TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Erik Kim

Kuidas mõjutab mobiiliäpi arendusraamistiku valik arendusprotsessi ja lõpptoodet? Juhtumiuuring

Bakalaureusetöö (9 EAP)

Juhendaja: Kristiina Rahkema

Tartu 2022

Kuidas mõjutab mobiiliäpi arendusraamistiku valik arendusprotsessi ja lõpptoodet? Juhtumiuuring

Lühikokkuvõte:

Selles lõputöös uuritakse, kuidas mõjutab mobiiliäpi arendusraamistik arendusprotsessi ning lõpptoodet. Uuringu käigus luuakse kolm mobiilirakendust: *native* Android, *native* iOS ning Flutter. Mobiilirakenduste vahel uuritakse koodipikkust, kompileerimisaega, arendusteekide lisamist ning arendaja enda kogemust. Mobiilirakendused on loodud ettevõttele Leadfellow.

Võtmesõnad:

Mobiilirakendus, Android, iOS, Flutter

CERCS: P170 Arvutiteadus

How does the choice of the mobile application development framework affect the development process and the end product? A case study

Abstract:

The aim of this Bachelor's thesis is to examine how the choice of the mobile application development framework affects the development process and the end product. During the study, three mobile applications are made: native Android, native iOS and Flutter. The comparison is made based on code length, build time, adding of dependencies and the developers personal experience. The mobile applications are made for the company Leadfellow.

Keywords:

Mobile application, Android, iOS, Flutter

CERCS: P170 Computer science

Sisukord

Sissejuhatus

Leadfellow[1] on Eesti ettevõte, mis tegeleb aktsiate ostmise ja müümisega. Kasutaja saab Leadfellow-i teenuseid kasutada soovitajana (referrer) või teenusepakkujana (provider). Soovitaja on isik, kes saab edastada aktsiatehinguid ning teenusepakkuja on isik, kellele soovitaja pakub aktsiatehingut. Tehingut läbi viies, teenib soovitaja tehingult vahetasu. Antud hetkel töötab Leadfellow vaid veebibrauseris. Läbi olemasoleva veebirakenduse, saab teenusepakkuja ülevaate enda aktsiatehingutest ning neid ka hallata. Soovitaja saab teenusepakkujatele edasi saata uusi tehingud. Leadfellow soovib enda teenuseid klientidele, kes kasutavad Leadfellow teenuseid soovitaja rollis, mugavamaks teha, mistõttu on neil vaja mobiilirakendust, mida neil hetkel ei ole. Antud ettevõttele mobiilirakendust tehes, on hea võimalus uurida üleüldist mobiiliarenduse maailma ning paremini õppida erinevaid mobiilse tarkvara arenduse võimalusi.

Tänapäeva maailmas on nutitelefoni omamine justkui põhivajadus. Statista, Saksa ettevõtte, mis tegeleb turu- ning kliendiandmetega, väitel omab 80,63% maailma rahvastikust nutitelefoni [1]. Kuna nutiseadme mugavusfaktor on palju kõrgem kui näiteks lauaarvutil, on ettevõtetel mõistlik enda teenuseid pakkuda ka mobiilselt, et kliendil mugavam oleks. Seepärast soovib ka Leadfellow luua enda ettevõttele mobiilirakenduse. Kuna nii ettevõtel kui ka arendajal pole varasemat kokkupuudet mobiilsete rakenduste arendamisega, on keeruline valida, kas rakendust tuleks luua iOS ja Android *native* või hübriidselt. Lõputöö eesmärgiks on arendada ettevõttele Leadfellow mobiilirakendus kolmel erineval viisil ning leida erinevusi erinevate mobiilirakenduse arendamisviiside vahel, et otsustada, kumb variant antud kontekstis on mõistlikum. Erinevuste leidmiseks saab võrrelda sama rakendust, mis on arendatud täielikult Android-il ja täielikult iOS-il hübriidlahendusega. Viimane tähendab, et rakendus on arendatud mõlemale platvormile korraga. Uuringu lõpuks tehakse saadud tulemuste põhjal valik, kas ettevõttele tehtud rakendus tuleb Android-il ja iOS-il eraldi arendatult või hübriidselt. Uuringu käigus saadud tulemused aitavad nii käesoleva rakenduse arendajal kui ka teistel mobiilirakenduste arendajatel valida vastavalt olukorrale sobiliku arendusraamistiku.

