sovelluksen käynnistyessä, jolloin käyttäjä ohjataan suoraan dashboard-näkymään, eikä kirjautumisnäkymään.

Tiistai 19.2.2019

Jatkoin autentikaatio-tokenin tallentamisen kanssa. Sovelluksen aloitusnäkymänä on kirjautumisnäkymä. Lisäsin koodin, joka tarkistaa, että onko puhelimen muistissa jo valmiina token. Jos on, niin käyttäjä ohjataan suoraan dashboard-näkymään ja jos ei ole, niin käyttäjää pyydetään kirjautumaan sisään. Mikäli kirjautuminen onnistuu, palvelin palauttaa käyttäjälle tokenin, joka sitten tallennetaan muistiin.

Tällä hetkellä backendissä ei ole määritelty tokenin vanhenemisaikaa, vaan se on validi loputtomiin. Tietoturvan kannalta tokenin vanhenemisaika kannattaa määritellä ja suorittaa sen päivittäminen refresh tokenilla. Tällöin, mikäli kolmas taho saa autentikointi-tokenin haltuunsa, se ei ole toimiva loputtomiin.

Keskiviikko 20.2.2019

Keskityin tänään sovelluksessa widgetin päivittämiseen. Widgetin luokka perii AppWidgetProvider-luokan, joka tarjoaa metodit widgetin alustamiseen, päivittämiseen, poistamiseen ja päivityspyynnön vastaanottamiseen. Nämä metodit voidaan ylikirjoittaa Widget-luokassa, jolloin voidaan hallita yllämainittuja toimintoja. Toistaiseksi sovelluksessamme riittää onUpdate-metodin ylikirjoitus. Kun widget ensimmäisen kerran lisätään aloitusnäyttöön, päivitysmetodia kutsutaan kerran. Sen jälkeen päivitysmetodia kutsutaan joko manuaalisesti käyttäjän toimesta tai viimeistään, kun ennalta määritelty aika on kulunut. Pienin mahdollinen päivitysväli on 30 minuuttia.

Lisäsin päivitysmetodiin ehtolauseen, joka hakee käyttäjän datan widgetiin, jos autentikointi-token löytyy muistista. Jos tokenia ei löydy, widgetin teksti huomauttaa, että käyttäjän tulee kirjautua sisään ja widgetiä klikkaamalla käyttäjä viedään kirjautumisnäkymään.

Torstai 21.2.2019

Eilen saimme tiedon, että asiakasprojekti, johon minut on liitetty, on alkamaisillaan. Työ on asiakkaan mobiilisovelluksen ohjelmointivirheiden korjaamista sekä uusien toimintojen lisäämistä. Minun työni on tehdä sovelluksen Android-versiota. Työni alkaa sillä, että käyn läpi valmiiksi kirjoitetun lähdekoodin sekä sisäistän sovelluksen rakenteen, jonka jälkeen alan korjata asiakkaan ilmoittamia ongelmia.

Emme päässeet aloittamaan työtä vielä tänään, koska asiakas ei ole toimittanut vielä tunnuksia, joilla pääsisimme heidän versionhallintaansa. Tein viimeisiä muutoksia aiemmin työstämääni projektiin, kommentoin kirjoittamaani koodia ja puskin sen Git:iin seuraavaa tekijää varten.

Perjantai 22.2.2019

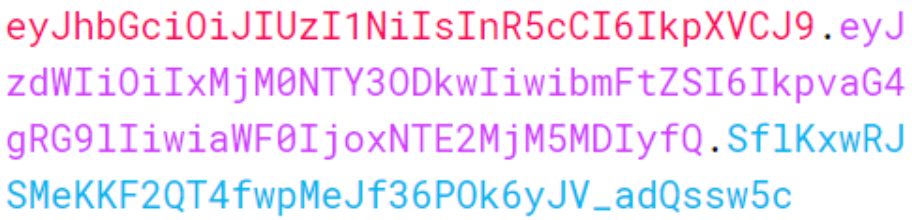
Perjantai meni muuttaessa projektitiimimme työympäristöä alempaan kerrokseen ja järjestellessä työpistettä. Katsoin myös iltapäivästä pari Android-tutoriaalia

Viikkoanalyysi

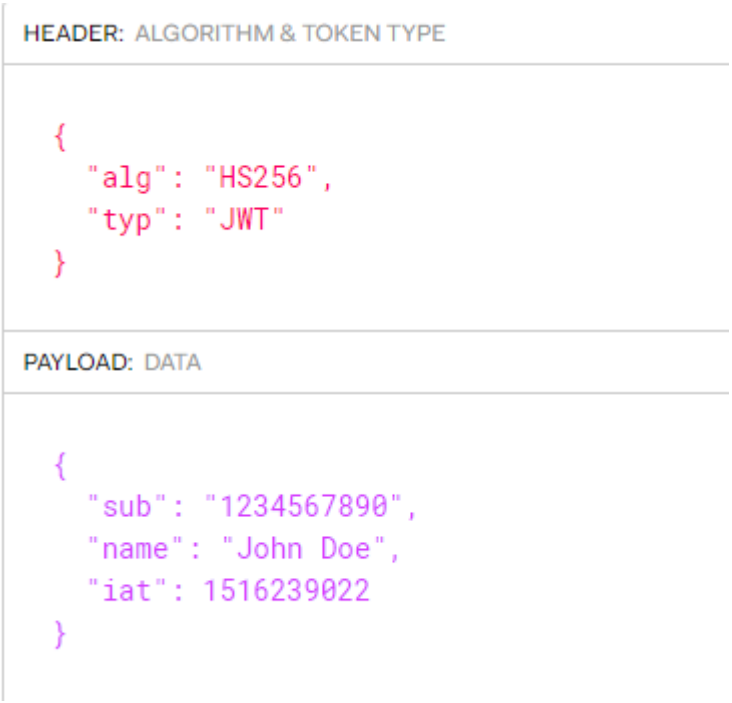
Viikon työt liittyivät pääsääntöisesti vielä yrityksen sisäisen sovelluksen kehittämiseen. Uutena asiana opettelin android widgetin ohjelmointia sekä autentikaatio-tokenin tallentamista mobiilisovelluksessa.

Käyttäjän autentikointi palvelimen kanssa kommunikointiin tässä tapauksessa toimi JWT:n (JSON Web Token) avulla. Tämä toimii siis niin, että rekisteröitynyt käyttäjä lähettää HTTP-pyynnön palvelimelle, jossa on mukana tietoa käyttäjästä. Tässä tapauksessa sähköpostiosoite ja salasana lähetetään. Jos salasana ja sähköpostiosoite vastaavat rekisteröitynyttä käyttäjää, palvelin lähettää takaisin tokenin, jolla autentikointia vaativat pyynnöt voidaan hoitaa. Toiminnassa olevan palvelimen tulee käyttää HTTPS-yhteyttä, jolloin palvelimen ja käyttäjän välinen yhteys on salattua, eikä salasanaa ja sähköpostiosoitetta lähetetä tekstimuodossa.

JWT on avoin standardi, joka määrittelee kompaktin ja omavaraisen keinon kuljettaa dataa osapuolten välillä turvallisesti JSON-formaatissa. Data on luotettavaa, koska se voidaan digitaalisesti allekirjoittaa. (Jwt.io, viitattu 23.2.2019.)



KUVIO 1. JWT kryptattuna (Jwt.io 2019)



KUVIO 2. JWT purettuna (Jwt.io 2019)

3.2 Viikko 2

Maanantai 25.2.2019

Maanantai vaikutti toiveikkaalta sen suhteen, että pääsisimme vihdoin käsiksi asiakkaan git-repositorioihin ja voisimme aloittaa lähdekoodin tutkimisen ja tehtävälistan läpikäymisen. Saimmekin käyttäjätunnukset asiakkaan palveluihin, mutta tarvitsemamme repositoriot eivät olleet näkyvissä meille, joten emme päässeet aloittamaan vieläkään.

Olen opetellut käyttämään Googlen Android Architecture Components -kirjastoa Android-sovelluskehityksessä parantamaan sovelluksen rakennetta. Koska emme päässeet vielä töihin, jatkoin tämän aiheen parissa.

Tiistai 26.2.2019

Pääsimme tänään käsiksi asiakasprojektimme koodiin. Kloonasin projektin Android-version Gitlabista ja aloitin koodin tutkimisen. Sovellus on rakennettu alun perin alihankkijan toimesta asiakkaallemme ja käsittääkseni sitä on muokannut jo useampi taho. Koska sovellus on jo useamman vuoden vanha, se on kirjoitettu Javalla, eikä nykyisin suositulla Kotlinilla. Sovellus käyttää kirjastoja, jotka on kirjoitettu C++- ja C -ohjelmointikielillä ja niitä käytetään Androidista Native Development Kitillä (NDK). Näitä natiivikoodilla kirjoitettuja funktioita ja luokkia kutsutaan Java Native Interfacen (JNI) kautta.

Sovelluksen kääntäminen osoittautui ongelmaksi. Projektin Git-repositoriossa on kaksi alimoduulia, jotka pitää kääntää ensiksi NDK-build-työkalulla ennen varsinaisen projektin kääntämistä. Toistaiseksi selvittämättömästä syystä toinen alimoduuli ei kloonaudu alihakemistoon vaan se jää tyhjäksi. En edennyt asian kanssa tänään, joten päätimme katsoa sitä huomenna työkaverini kanssa

Keskiviikko 27.2.2019

Saimme alimoduulit kloonautumaan ja projektin kääntymään. Alkuperäinen sovellus oli ilmeisesti toteutettu Eclipsellä, joten oikea tapa olikin importata se Android Studioon sen sijaan, että koettaisi avata sen suoraan Android Studiossa projektina.

Alimoduulista puuttui yksi tärkeä osa, jota ilman sovellus ei toimi ja asiakas ei ollut tätä toimittanut meille, joten aamupäivä meni koodia lukiessa ja odotellessa. Iltapäivällä, kun olimme saaneet kaikki projektin osat alas versionhallinnasta, saimme sovelluksen kääntymään ja ajettua emulaattorissa.

Torstai 28.2.2019

Emme ole saaneet vielä oikeuksia Jira-tehtävienhallintaohjelmistoon, jotta tietäisimme, mitä asioita sovellukseen tulee tehdä. Testailin sovellusta ja sen komponentteja ja tein muistiinpanoja.

Android Studioon ei ole lisätty vasta kuin hiljattain Storyboard-tyylinen näkymä, Navigation Controller, joka helpottaa sovelluksen rakenteen hahmottamista ja sovelluksessa navigoinnin suunnittelua. Tässä vuosia sitten tehdyssä projektissa ei tietenkään ole sellaista käytetty, joten sovelluksen rakenteen hahmottaminen on hitaampaa.

Suuri osa sovelluksessa käytetyistä kirjastoista on deprekoitunut eli vanhentunut. Nämä kirjastot toimivat yhä, mutta deprekoituneiden osien kanssa voi tulla ongelmia julkaistaessa Google Play-kauppaan. Koska emme tiedä vielä tarkkoja tehtäviämme, selvinnee myöhemmin, tuleeko meidän päivittää kaikki vanhentuneet osat.

Perjantai 1.3.2019

Tämä päivä meni suurelta osalta samalla kaavalla kuin eilinenkin. Testailin sovellusta parilla eri Android-puhelimella ja huomasin, että jostain syystä sovelluksen kirjautuminen ei onnistunut Nokia 6.1 älypuhelimella, kun se käännettiin Android Studiosta minun Windows-käyttöjärjestelmäisellä koneella. Macilta käännettäessä se toimi myös Nokialla. Omituisen tästä ongelmasta tekee se, että kun käänsin sovelluksen omalta koneeltani Honor 8:lle, se toimi täysin odotetulla tavalla.

Ensi viikko on hiihtolomaviikko ja suurin osa työkavereistani jää lomalle. En tiedä vielä, mitä ensi viikon työtehtäväni tulevat pitämään sisällä, jos kaikki ovat lomalla, eikä asiakas ole järjestänyt minulle oikeuksia tehtävienhallintaan.

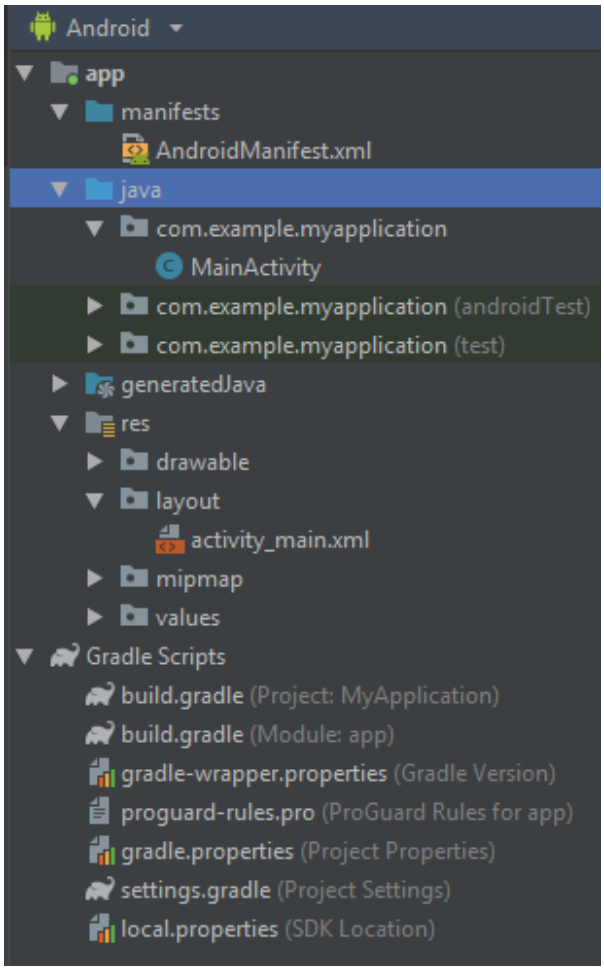
Viikkoanalyysi

Viikon työtehtävät olivat vielä toistaiseksi hieman hajanaisia ja sekalaisia. Viikko meni suurimmaksi osaksi asiakkaan sovelluksen koodia tutkiessa ja valmistautumisessa tulevaan työhön. Olen ehtinyt käydä sovelluksen rakennetta läpi huolellisesti ja uskon, että tämä tulee auttamaan itse kehitys-työssä.