[1] https://leadfellow.com/

2. Seotud kirjandus

Varasemalt on sarnaseid uurimustöid juba ka tehtud, kus uuritakse *native* rakendusi ning hübriidselt arendatud rakendusi. Näiteks 2020 aastal uuris Matilda Olsson Flutteri ja *native* rakendusi [2]. Olsson leidis, et Flutteril loodud hübriidne rakendus vajab märgatavalt vähem koodiridu. Surf, ettevõte mis tegeleb mobiilsete rakenduste arendamisega, on samuti uurinud *native* rakenduste ning Flutteriga loodud hübriidrakenduste erinevusi. Surf ettevõtte arendajad on öelnud, et hübriidselt loodud koodi on lihtsam korrastada ning korrigeerida, kui kahele eraldi platvormile loodud koodi. Mobiilirakendustel on vaja pidevalt teha uuendusi ning erinevaid tekkinuid probleeme lahendada, mis võtab eraldi koodibaasidega töötades rohkem aega kui ühe koodibaasiga töötades. Samuti toob Surf välja ka hübriidse arenduse mõju suurematele projektidele. Kuna hübriidse arenduse koodibaas on väiksem, vajab rakendus seega vähem tööjõudu. Kuid kuna tegemist on võrdlemisi uue tehnoloogiaga, on ka tööjõudu antud valdkonnas vähem [3]. Poola ettevõte itCraft, mis tegeleb mobiilirakenduste arendamisega on uurinud erinevust hübriidse arenduse (Dart/Flutter) ning iOS *native* vahel. Selles võrdluses ei arvestatud Androidiga. ItCraft võrdluses tuli välja, et esialgne seadistus on *native* iOS puhul lihtsam, kuna selleks on vaja allalaadida vaid Xcode. Flutter arenduse puhul on lisaks Xcode-le vaja ka mõnda Android arenduskeskkonda. Rakenduste kompileerimisaeg oli võrdluses erinev, *native* rakenduse kompileerimisaeg oli lühem kui hübriidse. Lisaks kompileerimisajale ning ülesseadistamisele võrreldi ka rakenduste testimist. Rakenduste testimine on mõlema arendusraamistiku puhul erinev kuid üks pole lihtsam kui teine ehk lõplik valik jääb isiklikule eelistusele [4].

3. Taust

Tausta peatükk tutvustab antud lõputöös kasutatud tehnoloogiaid ning Leadfellow ettevõtte poolt paika seatud nõuded mobiilse rakenduse jaoks. Kasutatud tehnoloogia alapeatükkides tutvustatakse erinevaid arendusraamistikke ning keeli, milles rakenduste koode kirjutati. Mobiilirakendus peab töötama nii Androidi kui ka iOSi platvormidel, mistõttu on vaja kasutada erinevaid programme. Androidi nutitelefonidele rakenduse valmistamiseks saab kasutatada Android Studiot ning programmikoodi kirjutada Kotlini keeles. IOS- seadmetele saab valmistada rakendusi tarkvaraga Xcode ning programmeerimiskeelega Swift. Mobiilirakendused peavad ühilduma ka andmebaasiga, andmete saamiseks ning salvestamiseks. Järgnevalt kirjeldatakse täpsemalt lahti platvormid, millele rakendused luuakse, Android ning iOS. Samuti kirjeldatakse ka arenduskeeli ning programmeerimiskeskkondi, mida kasutatakse uuritava mobiilirakenduse loomiseks.

3.1. Android

Android on avatud lähtekoodiga, Linuxi põhjal loodud tarkvarakomplekt. Androidi aluseks on Linux-i operatsioonisüsteemi tuum (kernel). Näiteks Android Runtime (ART) raamistik kasutab Linux-i kernel-it alusfunktsionaalsuste jaoks, nagu hargtöötlus ning mälujaotus [5].