Android-projektin rakenne saattaa vaihdella projektikohtaisesti, mutta perusrakenne on yleensä sama. Projekti koostuu moduuleista, kirjastoista, lähdekooditiedostoista, resurssitiedostoista sekä Android Manifestista. Manifest-tiedosto kuvaa olennaiset tiedot sovelluksestasi käännöstyökalulle, Android-käyttöjärjestelmälle sekä Google Playlle (Android Developers 2019, viitattu 2.3.2019).

Tässä viikkoanalyysissä käydään läpi Android Studiossa luodun uuden, tyhjän projektin rakennetta. Oletuksena Android Studio näyttää projektirakenteen Android-näkymänä. Tämä näkymä ei vastaa suoranaisesti tiedostohistoriaa projektikansiossa, vaan on järjestelty moduulien ja tiedostotyyppien

mukaan helpottamaan projektissa navigointia piilottaen vähemmän käytetyt tiedostot. (Android Developers 2019, viitattu 2.3.2019.)



KUVIO 3. projektirakenne Android-näkymänä

manifest

Sisältää projektiin vaadittavan AndroidManifest.xml-tiedoston.

java

Sisältää hakemiston projektin pakettinimellä, jossa sijaitsevat lähdekooditiedostot, kuten luokat, activityt ja fragmentit.

res

Sisältää resurssitiedostot, kuten näkymien xml-tiedostot, väritiedostot, kuvat, ikonit ja tekstiresurssit.

Gradle Scripts

Sisältää projektin käännösskriptit, mikäli projektissa käytetään Gradle-käännöstyökalua. Android Studiossa projektia luotaessa tämä on oletuksena.

3.3 Viikko 3

Maanantai 4.3.2019

Sovimme perjantaina töistä lähtiessämme, että sillä aikaa, kun muut tiimistämme ovat lomalla, alan muuttamaan projektia Gradlen alaiseksi projektiksi. Gradle on käännöstyökalu (Build tool), jota käytetään Android Studiossa. Projektin kääntäminen on tällä hetkellä moniosainen prosessi ja Gradlella se voidaan automatisoida niin, että kaikki projektin osat saadaan käännettyä kerralla.

Alkuperäinen projektin kääntäminen tapahtuu niin, että ensiksi C++-moduulit käännetään NDK-build -työkalulla, sen luomat shared library (.so) -tiedostot kopioidaan päämoduulin kansioon, josta ohjelma osaa löytää luokkatiedostot, kun niiden funktioita kutsutaan Java Native Interfacen kautta. Kun koko projekti on käännetty, siitä luodaan APK-tiedosto, joka sisältää kaiken, mitä sovellus tarvitsee, jotta se voidaan ajaa älypuhelimessa, mukaan lukien C++ .so -kirjastot. Tämän viikon viikkoanalyysissä tulen käymään läpi, kuinka C- ja C++-koodia voidaan käyttää Android-projektissa, sekä millä tavalla Gradle hoitaa Android-sovellusten kääntämisen.

Aloitin päivän työt lukemalla Googlen kirjoittamaa ohjetta developer.android -sivustolla ”Migrate to Android Studio”. Eclipse projektin muuttaminen Android Studion Gradle-projektiksi oli automatisoitu prosessi, jonka Android Studio hoiti. Tuontiprosessi lisää Eclipse-projektin kirjasto- ja projektiriippuvuudet (dependencies) build.gradle -tiedostoon.

Sovellusta käännettäessä Gradle etsii projektista build.gradle -tiedoston, jonka mukaan se kääntää sovelluksen. Näitä tiedostoja voi olla useampi riippuen siitä, onko projektissa useampi moduuli. Sain projektin jo osittain muutettua kääntymään Gradlella niin, että päämoduuli kääntyy ja sovellusta voi ajaa, mutta tietyt osat eivät toimi vielä, koska en ehtinyt tuoda C++ -moduuleja vielä uuteen versioon.

Tiistai 5.3.2018

Aamu alkoi lukemalla artikkeleita ja esimerkkejä siitä, millä tavalla C++/C-projekteja voidaan yhdistää Gradleen ja Android-projektiin. Koska toinen moduuleista oli pienempi ja vaikutti yksinkertaisemmalta ymmärtää, päätin aloittaa tuomalla sen projektiin.

Aloitin valitsemalla Android Studiossa File -> New -> Import module. Tämän jälkeen valitaan hakemisto, josta moduuli tuodaan ja Android Studio lisää sen moduuliksi pääprojektin alle. Gradle luo tuodulle moduulille automaattisesti oman build.gradle -tiedoston.

Päämoduulin build.gradle -tiedostoon lisätään riippuvuudeksi tämä tuotu moduuli, jotta projektia ajettaessa se otetaan mukaan. Tiedoston dependency-osio näyttää tältä

```
dependencies {
   implementation project(':moduleX')
```

Gradle ei kuitenkaan vielä osaa kääntää C++-tiedostoja kirjastoiksi, vaan tämä pitää erikseen määritellä moduulin build.gradle -tiedostossa. Tämä onnistuu klikkaamalla hiiren oikealla painikkeella haluttua moduulia ja valitsemalla ”Link C++ Project with Gradle”. Tämän jälkeen valitaan kääntäjäksi joko CMake tai NDK-build. Valitsin NDK-buildin ja lisäsin tiedostopolun osoittamaan Android.mk makefileä. Tästä asiasta löytyy tarkempaa tietoa tulevassa viikkoanalyysissä. Kun projektin yhdistäminen on suoritettu, build.gradle tiedostoon lisätään osio

```
externalNativeBuild {
   ndkBuild {
      path file('src/main/jni/Android.mk')
   }
}
```

Tämä kertoo NDK-build -työkalulle, mistä löytää kääntämiseen tarvittava makefile.

Päivä meni pitkälti säätämisessä ja asioiden opettelemisessa. Toinen moduuleista on nyt linkitetty Gradlella kääntyvään versioon, mutta kääntämisvaiheessa tapahtuu jokin virhe, joka johtuu C++-koodista. Huomisen tavoitteena on paikantaa tuo virhe ja korjata se.

Keskiviikko 6.3.2018

Päivän tavoitteena oli saada eilen lisäämäni moduuli kääntymään ja toimimaan yhdessä sovelluksen kanssa ilman virheitä. Koska minulla ei ole aiempaa kokemusta C++:n kanssa ja kaikki työkaverini ovat lomilla, aamupäivä meni lukiessa ja opetellessa perusasioita C++-syntaksista ja siitä, miten makefilet toimivat.

Paikansin ongelmakohdan ja sain korjattua sen. Ongelma oli tietotyyppimuunnoksessa C++ -funktiossa, joka aiheutti virhetilanteen kääntämisen aikana, eikä kirjastoa lisätty apk-tiedostoon. Siitä johtui UnsatisfiedLinkError, kun JNI-funktioita kutsuva luokka koetti ladata kirjastoa ajon aikana.

Päivä oli onnistunut, koska sain ensimmäisen kahdesta moduulista toimimaan sovelluksen kanssa. Huomenna siirryn toisen moduulin pariin, joka vaikuttaa monimutkaisemmalta kuin aikaisemman osan kääntäminen. Jos asia ei etene huomenna ja juoksen niin sanotusti seinään, lopetan työt tältä viikolta ja hoidamme tehtävän maanantaina loppuun, kun työkaverini palaavat lomalta, koska uskon, että tämä asia on hoidettavissa yllättävänkin helposti, kunhan tietää mitä tekee.

Torstai 7.3.2019

Päivä oli lyhyt ja tulokseton. Aloitin työt lisäämällä toisen moduulin projektiin samalla tavalla kuin olin lisännyt aiemman. Mutta sitä käännettäessä tapahtuu virhe, jonka virheilmoitus on jokseenkin epäselvä eikä kerro esimerkiksi, missä tiedostossa ja rivillä virhe tapahtuu. Koin, että olin taistellut näiden asioiden kanssa jo aivan tarpeeksi tälle viikolle ja koska en päässyt etenemään, pakkasin reppuni ja lähdin viikonlopun viettoon.

Perjantai 8.3.2019

Ei töitä

Viikkoanalyysi

Viikko oli puuduttava, koska paljoa etenemistä ei tapahtunut. Olin kyllä osannut odottaakin tätä jo viime viikolla. Siksi viikkoanalyysiinkään ei hirveänä ole asiaa analysoitavaksi. Onnistumisia kuitenkin oli sen verran, että sain 2/3 projektista toimimaan Gradlella.

Opin viikon aikana ymmärtämään C++ -perusteita ja sitä, miten C++ -projektia ja Android-projektia voidaan käyttää yhdessä. Viimeinen ongelma, joka tuli toista moduulia linkittäessä, ei johtunut Androidin ja C++:n välisestä yhteydestä, vaan virhe on jossain syvemmällä C++ -koodissa.

Myöskään asiakkaasta ei kuulunut koko viikon aikana mitään, mutta se ei kyllä haitannutkaan, koska olemme tulleet siihen päätökseen, että tämä projekti pitää ensiksi saada kääntymään Gradlella ja deprekoituneet riippuvuudet päivitettyä, ennen kuin tehtävälistaa aletaan käymään läpi.

Koska tähän mennessä tätä päiväkirjaa kirjoittaessani työtehtävät ovat edelleen olleet sekalaisia, myös viikkoanalyysin aiheet vaihtelevat paljon. Ensimmäisellä viikolla kirjoitin tunnistautumisen hoitamisesta web-pavelimelle JWT:llä, mutta nyt kun työtehtäväni ovat muuttuneet, siirryn kertomaan enemmän Android-kehitykseen liittyvistä asioista.

Native Development Toolkit (NDK) on työkalu, joka mahdollistaa C ja C++ -koodin käyttämisen Android-projektissa. NDK:ta ei toisi monessakaan tapauksessa lisäarvoa sovellukselle ja suurin osa sovelluksista voidaankin kirjoittaa pelkästään esimerkiksi Javalla tai Kotlinilla, mutta joissakin tapauksissa se voi olla hyödyllinen. Android Developers -sivuston ohje antaa pari esimerkkiä, joissa tämän työkalun käyttö voi olla hyödyllistä:

- kun täytyy saada kaikki mahdollinen suorituskyky irti laitteesta viiveen pienentämiseksi tai kun täytyy ajaa laskennallisesti raskaita sovelluksia, kuten pelejä tai fysiikan simulointeja
- kun halutaan uudelleenkäyttää jo valmiiksi kirjoitettuja C tai C++ -kirjastoja.
  (Android Developers 2019, viitattu 9.4.2019.)

Alempi esimerkeistä on juurikin se, mihin me tarvitsemme kyseistä työkalua. NDK:lla voidaan kääntää C- ja C++-koodia natiivikirjastoksi ja pakata se APK-tiedostoon mukaan. Sen jälkeen Java tai Kotlin voi kutsua natiivifunktioita Java Native Interfacen läpi. Android-kehityksessä C ja C++ asioista käytetään useasti termiä ”native code” tai ”native library”, joten jos tässä kirjoituksessa jatkossa puhutaan natiivista, se tarkoittaa C ja C++ -asioita. Jos halutaan lisätä natiivitoiminnallisuutta Android-projektiin, vaiheet menevät yksinkertaistettuna näin:

Luodaan jni-hakemisto projektin juureen, johon natiivitiedostot sijoitetaan.

nativetest.c -tiedostossa oleva funktio, jota kutsutaan JNI:n läpi

```
#include <jni.h>
#include "includes/nativetest.h"

JNIEXPORT jstring JNICALL Java_com_henrimakela_ndktest_MainActivity_getMessage(JNIEnv* env, jobject object){
    return (*env)->NewStringUTF(env, "Bello?! This message comes from JNI!");
};
```

Android.mk eli makefile, joka hoitaa tiedostojen pakkaamisen natiivikirjastoksi

```
LOCAL_PATH:=$(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE:=nativetestjni
LOCAL_SRC_FILES:=nativetest.c

include $(BUILD_SHARED_LIBRARY)
```

Tässä esimerkissä en lisännyt ndk-build -komentoa build.gradle tiedostoon, vaan ajoin sen command linelta, jolloin jniLibs-hakemisto ja natiivikirjastot luodaan eri prosessorimalleille.

Nyt kirjasto voidaan ladata Javasta tai tässä tapauksessa Kotlinista käsin ja kutsua sen funktioita.

```
init {
    System.loadLibrary( libname: "nativetestjni")
}
external fun getMessage(): String

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    textView.text = getMessage()
}
```

Tämä natiivifunktio palauttaa vain String-muuttujana tekstin, joka laitetaan Kotlinin puolella textView-näkymän tekstiksi.

3.4 Viikko 4

Maanantai 11.3.2019

Työkaverini palasivat tänään lomalta, mikä oli hyvä asia, koska saimme tuoreita silmäpareja katsomaan Android-koodia. Etenin jälleen hieman eteenpäin projektin implementoinnissa Gradleen ja sain toisen moduulin kirjastot kääntymään.

Vielä kuitenkin tuli virhetilanne, jossa build-skripti ei löytänyt muutamaa C++ -header-tiedostoa. Epäilin, että ongelma johtuu Windowsista, joka ei ymmärtänyt polkuja oikein. Tein versionhallintaan uuden haaran ja puskin projektin sinne, josta työkaverini ottivat sen alas ja koettivat ajaa sitä Macilla. Kokeilu jäi keskeneräiseksi, koska työpäivä loppui kesken. Huomenna asiakkaamme tulee toimistollemme aloituspalaveriin. Uskon, että palaverin jälkeen työt saadaan virallisesti alulle.

Tiistai 12.3.2019

Päivä alkoi pikaisella viidentoista minuutin seisomapalaverilla asiakkaamme kanssa toimistollamme. Sovimme, miten palaverit tullaan jatkossa järjestämään sekä muita projektin kulkuun liittyviä asioita. Painotimme sitä, että tarvitsemme oikeudet Jiraan mahdollisimman nopeasti ja asia hoituikin tunnin sisällä tapaamisesta.

Päätimme työkaverini kanssa, että hän jatkaa projektin Android-ympäristön päivittämistä nykytasolle, sekä koettaa korjata viimeisiä ongelmia projektin yhdistämisessä yhdeksi kokonaisuudeksi. Minä siirryin tutkimaan, mitä bugeja minun on mahdollista alkaa jo korjailemaan.

Vaikka sovelluksen koko toiminnallisuus ei tällä hetkellä ole käytössä yllämainittujen ongelmien takia, suuri osa tehtävänhallintajärjestelmässä olevista tehtävistä olivat käyttöliittymään liittyviä ongelmia, jotka eivät ole riippuvaisia toisten moduulien toiminnasta.

Keskiviikko 13.3.2019

Jatkoin käyttöliittymäbugien korjaamista. Useassa näkymässä ongelmana oli se, että älypuhelimen näytölle ilmestyvä näppäimistö peittää osan näkymästä, tässä tapauksessa painikkeen sekä toisen tekstikentän.

Androidissa tämän voi korjata muuttamalla AndroidManifest.xml -tiedostossa windowSoftInputMode-attribuutin arvoa, sekä lisäämällä näkymään vieritystoiminto, mikäli käyttöliittymäkomponentteja on näkymässä paljon.

Nyt kun tekstikenttään kohdistetaan ja näppäimistö ilmestyy näytölle, näkymää nostetaan ylöspäin ja jos sen taakse jää edelleen osa näytön alaosassa olevista kentistä, ne voidaan vierittää esille, eikä käyttäjä joudu sulkemaan ensin näppäimistöä.

Torstai 14.3.2019

Päivä ei eronnut merkittävällä tavalla edellisestä päiväkirjaraportoinnin kannalta. Korjailin lisää käyttöliittymäongelmia ja tein muutoksia tekstin lokalisointiin. Iltapäivällä puskin tekemäni muutok-

set repositorioon ja tein yhdistämispyynnön repositorion ”master”, eli päähaaraan yhdistämistä varten, mutta meidän tiimillämme ei ollut oikeuksia valtuuttaa sitä, joten se jäi odottelemaan asiakkaan hyväksyntää.

Sovelluksen kääntäminen master-haarasta apk-tiedostoksi tapahtuu Jenkinsillä. Jenkins on automatisointityökalu, jolla voidaan muun muassa automatisoida projektin kääntäminen aina, kun Git-repositorion master-haaraan tulee uutta koodia.

Perjantai 15.3.2019

Olin korjannut nyt Jirassa listatut UI (käyttöliittymä) -ongelmat, jotka olivat tällä hetkellä mahdollista ja järkevää korjata. Aloitin aamun auttamalla työkaveriani, joka työskentelee tällä hetkellä sovelluksen toisen moduulin kanssa, debuggaamaan sovelluksesta ne kohdat, joissa moduulin natiivikirjaston metodeja käytetään ja kohdat, joissa ongelmat syntyvät.

Etenimme asian parissa aamupäivän, jonka jälkeen siirryin takaisin Jiran läpikäymiseen ja bugien korjaamiseen. Aloin korjata ongelmaa, jossa asiakas halusi turvallisuuden kannalta, että käyttäjää pyydetään tunnistautumaan aina, kun sovellus palaa taustalta takaisin, eli jos käyttäjä on painanut home-painiketta, eikä pelkästään silloin, kun sovellus on suljettu. Sain ongelman ratkaistua lisäämällä Androidin activity-luokan onStop() -elinkaarimetodiin (lifecycle method) koodin, joka katkaisee muodostetut yhteydet. Tätä samaa tapaa käytettiin jo aiemmin, mutta ainoastaan onDestroy() -metodissa, jota kutsutaan vain silloin, kun sovellus suljetaan. En kuitenkaan lisännyt koodia vielä repositorioon, koska halusin varmistaa työkaveriltani, onko tämä järkevä tapa ratkaista asia.

Viikkoanalyysi

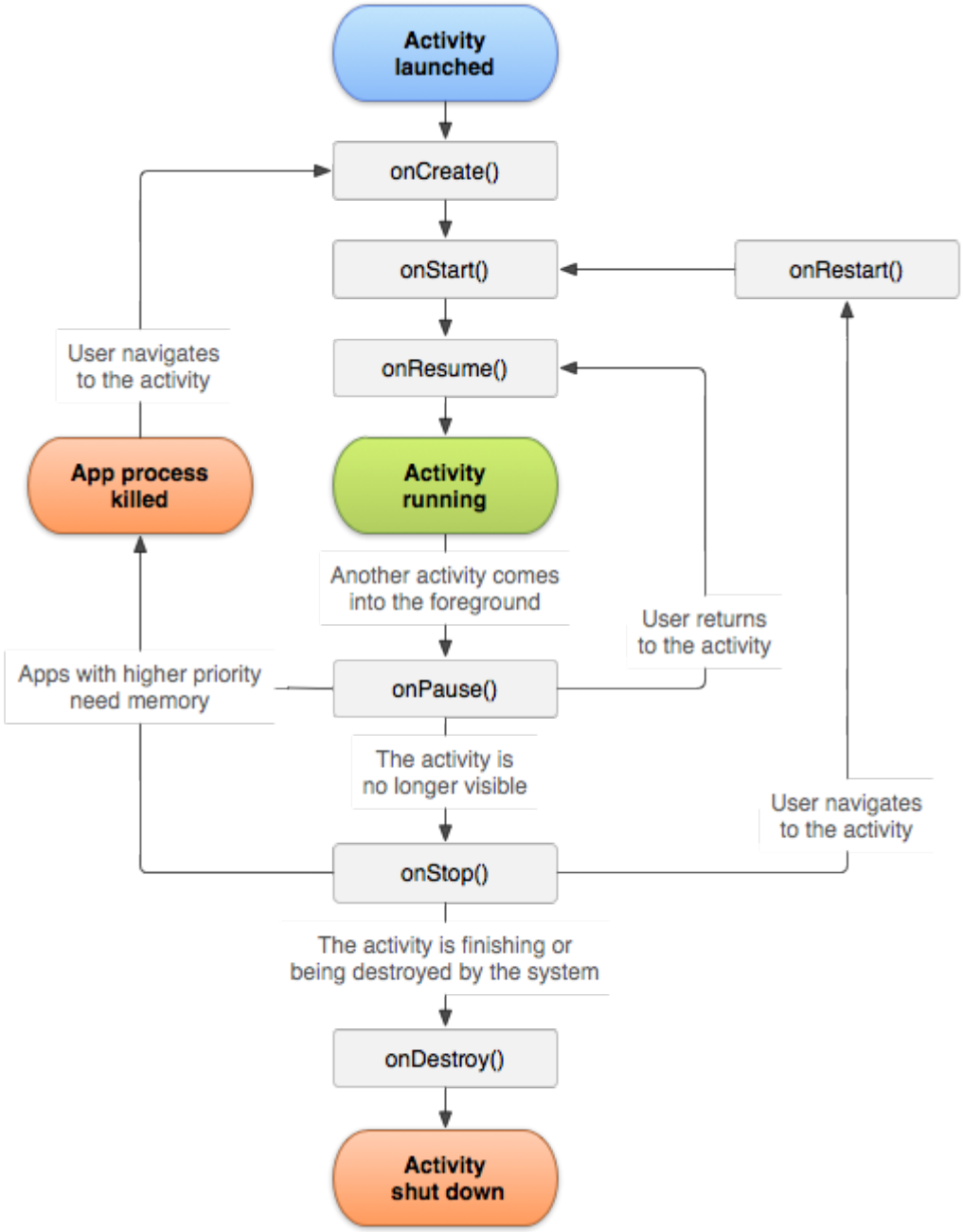
Viikko oli tuloksien kannalta edellistä viikkoa parempi. Sain korjailtua paljon käyttöliittymään liittyviä ongelmia, sekä yhden koodissa esiintyvän ongelman. Uusia asioita ei tullut tällä viikolla, koska Androidin käyttöliittymäasiat olivat minulle jo tuttuja.

Jouduin kuitenkin palauttelemaan mieleeni asioita Android-sovelluksen elinkaaresta ja tilasta eri vaiheissa. Käyn tässä viikkoanalyysissä läpi, millä tavalla Android-sovelluksen tilaa voidaan hallita sen tarjoamilla elinkaarimetodeilla.

Android-sovellus koostuu Activityistä. Activityt huolehtivat näkymien piirtämisestä ruudulle, klikkaustapahtumista, käyttäjän syötteestä, datan esittämisestä näkymässä, sekä muista käyttöliittymätapahtumista.

Activityyn voidaan kirjoittaa metodeja yhtä lailla kuin kaikkiin muihinkin Java- tai Kotlin-luokkiin. Esimerkiksi Activityssä voisi olla metodi datan hakemiseen HTTP-pyynnöllä palvelimelta, mutta tämä ei kuitenkaan ole suositeltavaa eikä arkkitehtuurin kannalta hyvä asia, koska se vaikeuttaa sovelluksen testattavuutta, modulaarisuutta sekä skaalautuvuutta. Paras tapa on pitää Activity mahdollisimman ”tyhmänä” ja kirjoittaa sovelluksen logiikka sovelluksen datakerrokseen.

Activity luodaan sovelluksessa tekemällä aliluokka Activity-luokasta, jolloin se perii Activity-luokan elinkaarimetodit.



KUVIO 4. Android Activityn elinkaari (Developers Android 2019, viitattu 16.3.2019).

Activity-luokka tarjoaa kuviossa 1. näkyvät kuusi elinkaarimetodia: onCreate, onStart, onResume, onPause, onStop ja onDestroy. Järjestelmä kutsuu näitä callback-metodeja sen mukaan, kun activity siirtyy uuteen tilaan. Activityssä ei ole välttämätöntä implementoida jokaista perittävää elinkaarimetodia. Tämä riippuu täysin activitysi toiminnallisuudesta, mutta onCreate-metodi on implementoitava, koska se on ensimmäinen metodi, jota kutsutaan activityä luotaessa ja on vastuussa näkymän rakentamisesta. (Developers Android 2019, viitattu 16.3.2019.)

3.5 Viikko 5

Maanantai 18.3.2019

Maanantaiaamu alkoi selailemalla Jiraa ja päättämällä, mitä bugeja alan seuraavaksi korjaamaan. Löysin vielä muutaman yksinkertaiselta vaikuttavan käyttöliittymäongelman, mutta niiden korjaamisessa menikin koko päivä.

Sovelluksessa on käytetty samaa näkymää moneen eri asiaan, kuten uuden salasanan asettamiseen, salasanan vaihtamiseen sekä kirjautumiseen. Näkymästä poistetaan kenttiä näkyvistä tilanteen mukaan. Siksi oli aluksi vaikea paikantaa, missä virheet esiintyvät, ennen kuin huomasin tämän toteutuksen. Päivä oli onnistunut ja pääsin tavoitteisiin, jotka asetin itselleni aamulla.

Tiistai 19.3.2019

Koska kahden viikon yrittämisen jälkeen emme ole edelleenkään saaneet projektia kokonaan käännettyä gradle-ympäristöön, päätimme toistaiseksi siirtyä toiseen tapaan. Moduulit kääntyvät Jenkinsissä ongelmitta, joten kopioimme Jenkinsissä olevista tiedoistoista ”libs”- ja ”assets”-kansiot, joissa moduulien tarvitsemat natiivikirjastot ja tiedostot ovat ja kopioimme ne omiin projekteihimme. Näin meillä on ainakin täysin kääntyvä projekti kehitystyötä varten. Nyt kun teemme committeja repositorioon, Jenkins kääntää uuden apk-tiedoston automaattisesti.

Siirryin korjaamaan ongelmia, joissa tietyissä tilanteissa sovellus kaatuu. Debuggaamisen jälkeen löysin syyksi null pointer exceptionin ja sain korjattua sen.

Keskiviikko 20.3.2019

Aamupäivä meni tapaamisessa yrityksen uuden mahdollisen yhteistyökumppanin kanssa. Keskipäivän aikoihin tulin takaisin toimistolle ja aloin käydä läpi bugilistaa.

Sovellus käyttää VPN-yhteyksiä, ja eräs ongelmista oli, että VPN-tunneli ei sulkeutunut, kun sovellus suljettiin. Debuggailun jälkeen löysin oikean metodin, jota pitää kutsua pää-activityn onDestroy-elämäkaarimetodissa ja sain korjattua ongelman.

Loppupäivästä tein dokumentaatiota UI/UX -suunnittelijallemme sovelluksen näkymistä ja flowsta, koska hän liittyy mukaan projektiin suunnittelemaan asiakkaallemme parannuksia käyttöliittymään ja käyttäjäkokemukseen.

Torstai 21.3.2019

Päivän työtehtävät olivat hajanaisia. Tutkin aamupäivän, millä tavoin saamme allekirjoitettua uuden sovelluskauppaan menevän apk-tiedoston eri salausalgoritmilla, koska nykyinen apk-tiedosto on allekirjoitettu SHA1:llä, joka ei ole enää turvallinen mahdollisten törmäysten takia.