3.1.1. Android Studio

Android Studio on ametlik integreeritud programmeerimiskeskkond (Integrated Development Environment - IDE) Androidi mobiilirakenduste arendamiseks. Android Studio on loodud JetBrains-i IntelliJ IDEA programmeerimiskeskkonna põhjal. Androidi süsteemidele luuakse rakendusi enamasti keeltes Java ning Kotlin [6]. Siinses uuringus on Androidi mobiilirakenduse arendamiseks kasutusel Kotlin.

3.1.2. Kotlin

Kotlin on tasuta avatud lähtekoodiga pragmaatiline programmeerimiskeel, mis on loodud JVMi (Java Virtual Machine - Java virtuaalmasin) ning Androidi jaoks. Kotlin ühendab objektorienteeritud ning funktsionaalse programmeerimise põhimõtteid. Nii Android Studio kui ka Kotlin-i asutas JetBrains. Kotlin loodi 2010. aastal ning alates 2012. aastast on antud programmeerimiskeel avatud lähtekoodiga [7].

3.2. iOS

iOS (iPhone Operating System) on Apple'i loodud operatsioonisüsteem ekslusiivselt Apple telefonidele.

3.2.1. Xcode

Xcode on Apple loodud ametlik programmeerimiskeskkond, millega luuakse rakendusi macOS-ile (Mac lauaarvutid, Macbook sülearvutid), iOS-ile (Apple-i nutitelefonid ning tahvlid), watchOS-ile (Apple-i nutikellad) ning tvOS-ile (Apple-i *streaming* ehk voogedastusseade). Xcode sisaldab kõiki vajalikke tööriistu, et kirjutada, testida ning kokku panna erinevaid rakendusi. Xcode toetab erinevaid programmeerimiskeeli: Swift, Objective-C, C++ jne. Apple lõi Xcode-i, et arendajatel oleks kõik vajalik olemas ühes kindlas komplektis [8]. Antud uuringu jaoks valiti programmeerimiskeeleks Swift, kuna see on Apple poolt soovitatud programmeerimiskeel Apple seadmetele programmide kirjutamiseks.

3.2.2. Swift

2014. aasta juunis avaldas Apple Objective-C programmeerimiskeele järeltulija – Swift. Swift on mitmeparadigmaline programmeerimiskeel, mis ühendab imperatiivset, objektorienteeritud ning funktsionaalset programmeerimist. Ainult kaks aastat pärast väljatulekut jõudis Swift kahekümne kõige populaarsema programmeerimiskeele hulka [9].

3.3. Hübriidne lahendus

Hübriidne mobiilirakendus on rakendus, mida saab kasutada mitmel platvormil. Mobiilirakenduses kasutatakse hübriidset arendamist, kuna luua tuleb vaid üks koodibaas, mida ka pärast ülal pidada. Ühine koodibaas mitmele platvormile vähendab arendamisele kulutatud aega, kuna kahe erineva koodibaasi asemel on vaid üks. Populaarsed hübriidse mobiilirakenduse arendamise tööriistad on Flutter, React Native ning Ionic [2]. Siinse lõputöö raames sai hübriidse lahenduse arendusraamistikuks valitud Flutter kuna Flutter oli esimene hübriidne arendusraamistik mis internetiotsingutel ette tuli.

3.3.1. Flutter

Flutter on Google'i loodud tasuta ning avatud lähtekoodiga kasutajaliidese raamistik. Flutter'i raamistik koosneb kahest põhiosast: SDK ning Framework. SDK (Software Development Kit) ehk tarkvaraarenduskomplekt on tööriistade kogu, mida kasutades on võimalik luua mobiilirakendus korraga iOS ning Android platvormidele. Framework koosneb vidinatest (nupud, tekstiväljad vms), mida saab kasutada enda rakenduse valmistamiseks [10]. Flutter kompileerib Dart koodi *native*-teegiks, vastavalt Android *native* või iOS *native.* Loodud *native*-teegi põhjal ehitatakse üles rakendus vastavalt siis valitud platvormist (kas Android või iOS) [11].

3.3.2. Dart

Dart on kliendile optimiseeritud programmeerimiskeel, millega luuakse rakendusi igale platvormile. Antud programmeerimiskeele eesmärk on pakkuda võimalikult tulemuslikku arendamist mitmele platvormile [12].