En saanut ratkaistua tätä asiaa vielä ja tulin siihen tulokseen, että se ei tällä hetkellä ole prioriteettilistan kärjessä. Selvitin myös toista tietoturvaan liittyvää ongelmaa, jossa meidän pitäisi ensiksi pystyä tarkastamaan onko laite, josta sovellusta ajetaan root-tilassa ja estää sovelluksen käynnistyminen, jos näin on.

Päivän lopuksi keräilin UX-suunnittelijallemme sovelluksessa käytettäviä ikoneita ja logoja. Vaikuttaa siltä, että koko käyttöliittymä suunnitellaan täysin uusiksi, joten todennäköisesti tulevaisuudessa minun työkseni tulee uudistaa Androidin käyttöliittymä suunnitelman mukaiseksi.

Perjantai 22.3.2019

Päivä meni tutkiessa ja analysoidessa loppuja Android-bugeja Jirasta. Oikeastaan mitään konkreettista ei tullut tänään tehtyä. Aika meni asioiden opiskelussa ja tutkiessa, millä tavalla näitä ongelmia voisi debugata. Suuri osa näistä viimeisistä ongelmista on harvoin toistuvia yhteyksiin liittyviä ongelmia, ja muutama niistä pitäisi debugata palvelimen päästä, eikä mobiiliclientistä.

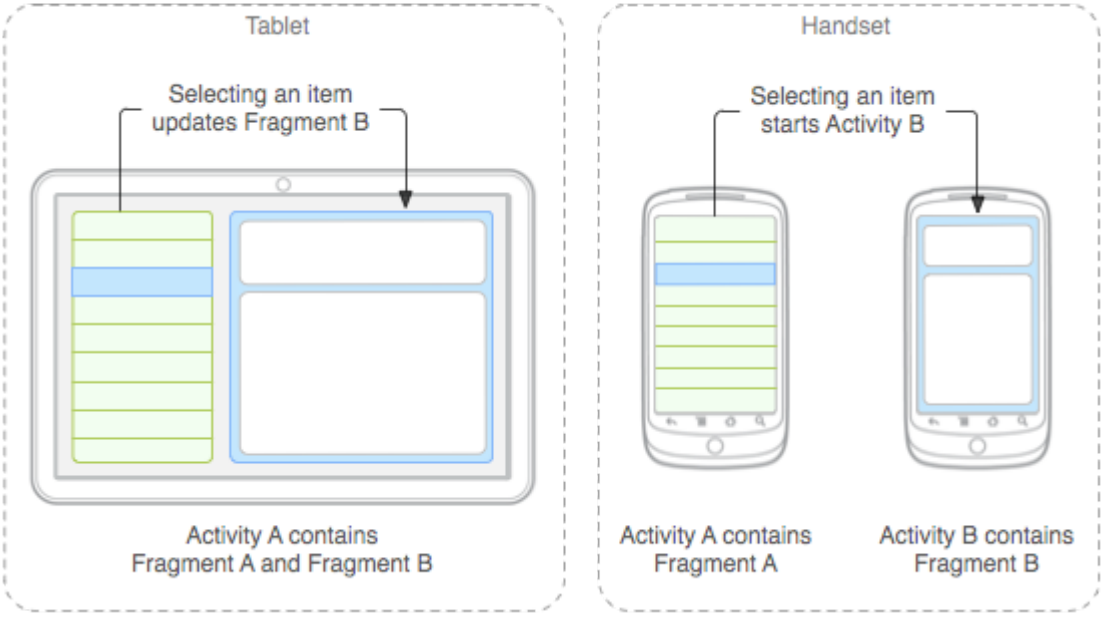
Viikkoanalyysi

Viikko meni pitkälti opetellessa ja lukiessa uusista asioista joihin törmäilin. Uutena asiana opin ymmärtämään jollakin tasolla, kuinka VPN-protokollat ja VPN-tunnelit toimivat. Suurin osa bugeista on korjattu, mutta muutama monimutkaisen oloinen ongelma vielä esiintyy. Ensi viikolla, jos asiakas hyväksyy UI/UX-suunnittelijamme piirrokset, alan mahdollisesti muuttamaan sovelluksen ulkoasua niiden mukaisiksi.

Koska viime viikon viikkoanalyysissä puhuin activityistä, tämän viikon analyysissä kerron toisesta tärkeästä elementistä Android-sovelluksen käyttöliittymässä, fragmentista. Fragmentti on modulaarinen osa activityä, jolla on oma elinkaarensa, joka huolehtii itse oman syötteen vastaanotosta ja jonka voi poistaa tai lisätä ajon aikana, kun activity on elossa. Fragmenttia voidaan ajatella ikään kuin ali-activitynä. (Android Developers 2019, viitattu 22.3.2019.)

Fragmentteja ei voida kuitenkaan käyttää samalla tavoin kuin activityjä, sillä fragmentti ei voi olla oma kokonaisuus, vaan se on aina activityn lapsi. Fragmenttia isännöivän activityn elinkaari vaikuttaa suoraan fragmentin elinkaareen. Jos activity tuhotaan, myös fragmentti tuhotaan.

Etu käytettäessä fragmentteja käyttöliittymää rakentaessa on niiden tarjoama mahdollisuus paloitella näkymä modulaarisiin, uudelleenkäytettäviin osiin. Nämä osat helpottavat käyttöliittymän suunnittelua eri kokoisille laitteille. Tabletin näytölle voidaan sijoittaa esimerkiksi kaksi fragmenttia vierekkäin ja älypuhelimen näytölle päällekkäin, jolloin toinen fragmenteista tuodaan esille käyttäjän vuorovaikutuksen mukaan.



KUVIO 5. Fragmenttien sijoittaminen erikokoisille näytöille (Developers Android 2019, viitattu 22.3.2019).

3.6 Viikko 6

Maanantai 25.3.2019

Viikko alkoi keskustelemalla, mitä kukin meistä tekee tällä hetkellä ja mitä tehdään seuraavaksi. En löytänyt bugilistalta enää mitään ongelmaa, joka olisi tällä hetkellä mahdollista tehdä, joten päivä meni suurimmaksi osaksi opiskellessa tarkemmin Android Studion UI-editoria ja millä tavalla Androidissa voidaan tehdä animaatioita, koska UX-suunnittelijamme uudessa designissa niitä tulee olemaan jonkin verran. Iltapäivästä autoin työkaveriani debuggaamaan Gradlessa esiintyviä ongelmia projektin Gradle-haarassa.

Tiistai 26.3.2019

Saimme Gradlen toimimaan, mikä mahdollistaa nyt helpommin sovelluksen vanhentuneiden osien päivittämisen. Ensimmäiseksi poistin paikallisena kirjastona olevan vanhan tukikirjaston, joka tarjoaa luokat vanhempien Android-versioiden tukemiseen ja lisäsin uuden version siitä Gradle-riippuvuutena. Tällä tavalla kirjastoa ei tarvitse sisällyttää paikallisesti projektiin.

Käytin Android Studion tarjoamaa analysointityökalua paikantamaan vanhentuneita osia koodista ja päivitin niitä uudempiin. Suuri osa näistä oli helppoja korjata, koska Android Studio antoi vihjeitä, mikä on uusi tapa vanhentuneen tilalle.

Päivän työt onnistuivat hyvin ja projektia saatiin vietyä jo paljon nykyaikaisen tasolle. Pari isompaa muutosta on kuitenkin vielä tehtävä, koska sovellus käyttää sormenjälkitunnistautumista, jonka tilalle on tullut Android API 28:ssa biometrinen tunnistautumispalvelu, joka kattaa kaikki nykyaikaiset biometriset tunnistautumiset. Toinen isompi päivitys tulee olemaan Androidin kamerakirjaston päivittäminen uudempaan versioon. Tämä voi olla haasteellista, koska sovellus käyttää QR-koodin lukuun kirjastoa, joka tukee vain tätä aiempaa kameraversiota.

Keskiviikko 27.3.2019

Aloitin työpäivän lukemalla dokumentaatiota Androidin uudesta Biometric Prompt API:sta. Käytin aamupäivän katselemalla esimerkkejä ja tekemällä muistiinpanoja. Lounaan jälkeen aloin katselemaan sovelluksen nykyistä koodia sormenjälkitunnistautumiseen liittyen. Sormenjälkitunnistus on

tehty siten, että jos sormenjälki tunnistetaan, se ei päästä suoraan kirjautumaan käyttäjää sisälle, vaan hakee kryptatun salasanan Androidin SharedPreferenssistä ja antaa salasanan metodille, joka käsittelee sen.

Tein uuden haaran versionhallintaan tunnistautumispäivitystä varten ja implementoin uuden biometrisen tunnistautumisen projektiin, mutta en saanut toimimaan sitä täysin vielä. Ongelma ilmenee jossakin kohtaa salasanaa haettaessa.

Päivän työt etenivät siinä määrin, että sain uuden kirjaston jo osaksi toimimaan sovelluksessa. Huomenna jatkan asian parissa.

Torstai 28.3.2019

Päivän tavoite oli saada uusi biometrinen tunnistautumispalvelu implementoitua sovellukseen. Tutkimalla lisää sovelluksen aiempaa toteutusta sekä salasanan kryptausta ja dekryptausta, Sain uuden biometrisen tunnistautumisen toimimaan vanhan tilalla. Opin myös lisää sovelluksen logiikasta, koska huomasin tätä asiaa tutkiessani, että toteutuksessa tallennetaan salasana kahteen paikkaan, mikäli sormenjälkitunnistus on käytössä.

Jos käyttäjä on aktivoinut sormenjälkitunnistuksen uutta salasanaa rekisteröitäessä, salasana kryptataan ja tallennetaan Android Keystoreen sekä SharedPreferenceihin. Jos sormenjälkitunnistusta ei ole aktivoitu, salasana tallennetaan vain Keystoreen. Sitä, miksi kyseinen toiminnallisuus on toteutettu juuri näin en osaa sanoa. Kuten aiemmin mainitsin, sovelluksen rakenne on erittäin kehno, eikä se omaa juuri minkäänlaista suunnittelumallia tai arkkitehtuuria. Esimerkiksi pää-activityssä on noin 2000 riviä koodia ja se sisältää lukuisia eri toiminnallisuuksia. Tämä tekee koodin tulkitsemisesta ja ymmärtämisestä hidasta.

Huomasin hieman puutteita uudessa API:ssa, jotka tekevät aiemmin helpoista toiminnoista vaivalloisempia toteuttaa. Aiempi sormenjälkikirjasto tarjosi metodit tarkistaa, onko käytettävässä laitteessa sormenjälkitunnistukseen tarvittavaa sensoria ja onko laitteeseen tallennettu sormenjälkiä. Uudessa kirjastossa niitä ei ole, vaan ainoa tapa saada tieto siitä on AuthenticationCallback-rajapinnan ongelmatilanteessa kutsuttavan metodin parametrina tuleva virheviesti. Käyttämäni support

library on vasta alpha vaiheessa, joten nämä metodit mahdollisesti lisätään myöhemmin. Uusi toiminnallisuus on kuitenkin nyt vaihdettu vanhan tilalle yhteen git-haaraan, josta voimme ottaa käyttöön se tarvittaessa riippuen siitä, mihin päätökseen tulemme.

Perjantai 29.3.2019

Aamulla aloitin puhdistamaan sovellusta Lintin (työkalu Android Studiossa, joka analysoi koodia ja kertoo kehittäjälle varoituksista ja erroreista) löytämistä ongelmista, koska muuta tehtävää minulle ei löytynyt tällä hetkellä. Sain korjattua kohtalaisen paljon varoituksia, koska ne olivat pieniä ja Android Studio osasi ehdottaa näihin ratkaisua, joten useat ongelmat korjautuivat nappia painamalla.

Iltapäivällä autoin työkaveriani (lähinnä googlaamalla) säätämään projektin build.gradlea siten, että se osaa kopioida toisesta moduulista sen moduulin käännöksen jälkeen tuottamat .so-tiedostot ja assetit päämodulin kansioihin. Gradle tiedostot voidaan kirjottaa joko Groove- tai Kotlin -ohjelmointikielillä ja niihin voidaan tehdä taskeja (=metodeja). Itse taski oli helppo kirjoittaa, mutta meillä meni hetken aikaa tajuta, millä tavalla taskin voi suorittaa suoraan käännöksen aikana, eikä erikseen käskemällä taskia. Saimme sen kuitenkin toimimaan ja lähdin viettämään viikonloppua.

Viikkoanalyysi

Työviikkoon mukavaa vaihtelua toi se, että pääsin oikeastaan ensimmäistä kertaa tämän projektin aikana kirjoittamaan uutta koodia biometrisen tunnistautumispalvelun merkeissä. Sain onnistumisen tunteita siitä, kun sain uutta teknologiaa käyttävän kirjaston toimimaan vanhan lähdekoodin kanssa.

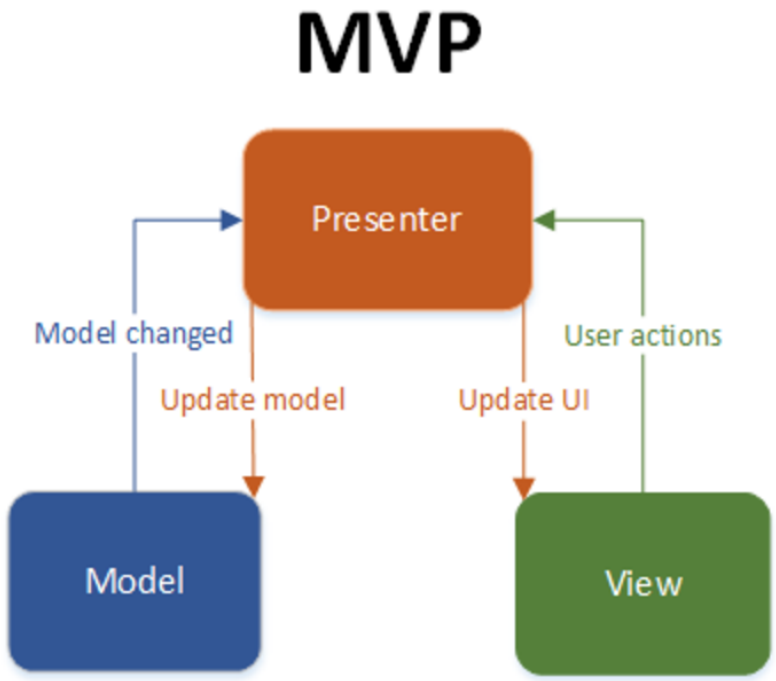
Vaikuttaa siltä, että asiakaskin on tietoinen siitä, että sovellus on rakennettu melkoisen monimutkaisella tavalla. He eivät siis käsittääkseni itse ole tehneet sitä, vaan sovelluksen tekemiseen on palkattu ulkopuolinen taho. UX-suunnittelijamme piirsi sovelluksesta täysin uuden suunnittelun ja asiakas oli tyytyväinen, joten ottaen huomioon tämän ja muutaman isomman uudistuksen, joka sovelluksessa saatetaan tehdä, voi olla että projektista tulee paljon isompi kuin aluksi oli tarkoitus. Tämä on tietysti hyvä asia meidän kannaltamme.

Koska mainitsin torstain päiväraportissani sovelluksen kehnosta rakenteesta, tämän viikon viikkoraportissa käsittelen sovellusarkkitehtuurin perusperiaatteita ja millä tavoin esimerkiksi Android-sovellus voidaan toteuttaa, jotta koodi on modulaarisempaa, testattavampaa ja skaalautuvampaa.

Suunnittelumalleja (design pattern) on useita erilaisia eikä yhtä tiettyä parasta ole. Eri yritykset ja kehittäjät käyttävät eri malleja. Yksi tärkeimmistä asioista suunnittelumallien käyttämisessä on kuitenkin sovelluksen logiikkakerroksen erottaminen näkymäkerroksesta. Android-kehityksessä suosituimpia malleja ovat MVP (Model-View-Presenter) ja MVVM (Model-View-ViewModel).

Tarkoituksena on tehdä näkymäkerroksesta, eli Androidissa activityistä ja fragmenteista mahdollisimman ”tyhmiä” ja passiivisia. Niiden ainoa tehtävä on esittää dataa ja hoitaa käyttäjän suorittamat interaktiot.

Erittäin yksinkertaistettuna: Model on logiikkakerros, joka hoitaa esimerkiksi tietokantatoiminnot ja http-pyynnöt, View esittää tarjolla olevan datan käyttäjälle ja hoitaa interaktiot ja Presenter tai ViewModel toimivat välikätenä Viewin ja Modelin välillä. MVP-arkkitehtuurissa käyttöliittymän päivitys tapahtuu presenteristä käsin kutsumalla parametrina sille annetun rajapinnan metodeja, jonka näkymä on implementoinut. Tällöin siis presenterille ei tarvitse antaa parametriksi käyttöliittymän päivitykseen activityä vaan ainoastaan sen rajapinta, jolloin presenter-kerros ei ns. tiedä näkymäkerroksesta. Myöskään logiikkakerroksen, eli Modelin ei ole tarkoitus tietää presenter-kerroksesta. Tätä tapaa kutsutaan termillä ”loose-coupling”. Löyhät sidokset sovelluksen osien välissä mahdollistavat sen, että osia voidaan muuttaa ilman, että se saastuttaa muita osia sovelluksesta.



KUVIO 6. MVP-arkkitehtuuri (Argawal. N. 2017).

3.7 Viikko 7

Maanantai 1.4.2019

Koko päivä meni taistellessa Windowsista aiheutuvien ongelmien kanssa sovelluksen käännösvaiheessa. Työkaverini sai perjantaina Gradle buildin hiottua kuntoon niin, että kustomoidut taskit ajetaan samaan aikaan käännöksen kanssa ja generoidut assetit ja kirjastot kopioidaan automaattisesti. Moduuli käyttää Cmake-työkalua, joka käyttää Swig-työkalua. Cmakessa on tiedosto, jonka funktiona on etsiä Swig tietokoneelta, mutta jostain syystä Windowsissa se ei löytänyt sitä yhtä helposti kuin Macissa, vaikka olin laittanut Swigin ympäristömuuttujiinkin. Sain kuitenkin ongelman korjattua, kun tarpeeksi yritin eri tapoja ja vaihdoin tiedostopolkua.

Tässä vaiheessa tämä pieni hidaste ei kuitenkaan haitannut niin paljon, koska minulla ei tällä hetkellä ole edes kiireellisiä asioita, mitä voisin tehdä projektiin liittyen.

Minulla ei ole mitään käsitystä, mistä tämä ongelma johtui. Olen kohdannut ennenkin Windowsin kanssa outoja ongelmia kehitystyötä tehdessäni. Päätimme tilata minullekin Macin, jotta kaikilla

tiimissä on samat työkalut, eikä aikaa tarvitse tuhlata tämän kaltaisiin ongelmiin. Positiivista päivässä oli se, että työsuhteeni jatko varmistui ja minulle on työpaikka täällä, kunhan valmistun koulusta.

Tiistai 2.4.2019

Aamusta iltapäivään hoidin useamman pienen tehtävän kehitysympäristöömme liittyen, jotka helpottavat työskentelyä. Iltapäivällä otin Jirasta tehtävän, jota olin lykkäillyt, koska se oli tärkeysjärjestyksessä matalalla ja uskoin olevan vaikea tuottaa uudelleen. Ongelma liittyi listanäkymän päivittymiseen, kun siihen lisätään uusi objekti. Nämä listassa näkyvät objektit ovat eräänlaisia laitteita, joita voidaan lisätä etänä sovellukseen. Jos käyttäjällä on paljon laitteita listassa ja hän on joutunut käyttämään vieritystoimintoa silloin, jos sovellukseen lisätään uusi laite, lista päivittyy ja rullautuu takaisin ylös, jolloin käyttäjä menettää silloisen sijaintinsa listalla.

Ongelma oli se, että meillä ei ole tarpeeksi laitteita X testaamaan tätä tilannetta, joten en voinut testata toimiko korjaukseni. Ilmoitin asiakkaalle asiasta päivittäisessä raportissani, ja he todennäköisesti toimittavat meille testauslaitteen.

Keskiviikko 3.4.2019

Aamupäivä sisälsi lisää projektin siistimistä. Poistin kaikki paikalliset kirjastot ja muutin ne Gradle-riippuvuuksiksi (dependency) sekä korjailin siistimisestä hajonneet osat.

Iltapäivällä aloin tutkimaan, kuinka saamme automatisoitua CI/CD (jatkuva integraatio/jatkuva toimitus) -työkalumme luomaan alpha-julkaisun tai sisäisen testin Google Play –kaupassa aina, kun olemme lisänneet uusia ominaisuuksia sovellukseen.

Tällä hetkellä projektissamme on kaksi CI-työkalua: toinen Jenkinsissä, joka kääntää vanhan, ei-gradle-version päähaaraa ja toinen Gitlabissa, jonka loimme itse kääntämään uuden version päähaaraa.

Torstai 4.4.2019

Korjailin lisää ei-niin-tärkeitä käyttöliittymäongelmia. Muun muassa ongelman, jossa laitelistaa selattaessa tabletilla animoitu nuoli-ikoni jää väärään asentoon, jos laitteen näytön kierto muuttuu.

Lounaan jälkeen työkaverini antoi minulle tehtäväksi parannella Gradle build -skriptiä siten, että ajettaessa clean-komento Android Studiossa, joka poistaa kaikki edellisessä käännöksessä tuotetut tiedostot, myös toisesta moduulista kopioidut JNI-tiedostot sekä assets-kansio poistetaan. Tämä hoitui kirjoittamalla pari poistotehtävää (task) ja linkittämällä nämä puhdistustehtävään tähän tyyliin: clean.doLast{ deleteTask1 deleteTask2 }

Olen kehittynyt tämän projektin aikana paljon Gradlen käyttämisessä oppimalla kirjoittamaan kustomoituja tehtäviä ja ymmärtämään tarkemmin, mitä Gradle suorittaa käännöksen aikana.

Perjantai 5.04.2019

Sain tänään uudeksi työkoneeksi MacBookin, joten päivä meni opetellessa käyttämään sitä ja asentaessa työhön tarvittavia työkaluja ja sovelluksia. Olen aikaisemmin käyttänyt pelkästään Windows-käyttöjärjestelmällisiä koneita, joten macOS-käyttöjärjestelmässä on hieman totuttelemista.

Iltapäivällä aloimme luomaan suljettua alpha-julkaisua uudesta versiosta, johon olemme korjanneet suurimman osan Jira:ssa luetelluista ongelmista. Suunnittelimme että tästä lähtien alkaisimme tekemään uuden suljetun julkaisun asiakkaallemme joka perjantai. Tämä ei kuitenkaan tänään onnistunut, koska APK-tiedoston täytyy olla allekirjoitettu salausavaimella, jotta se voidaan ladata Google Playhin. Avaimen voi luoda helposti Android Studiossa uutta julkaisua varten, mutta koska asiakkaalla on jo versio Google Playssa, seuraavien päivitysten APK-tiedostot tulee allekirjoittaa samalla avaimella. Ilmoitimme tästä asiakkaallemme iltapäivällä, mutta emme saaneet vastausta, joten se jää ensi viikon työksi.

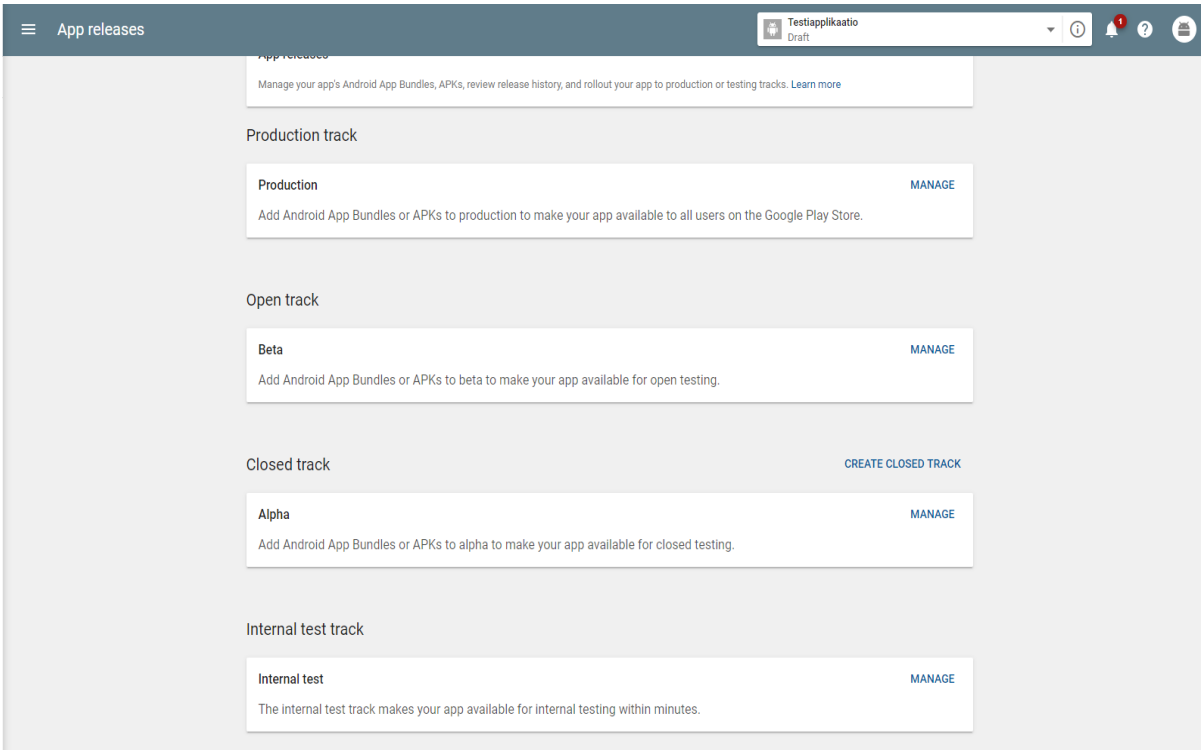
Viikkoanalyysi

Viikko oli samankaltainen aikaisempiin verrattuna. Projekti etenee omasta mielestäni kohtalaisen hitaasti erinäisistä asioista johtuen. Henkilökohtaista kehitystä on kuitenkin tapahtunut koko ajan. Tällä viikolla opin lisää Gradlen käyttöä ja käyttöliittymäasioita.

Sovelluksen julkaiseminen Google Playhin tapahtuu Google Play Consolen kautta. Consoleen luodaan käyttäjätili ja maksetaan 25 dollarin liittymismaksu, joka kattaa kaikki sovellusten julkaisut. 25 dollarin kertamaksu on siis ainoa, joka vaaditaan sovellusten julkaisemiseen.

Sovelluksen julkaisuprosessi sisältää kaksi päävaihetta: sovelluksen valmistelu julkaisua varten luomalla julkaisuversio APK-tiedostosta sekä sovelluksen julkaiseminen Google Play Consolesta asiakkaillesi. Tämän lisäksi prosessi sisältää muita tehtäviä, kuten yksityisen avaimen luomisen APK-tiedoston allekirjoitusta varten, sovellusikonin luomisen sekä mahdollisten käyttäjäehtojen luomisen. (Android Developers 2019, viitattu 6.4.2019

Sovelluksesta voidaan luoda eriasteisia julkaisuja: Sovelluksen sisäinen testi, alpha-versio, beta-versio ja tuotantoversio (kuvio 7).



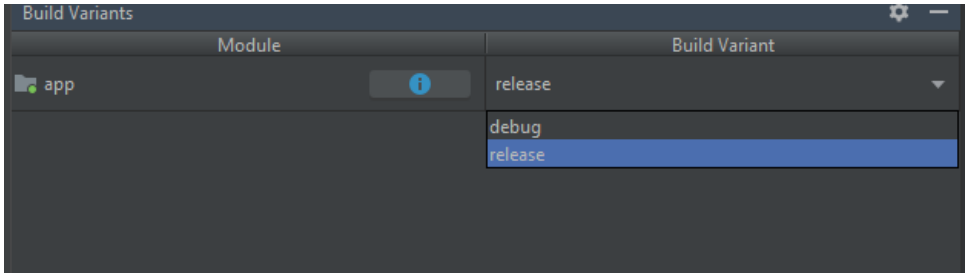
KUVIO 7. Julkaisuvaihtoehdot

Sisäinen testi on kätevä tapa saada sovellus nopeasti testattavaksi testiryhmälle. Testijulkaisuun lisätään sähköpostilista testaajista ja julkaisun jälkeen testaajat voivat ladata sovelluksen Google Playsta heille lähetetyn osoitteen kautta.

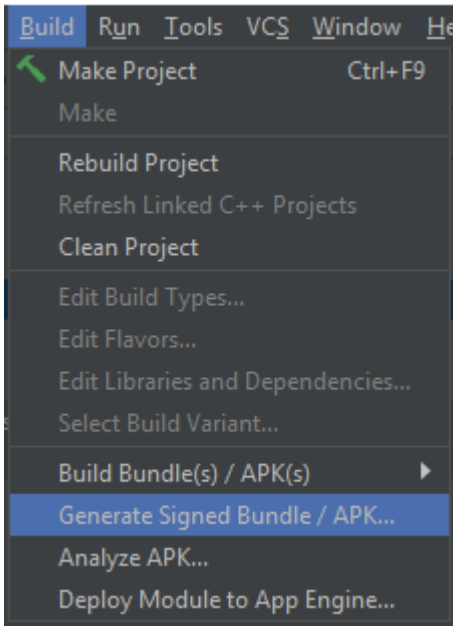
Kauppaan ladatun sovelluksen APK-tiedoston täytyy olla allekirjoitettu salausavaimella, jotta sen eheys ja luotettavuus voidaan tarkistaa. Allekirjoituksen voi lisätä joko itse Android Studiossa julkaisukäännöstä luotaessa tai antaa Google Playn hoitaa allekirjoittaminen. Antamalla Google Playn huolehtia sovelluksen allekirjoittamisesta, ei tarvitse itse huolehtia avaimien säilyttämisestä. Jokainen uusi versio sovelluksesta pitää allekirjoittaa samalla avaimella kuin aiempi.

Julkaisukelpoisen, allekirjoitetun APK-tiedoston luominen Android Studiossa

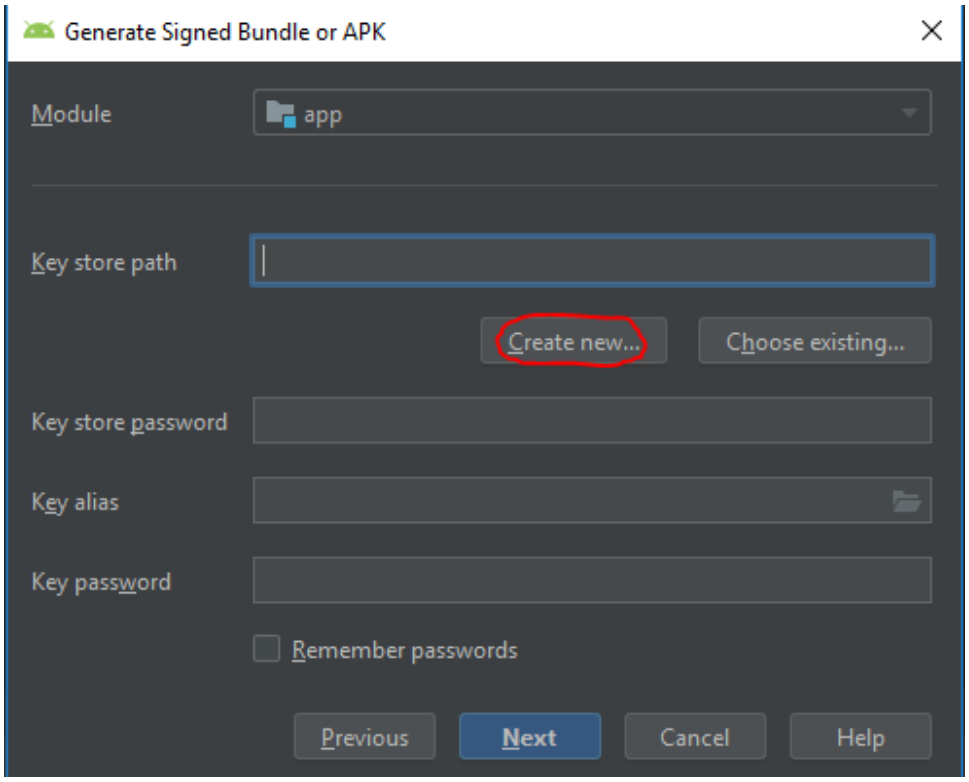
1. Muuta Build Variant julkaistavaksi versioksi



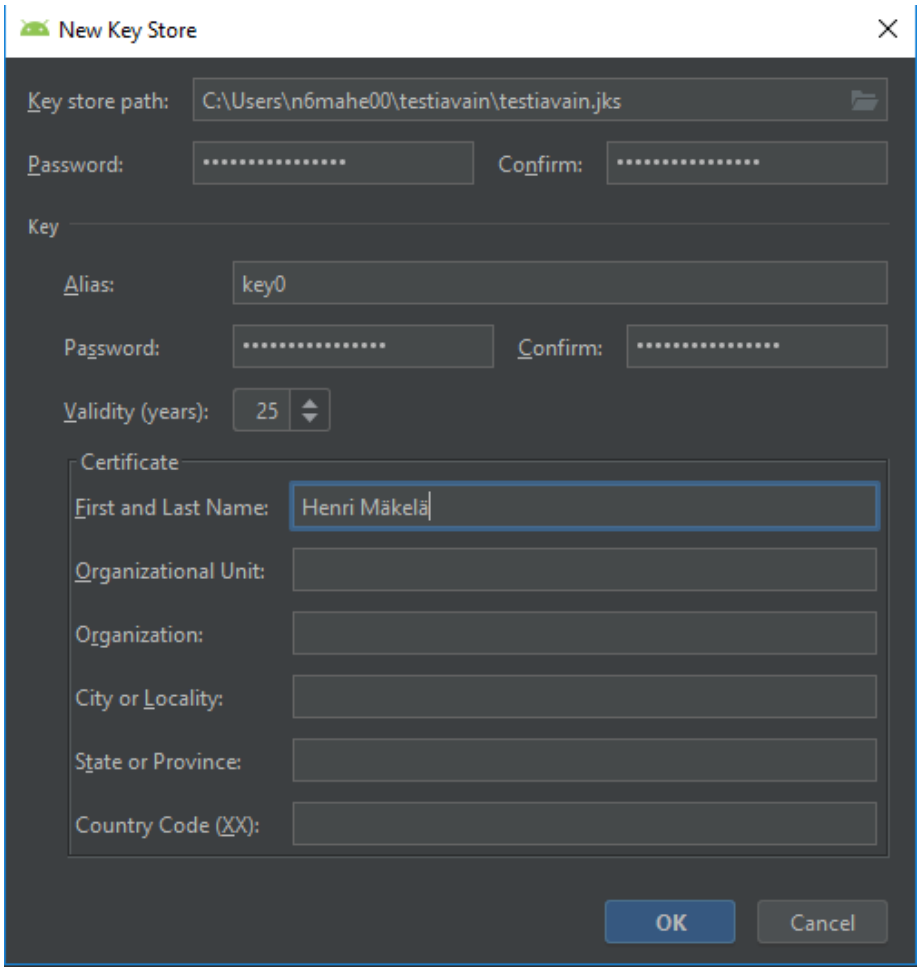
2. Valitse **Build -> Generate Signed Bundle / APK**



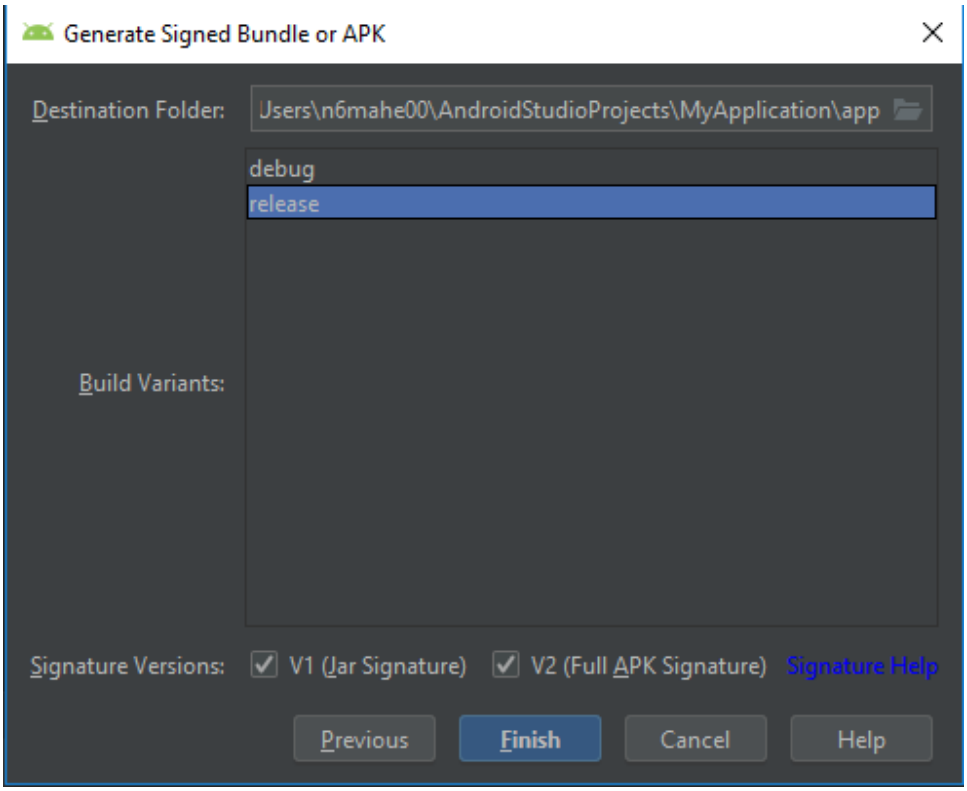
3. Valitse APK, jonka jälkeen valitaan avain allekirjoitusta varten. Voit vaihtoehtoisesti joko luoda uuden avaimen tai käyttää aiemmin luotua.



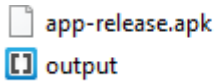
4. Anna avaimen tallennussijainti, salasana avainvarastolle, avaimen nimi, salasana avaimelle sekä täytä vähintään yksi kohta sertifikaattiin.



5. Anna käännösvariantiksi julkaisuversio ja laita molemmat allekirjoitusohjelmat päälle: V1 ja V2. V1 tukee laitteita, joissa on vanhempi API-taso (Application Programming Interface) ja luo allekirjoituksen käyttämällä SHA-1 tiivistefunktiota. Nykyisin SHA-1 ei katsota olevan enää turvallinen törmäysten takia, jotka tarkoittavat tilannetta, jossa kaksi eri syötettä saavat aikaan saman lopputuloksen. V2 on laitteille, joissa API-taso on korkeampi kuin 21. V2 käyttää uudempaa SHA-tiivistefunktiota, joka on turvallisempi.



6. Paina “Finish”, jonka jälkeen APK:n generoiduttua se on kopioitavissa juurihakemiston alapuolelle luodusta “release”-hakemistosta



3.8 Viikko 8

Maanantai 8.4.2019

Tänään aloin implementoida sovellukseen tarkistusta puhelimista, joissa on root-käyttäjän oikeudet. Sovelluksessa halutaan siis sen käynnistyessä huomauttaa käyttäjää, mikäli hänen puhelimensa on niin sanotusti rootattu. Rootattuun puhelimeen liittyy paljon tietoturvariskejä, koska kaikki järjestelmän tiedostot ja tiedot ovat puhelimessa saatavilla.

Löysin Githubista tähän tarkoitukseen kirjaston nimeltä RootBeer. Tein sovelluksen aloituskohtaan tarkistuksen, joka näyttää käyttäjälle huomautusdialogin, mikäli kirjasto tunnistaa laitteen rootatuksi. Meillä ei kuitenkaan ollut yhtään rootattua Android-laitetta, joten seuraavaksi tehtäväkseni tuli rootata LG K8 -älypuhelin. Tämä paljastui isommaksi työksi kuin olin ajatellut. Työskentelin asian parissa aamupäivästä työpäivän loppuun ja päivän päätteeksi sain puhelimen käynnistymään todennäköisesti rootattuna, mutta se jäi lataamaan käynnistysanimaatioon, joten jätin puhelimen laturiin ja huomenna näemme, onko mitään tapahtunut.

Tiistai 9.4.2019

Aamulla LG -puhelin oli yhä käynnistysanimaatiossa, joten totesin yritykseni epäonnistuneeksi ja palautin puhelimen tehdasasetuksiin. Onneksi yksi työkavereistani tuli toimistolle päiväksi, koska hänellä ei ollut töitä omassa asiakasprojektissaan tälle päivälle. Hän alkoi sitten yrittämään samaa Motorolan älypuhelimella, ja onnistui roottaamaan laitteen, joten pääsimme testaamaan root detection implementaatiotani. Implementaatio oli onnistunut, koska tekemäni dialogi ilmestyi sovelluksen käynnistyessä

Keskiviikko 10.4.2019

Työpäivä alkoi kahden viikon välein järjestettävällä palaverilla asiakkaan kanssa. Esittelimme suunnittelemaamme aikataulua sovelluksen uuden version valmistumiseen, jonka asiakas hyväksyi. Näytin myös palaverin yhteydessä root detection -ominaisuutta, jonka tein alkuviikosta. Ominaisuus oli sellainen kuin asiakas oli halunnut, mutta sain ohjeet lisätä siihen mahdollisuuden olla näkemättä huomautusta enää seuraavilla kerroilla, kun sovellus käynnistetään. Palaverin jälkeen aloin tehdä tätä lisäominaisuutta lisäämällä dialogiin valintaruudun ”Älä näytä tätä huomautusta enää” ja tallentamalla valinnan laitteeseen, jos valintaruutu on valittuna. Tämän jälkeen lisäsin tarkistuksen root-tarkistuksen yhteyteen siitä, onko asia huomioitu jo aiemmin, eikä dialogia haluta nähdä uudelleen.

Lisäominaisuus toimi odotetulla tavalla. Nyt jos asia on huomioitu ja tallennettu laitteeseen, dialogia ei näytetä uudelleen vain kuin silloin, jos sovellus asennetaan uudelleen, jolloin kaikki sovellukseen liittyvä data poistetaan Shared Preferenceistä.

Torstai 11.4.2019

Aloin refaktoroida sovelluksen pää-activityn koodia. Kuten olen aiemmin jo maininnut, kyseiseen activityyn on kirjoitettu yli 2000 riviä ja koko activity on niin sanottua ”spagettikoodia”. Tämä refaktorointi tulee olemaan tuskallista, koska sovelluksen logiikkaa on sidottu yhteen näkymän ja elämänkaaren kanssa nii paljon. Mielestäni tämä refaktorointi on kuitenkin välttämätön tehtävä tulevien muutosten onnistumisen kannalta.

Aloitin työn helpoimmista asioista, kuten tekemällä Shared Preferencien käyttämiseen apuluokan, joka on singleton ja hoitaa kaikki Shared Preferenceihin liittyvät tehtävät. Tämän jälkeen aloin tehdä uutta tunnistautumisfragmenttia, jonka tarkoituksena on hoitaa tunnistautuminen, tapahtui se sitten pin-koodilla tai sormenjäljellä. Tarkoituksena on saada aikaan toiminto, jossa activitylle ei kerrota muuta kuin, että onnistuiko tunnistautuminen.

Todennäköisesti tähän toimintoon olisi järkevää lisätä jonkinlainen jaettu viewmodel-luokka, joka pitää huolta tunnistautumiseen liittyvästä datasta ja välittää dataa fragmentin ja activityn välillä. En päässyt kuitenkaan asiassa vielä niin pitkälle, koska päivä meni kopioidessa metodeja activitystä fragmentin hoidettavaksi, kommentoimalla metodit pois käytöstä activitystä ja tarkistamalla vähän väliä, hajosiko mitään.

Perjantai 12.4.2019

Jatkoin refaktoroinnin parissa. Lisäsin tunnistautumisominaisuuteen Live Dataa tarjoilevan viewmodel-luokan. Tarkoituksena on määrätä activity tarkkailemaan viewmodelin muutoksia ja päivittää näkymä, jos tarkkailtava kenttä ”authenticated” muuttuu todeksi. Live Datasta kerrotaan enemmän tämän viikon viikkoanalyysissä.

Tämä ratkaisu vaikutti toimivan pin-koodilla tunnistauduttaessa, mutta jostain syystä sormenjälkitunnistautuminen hajosi. En ehtinyt jatkaa pidempään asian parissa, koska keskityimme iltapäivällä tekemään ensimmäisen alpha-julkaisun tekemistämme muutoksista Google Playhin asiakkaan testattavaksi. Saimme julkaisun tehtyä ja lähdimme viettämään viikonloppua.

Viikkoanalyysi

Viikko sisälsi mielenkiintoisia työtehtäviä, joissa onnistuttiin. Sovelluksen rakenteen siistiminen ei ole vielä lähelläkään valmista ja jatkan asian parissa todennäköisesti ensi viikolla, ellei mitään akuutimpaa ilmene. Tarkoitus olisi istua alas työparini kanssa ja piirtää sovelluksen rakennetta auki, sekä suunnitella uutta ja järkevämpää ratkaisua. Lisäksi tiistaina menemme lounaalle asiakasyrityksemme kontaktihenkilöiden ja oman projektitiimimme kanssa.

Opin viikon aikana lisää Androidin arkkitehtuurista uudistaessa pää-activityn koodia, sekä laiteläheisemmästä puolesta koittaessa rootata älypuhelinta.

ViewModel-luokka on suunniteltu varastoimaan ja hallinnoimaan käyttöliittymään liittyvää dataa activity elämänkaari huomioon ottaen. ViewModel-luokka ratkaisee näin myös konfiguraatiomuutosten tuomat ongelmat kuten datan säilymisen näytön rotaation muuttuessa. (Android Developers 2019, viitattu 13.4.2019.)

LiveData on tarkkailtava (observable) datan säilyttäjäluokka. LiveData on tietoinen sovelluksen komponenttien eli activityjen ja fragmenttien elämänkaaresta. Tämä varmistaa sen, että ainoastaan aktiivisena olevat komponentit, jotka tarkkailevat LiveDataa, päivittyvät. (Android Developers 2019, viitattu 13.4.2019.)

LiveDataa ja ViewModelia käytetään yhdessä siten, että ViewModel tarjoilee LiveDataa activityn tai fragmentin tarkkailtavaksi. Tämä on niin sanottu tarkkailija-suunnittelumalli (observer pattern).

3.9 Viikko 9

Maanantai 15.4.2019

Jatkoin refaktoroinnin parissa, mutta ilmeni muutama ongelma, jotka vaativat laajempia muutoksia sovelluksen koodiin, koska sovelluksen osat ovat niin tiukasti sidoksissa keskenään. Ongelmiin törmätessäni käytin loppupäivän tutkiessa parhaita menetelmiä legacy-koodin uudelleenkirjoittamiseen ja sovimme työkaverini kanssa, että huomenna pidämme suunnittelupalaverin, kunhan hän saa omat tehtävänsä alta pois ja piirrämme sovelluksen uutta rakennetta läpi.

Tiistai 16.4.2019

Aamulla pohdimme ja suunnittelimme sovelluksen arkkitehtuuria suullisesti työkaverini kanssa, jonka jälkeen lähdimme lounaalle tapaamaan asiakastamme. Lounaspalaveri kesti noin puolitoista tuntia ja oli enemmänkin avointa keskustelua asioista, joita mieleen oli tullut. Asiakas oli yhä hyvin tyytyväinen palveluumme ja siihen, mitä tähän mennessä oli saatu aikaiseksi.

Iltapäivällä istuimme työkaverini kanssa alas palaverihuoneeseemme, jossa on valkotaulu ja aloimme hahmotella suunnittelumallin implementoitia koodiin. Alustavan arkkitehtuurin piirrettyämme jaoimme tehtävät niin, että hän alkaa tehdä sovelluksen model-kerrosta ominaisuus kerrallaan ja minä hoidan vastaavasti tämän ominaisuuden datan tarjoamisen viewmodelin kautta viewille. Aloitamme ominaisuudesta, jossa objekteja X haetaan etäpalvelusta ja tarjotaan listanäkymänä käyttäjälle.

Keskiviikko 17.4.2019

Tänään työkaverini työskenteli model-kerroksen parissa ja minä tein listanäkymälle viewmodelin ja sille kuuluvat metodit, sekä karsin koodia pois listafragmentista. Emme päässeet vielä siihen pisteeseen, että olisimme voineet testata mitään.

Huomenna aloitan työt todennäköisesti tekemällä luokan, joka tarjoilee keinotekoista dataa viewmodelille, jotta pääsen testaamaan viewmodelia ja etenemään sovelluksen arkkitehtuurin parantelussa.

Torstai 18.4.2019

Päivän työt etenivät toivotulla tavalla. Aamupäivällä sain tehtyä keinotekoista (mock) dataa tarjoilevan luokan ja testattua view-viewmodel-yhteyttä, jonka totesin toimivaksi. Lounaan jälkeen yhdistimme muutokset minun ja työkaverini Git-haaroista, jotta pystyimme kytkemään viewmodelin oikeaan dataan, joka tulee model-kerrokselta. Pienen säätämisen jälkeen oikea datakin saatiin tulemaan UI-kerrokselle listaan näkyviin.

Tämän asian onnistuminen oli iso askel kohti parempaa arkkitehtuuria, koska saimme jo karsittua paljon logiikkaa pois UI-kerroksesta. Ensi viikolla jatkamme sovelluksen muiden osien paloittelemista MVVM-malliin.

Perjantai 19.4.2019

Pääsiäinen, ei töitä.

Viikkoanalyysi

Viikon työt menivät hyvin, ja jo osittain toimiva uusi arkkitehtuuri toi motivaatiota ja todisteen siitä, että sovellus on järkevämpää rakentaa tällä tavoin. Itse ohjelmointia ei taaskaan sisältynyt kovin paljon viikon töihin, vaan se oli suurimmaksi osaksi vanhojen metodien siirtelemistä uusiin paikkoihin.

Tässä viikkoanalyysissä esitellään LiveData-esimerkki käyttäen ViewModelia ja Data Bindingia. Android Studiossa on mahdollista hoitaa muuttujien arvojen lisääminen näkymäkomponentteihin suoraan xml-tiedostossa. Tämä tekee näkymästä vielä passiivisemman.

Datansidontakirjaston avulla kehittäjä voi sitoa käyttöliittymäelementtejä suoraan datan lähteeseen ilman, että se pitäisi tehdä ohjelmallisesti. Näin se poistaa useita muuten pakollisia käyttöliittymäkerroksen metodikutsuja, kuten näkymien hakemista id:n perusteella. Tämä tekee näkymästä yksinkertaisemman ja helpomman ylläpitää. (Adefioye. T. 2018, viitattu 20.4.2019.)

Esimerkissä meillä on ViewModel-luokka, joka tarjoilee näkymälle dataa paikallisesta tietokannasta LiveDatana ja sitoo sen näkymään Data Bindingin avulla. Olen toteuttanut esimerkin Kotlinilla.

3.9.1 ItemViewModel

```
class ItemViewModel(application: Application): AndroidViewModel(application) {

    private var itemRepository: ItemRepository = ItemRepository(application)
    private lateinit var allItems: LiveData<List<Item>>

    init{
        allItems = itemRepository.getAllItems()
    }

    fun getAllItems(): LiveData<List<Item>>{
        return allItems
    }

    fun getItemById(id: Int): LiveData<Item>{
        return itemRepository.getItem(id)
    }
```

KUVIO 8. ItemViewModel-luokka

ItemViewModelin metodit hoitavat kommunikoimisen ItemRepositoryn kanssa, joka vastaavasti huolehtii tietokantatoiminnoista. ItemRepositoryn get-metodit palauttavat LiveDataa, joka voidaan suoraa välittää näkymälle

3.9.2 DetailActivity

```
class DetailActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDetailBinding
    private lateinit var viewModel: ItemViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_detail)
        bindUI()
    }

    fun bindUI(){
        viewModel = ViewModelProviders.of( activity: this).get(ItemViewModel::class.java)
        viewModel.getItemById(intent.extras.get("item_id") as Int).observe( owner: this, Observer { it: Item!
            binding.item = it
        })
    }
}
```

KUVIO 9. DetailActivity-luokka

DetailActivity on näkymä, jossa luodaan sidonta xml-tiedoston komponenttien ja datan välille. ActivityDetailBinding-luokka on Android Studion automaattisesti generoima sidontaluokka activitylle, kun data binding on kytketty päälle

BindUI() -metodi on ainoastaan koodin järjestelyä varten. ViewModelin alustaminen ja observointi voitaisiin tehdä yhtä lailla suoraan onCreate -metodin sisällä.

3.9.3 Activity_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>
        <variable name="item" type="com.henri.yearlylist.data.room.Item"/>
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:context=".DetailActivity">

        <TextView
                android:text="@{item.title}"
                android:layout_width="wrap_content"
                android:layout_height="0dp"
                android:id="@+id/detail_title_txt" app:layout_constraintEnd_toEnd
                android:layout_marginEnd="8dp"
                app:layout_constraintStart_toStartOf="parent" android:layout_marg
layout › androidx.constraintlayout.widget.ConstraintLayout
```

KUVIO 10. Datan sitominen DetailActivityn layout-tiedostoon.

Datan sidontaa käyttäessä uloimmaksi xml-tagiksi laitetaan layout-tagi, jonka sisälle määritellään data ja sen muuttujat. Xml-tiedostossa oleva muuttuja ”item” vastaa Item-olioa, joka annetaan datansidontaluokalle näkymässä ViewModelin observointimetodin sisällä: binding.item = it. Nyt UI-komponentteihin voidaan valmiiksi määritellä arvot mitä niissä halutaan näyttää, eikä tätä tarvitse tehdä näkymässä. Tässä ylimmässä TextViewissä esimerkiksi halutaan näyttää Item-olion otsikko.

Myös klikkaustapahtumien metodit voidaan hoitaa suoraan xml-tiedostosta. Näin ClickListenereitä ei tarvitse määritellä activityssä. Alla olevassa kuvassa on listanäkymän yhden objektin xml-tiedosto, jota klikkaamalla kutsutaan rajapinnan metodia ”handleClick”.

```
    <data>
        <variable
                name="item"
                type="com.henri.yearlylist.data.room.Item"/>
        <variable
                name="itemClickListener"
                type="com.henri.yearlylist.ListItemClickListener"/>
    </data>

<androidx.constraintlayout.widget.ConstraintLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
         android:onClick="@{() -> itemClickListener.handleClick(item)}">
    <TextView
            android:id="@+id/list_item_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="@android:style/TextAppearance.Material.Title" android:text="@{item.title}"
            app:layout_constraintTop_toTopOf="parent"
            android:layout_marginBottom="8dp" app:layout_constraintBottom_toBottomOf="parent"
```

KUVIO 11. Klikkaustapahtuman hallinta.

3.10 Viikko 10

Maanantai 22.4.2019

Pääsiäinen, ei töitä

Tiistai 23.4.2019

Viikko alkoi asiakkaan vierailulla toimistollamme ja joka toisella viikolla järjestettävällä palaverilla. Palaverissa käytiin läpi sovelluksen backendissä ilmeneviä ongelmia, joita on tullut ilmi, mutta jotka asiakkaan pitää itse hoitaa, koska se ei kuulu projektiimme.

Palaverin jälkeen jatkoimme työkaverini kanssa autentikaatio-ominaisuuden paloittelemista modulaarisemmaksi minun hoitaessa view-viewmodel -yhteyttä ja hänen siirrellessä logiikkaosuutta model-kerrokseen.

Keskiviikko 24.4.2019

Jatkoin laitelistanäkymän refaktoroinnin parissa. Uuden arkkitehtuurin myötä näkymän tilanhallinta oli vielä hieman hukassa, eikä päivittänyt listanäkymän objektien tilaa ilmaisevaa tekstiä, kun muutoksia tapahtui. Saimme korjattua tämän, mutta aloin myös tehdä toiseen haaraan samaa listanäkymää, mutta Androidin ListView-komponentin tilalle vaihdettaisiin Recycler View -komponentti, joka on uudempi ja suositellumpi tapa esittää asioita listoissa. Recycler View helpottaa yksittäisien objektien päivittämistä ja on suorituskyvyn kannalta parempi vaihtoehto. Sain Recycler Viewin ja model-kerroksesta tulevan Live Datan yhdistettyä jo niin, että Recycler View näyttää modelista tulevat objektit, mutta ei hoida vielä objektien klikkaustapahtumia.

Torstai 25.4.2019

Jätin eilen aloittamani Recycler Viewin implementoinnin tauolle ja aloitin korjaamaan viimeistä UI-ongelmaa, jonka korjaus sisältyy sovelluksen ensimmäiseen päivitettyyn versioon. Sovelluksessa on käytetty kustomoitua Action Baria ja koska se on vanhentunut, uusimmassa Android-versiossa (API 28) Action Barin valikon tekstit näkyvät mustana, vaikka niiden pitäisi näkyä valkoisena.

Ongelmana oli siis vanhan Action Barin yhteensopimattomuus uusimman API-tason kanssa. Asia ratkesi refaktorointihaarassani vaihtamalla Action Bar Tool Bariksi, mutta päähaarassa, jossa tätä refaktorointia ei ole vielä tehty, se ei ollut mahdollista, joten päädyin vaihtamaan valikon tekstien värit ohjelmallisesti. Tulevaisuudessa vanhan Action Barin tilalle tulee uusi Tool Bar, kun saamme yhdistettyä arkkitehtuurimuutoksemme päähaaraan.

Perjantai 26.4.2019

Työpäivä kului hajanaisissa tehtävissä. Korjailin arkkitehtuurimuutoksen alla olevassa haarassa UI-muutoksia samannäköisiksi, kuin julkaisuversiossa. Tein vanhan Action Barin tilalle muuttamalleni Tool Barille samannäköisen ulkoasun kuin alkuperäisessä versiossa.

Lisäksi siirtelin ja poistelin lisää koodia MainActivitystä, joka alkaa olla jo paljon siistimmässä kunnossa. Iltapäivällä luin lisää dokumentaatiota datan sidonnasta (Data Binding) Androidissa.

Viikkoanalyysi

Viikko oli taas lyhyempi pääsiäisen takia, mutta uuden arkkitehtuurin rakentelussa edistyttiin kiitettävästi. Työ oli suurimmaksi osaksi jälleen kerran metodien siirtelyä pois activitystä ja activityn irrottamista eri luokkien kuuntelijoista.

Tuntuu siltä, että kaipaisin hieman lisää harjoittelua Git-versionhallinan käytössä. Olen kehittynyt sen käyttämisessä huomattavasti, mutta silti vielä jotkin toiminnot tuottavat epäröintiä.

Positiivista viikossa oli se, että allekirjoitin työsopimuksen yrityksen kanssa työnimikkeellä Software Engineer. Uskon, että tämän projektin onnistuminen tuo minulle lisää itsevarmuutta ja taitoa Android-kehityksessä ja arkkitehtuurin suunnittelemisessa.

Recycler Viewin käyttö Android-sovelluksessa on hyödyllistä silloin, kun sovelluksessa pitää esittää suuria määriä tai jatkuvasti muuttuvaa dataa listana. Recycler View on kehittyneempi ja joustavampi versio List Viewistä. (Android Developers 2019, viitattu 27.4.2019.)

Listan (Recycler View) sisältö esitetään View Holder -objekteina. View Holder -objektit ovat olioita luokasta, joka perii RecyclerView.ViewHolderin. Recycler View luo View Holder -objekteja vain sen verran, kuin näytöllä on tilaa esittää niitä sekä luo ja poistaa niitä sitä mukaan, kun käyttäjä vierittää listaa. View Holder -objekteja hallinnoimisesta vastaa Adapter-luokka, joka luodaan perimällä RecyclerView.Adapter.

Luodaan activity tai fragment, jonka layout-tiedostoon asetetaan Recycler View.

```
<androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
>


    <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/list_recyclerview"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_margin="0dp"
            android:padding="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"/>
```

KUVIO 12. RecyclerView xml-tagi.

Luodaan ViewHolder luokka, joka huolehtii referensseistä listan objektin xml-tiedoston näkymäkomponentteihin. Tässä esimerkissä olen käyttänyt datan sidontaa, joten parametrina annettava ListItemBinding vastaa yhden listaobjektin näkymää. Binding-oliolle annetaan muuttujaksi dataobjekti, joka halutaan näyttää.

```
class ItemViewHolder(val binding: ListItemBinding) : RecyclerView.ViewHolder(binding.root){
    fun bind(item: Item){
        binding.item = item
    }
}
```

KUVIO 13. ItemViewHolder vastaa referoimisesta xml-tiedostoon.

Luodaan Adapter-luokka, joka huolehtii View Holdereiden täyttämisestä datalla. Adapterin onCreateViewHolder -metodissa luodaan datansidontaluokka, joka annetaan parametrina metodissa palautettavalle ViewHolderille. Tämän jälkeen kutsutaan onBindViewHolder-luokkaa, jossa tämä luotu ViewHolder tulee nyt parametrina ja johon voidaan sitoa data.

```
class ItemRecyclerViewAdapter(context: Context) : RecyclerView.Adapter<ItemViewHolder>(), ListItemClickListener {

    private var context =  context
    var myItems: List<Item> = mutableListOf()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        // create a new view
        //ei tarvita nyt kun tehdään binding
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent,  attachToRoot: false)

        val inflater = LayoutInflater.from(parent.context)
        val binding = ListItemBinding.inflate(inflater)

        return ItemViewHolder(binding)
    }
```

KUVIO 14. Adapteriluokka, joka täyttää listanäkymän datalla.

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
    holder.bind(myItems[position])
    holder.binding.itemClickListener = this

}
```

KUVIO 15. Adapterin metodi, jossa data sidotaan listaelementtiin.

Lopuksi fragmentissa tai activityssä, jossa Recycler View sijaitsee, määritellään sille adapteri sekä Layout Manager. Kun LiveData muuttuu ViewModelissa kutsutaan adapteriin luotua metodia setItems, joka määrittelee listassa esittävän datan uudeksi ja ilmoittaa siitä Recycler Viewille, joka osaa päivittää muuttuneen datan.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    setupUI()
}

private fun setupUI(){
    val adapter = ItemRecyclerViewAdapter(context!!)

    viewModel = ViewModelProviders.of( fragment: this).get(ItemViewModel::class.java)
    list_recyclerview.adapter = adapter
    list_recyclerview.layoutManager = LinearLayoutManager(context)

    viewModel.getAllItems().observe( owner: this, Observer { it: List<Item>!
        adapter.setItems(it)
    })
}
```

KUVIO 16. Recycler Viewin alustus.

4 POHDINTA

Opinnäytetyön päiväkirjaraportoinnin kannalta asiakasprojekti ei välttämättä ollut ihanteellinen, koska projekti sisälsi paljon bugien korjaamista ja vanhan päivittämistä uuteen. Saattoi kulua viikko samaa ongelmaa debugatessa, jossa päivät kuluivat dokumentaatiota ja StackOverflow:ta lukiessa. Tämän kaltaisten asioiden raportointi järkevällä tavalla osoittautui haastavaksi. Todennäköisesti projektista, jossa luodaan uutta sovellusta alusta asti, lukija voisi saada paremman käsityksen ja asioita olisi helpompi selittää, kun koodi on itse kirjoitettua. Kuitenkin, opinnäytetyön aiheena oli kuvata Android-sovelluskehittäjän työtä kymmenen viikon ajalta ja siinä onnistuttiin.

Pyrin viikkoanalyysejä kirjoittaessani siihen, että käyn niissä läpi kuluneella viikolla eteeni osuneita asioita ja avaan teoriaa niiden taustalla. Kaikki kymmenen viikon viikkoanalyysit lukiessaan lukijan pitäisi saada kohtalainen yleiskuva siitä, mitä asioita Android-kehitykseen sisältyy. Joissakin viikkoanalyyseissä käydään läpi kehittyneempiä asioita, joita Androidin kanssa aloittelevan ohjelmoijan ei heti tarvitse miettiä. Osa teoriasta taas on hyödyllistä ymmärtää heti aloittelijankin, kuten millä tavalla Android-sovelluksen elämänkaari toimii.

Olen oppinut paljon uusia asioita Android-sovelluskehityksestä ja tiimityöskentelystä kuluneen projektin aikana. Yhtenä uutena asiana minulle tuli alkuviikkojen aikana opetellut asiat natiivikoodin käyttämisestä Android-sovelluksessa. Lukuisista muista asioista, kuten reaktiivisesta ohjelmoinnista ja suunnittelumalleista, olen tiennyt vain pintaraapaisun verran ja joihin nyt olen saanut syventävää oppia. Asiakasprojektin monimutkaisuudesta johtuen jotkut illat ovat menneet opiskellessa Androidiin liittyviä asioita ja tehdessä testiprojekteja. Olen tuntenut itseni päteväksi henkilöksi tähän projektiin ja onnistunut monissa vaikeissa asioissa. Isona apuna tässä on ollut erittäin osaava projektitiimi ja se, että olemme työskennelleet samassa tilassa yhdessä lähes joka päivä, mikä helpottaa kommunikointia ja neuvojen kysymistä.

Opinnäytetyöprosessi oli mielestäni haastavaa kirjoittamisen ja sisällön tuottamisen osalta. Kymmenen viikkoa meni kuitenkin kohtalaisen nopeasti, eikä opinnäytetyön kirjoittaminen töiden ohella ollut turhan raskasta. Tykkään sovelluskehityksestä Androidille ja pidän itseäni jo kohtalaisena tekijänä. Jatkan Androidin parissa työskentelyä tässä asiakasprojektissa, joka jatkuu vielä pitkän ajan. Uutena asiana jossakin vaiheessa alan opettelemaan mobiilisovellusten kehittämistä iOS-

laitteille, koska nyt minulla on tarvittavat työkalut siihen. Uskon että tulevaisuudessa tulen työskentelemään mobiilisovellusten ja sovellusarkkitehtuurin parissa. Tällä alalla on erityisen tärkeää omata halu oppia ja opetella itsenäisesti, koska asiakasprojektit saattavat pyöriä vain jonkun tietyn asian ympärillä, eikä niissä välttämättä aina pääse soveltamaan uusimpia käytäntöjä tai työkaluja sovelluskehityksessä. Omia sivuprojekteja tehdessä taitotasot kasvavat ja jatkuvasti kehittyvien teknologioiden mukana pysyminen on helpompaa.

LÄHTEET

Argawal, N. 2017. Android MVP for Beginners. Viitattu 30.3.2019, https://android.jlelse.eu/android-mvp-for-beginners-25889c500443

Adefioye, T. 2018. Empower your UI with data binding. Viitattu 20.4.2019, https://medium.com/@temidjoy/android-jetpack-empower-your-ui-with-android-data-binding-94a657cb6be1

Android Developers. 2019a. App Manifest Overview. Viitattu 02.3.2019, https://developer.android.com/guide/topics/manifest/manifest-intro

Android Developers. 2019b. Getting started with the NDK. Viitattu 09.3.2019, https://developer.android.com/ndk/guides

Android Developers. 2019c. Publish your app. Viitattu 06.4.2019, https://developer.android.com/studio/publish

Android Developers. 2019d. ViewModel Overview. Viitattu 13.4.2019, https://developer.android.com/topic/libraries/architec-ture/viewmodel

Android Developers. 2019e. LiveData Overview. Viitattu 13.4.2019, https://developer.android.com/topic/libraries/architecture/live-data

Android Developers. 2019f. Create a list with Recycler View. Viitattu 27.4.2019, https://developer.android.com/guide/topics/ui/lay-out/recyclerview

Jwt.io. 2019. Introduction to JSON web tokens. Viitattu 23.2.2019, https://jwt.io/introduction/