3.4. Leadfellow

Leadfellow-ga läbirääkides selgus, et ettevõte vajab mobiilirakendust enda klientide mugavuseks. Mobiilirakendus peab olema lihtsalt kasutatav ning arusaadav. Järgnevalt kirjeldatakse üldiselt funktsionaalseid ning mittefunktsionaalsed nõudeid äiesmahus nõuded on toodud Lisas 1.

Funktsionaalsed nõuded

- Kasutajana soovin, et vaadete vahelt oleks võimalik liikuda.
- Kasutajana soovin, et sisselogimislehelt saaks suunduda teenusepakkuja lisamise lehele.
- Kasutajana soovin, et teenusepakkuja lisamise lehelt saaks suunduda tehingu info lehele.
- Kasutajana soovin, et sisselogimislehel saaks sisestada enda e-maili ning parooli sisenemiseks.

Mittefunktsionaalsed nõuded

- Rakenduse vaadete vahel liikumine ei kulutaks rohkem kui sekund aega
- Rakenduse funktsioonide (sisselogimine, teenusepakkujate lisamine, tehinguinfo saatmine) ei võtaks aega rohkem kui 3 sekundit minimaalse interneti kiiruse puhul (50 Mbps Statista andmetel [13]).

4. Meetod

Lõputöö praktiline osa koosnes mobiilirakenduste loomisest erinevates arendusraamistikes ning loodud mobiilirakenduste võrdlusest.

4.1. Arendusprotsess

Mobiilirakendus koosneb viiest vaatest: sisselogimisleht, registreerimisleht,unustatud parooli leht, teenusepakkujate lisamise leht ning tehingu informatsiooni leht. Lisaks vaadetele on vaja lisada ka klassid, mis haldavad sisselogimisandmeid, teenusepakkujate nimekirja ning "lead"-i ehk tehingu informatsiooni, mis saadetakse edasi teenusepakkujatele. Sellist üldist ülesehitust pakuvad kõik kolm rakendust. Kolm rakendust suhtlevad ühise API-ga. Rakenduste arendamiseks otsitakse välja õpetused, mille järgi rakendust luua. Esialgu luuakse rakendustele vaated ning üldine väljanägemine. Vaadetele lisatakse klassid, mis hoiustavad andmeid. Seejärel lisatakse erinevad meetodid, mis tegelevad vaadetevahelise liikumisega, parooli räsimisega ning API-le päringute tegemisega.

4.1.1. Suhtlus API-ga

Mobiilirakenduse kasutaja turvalisuse mõttes tuleb lisada ka parooli räsimine. Paroolide räsimiseks tuleb kasutada Blowfish algoritmi, kuna seda kasutab Leadfellow andmebaas. Viimaks on vaja rakendusele lisada ka API päringud, et andmebaasile andmeid saata ning andmebaasist andmeid kätte ka saada. Tegemist on REST API-ga, mis võtab vastu ning tagastab vastused JSON kujul. API päringud on loodud sisselogimiseks, kus päringuga on vaja kaasa saata e-mail ning räsitud parool ning päring tagastab õnnestumise korral sõnumi ("Successful login"), staatuse (1), tokeni ehk identifikaatori (pikk sõne, mida kasutatakse õnnestunid sisselogimise kontrolliks), e-maili millega sisse logiti ning aja, millal sisselogimine aegub.

4.1.2. iOS arendus

Apple iOS arendamiseks on vaja arvutit, mis jooksutaks MacOS-i. Lisaks on vaja ka Apple App Store-st allalaadida ka Xcode. Internetist uurides, mis keeles kirjutatakse iOS rakendusi, tuli vastuseks Swift, mistõttu sattus Swift valituks keeleks iOS arendamisel. Rakenduse arendamise

õppimiseks kasutasin internetist leitud õpetusi. Kasutades otsingut “Swiftui login page” leian õpetuse “How To Build A Login Page In SwiftUI #1 – Mastering Text Fields And Understanding @State“[2]. Leitud õpetuse põhjal alustan sisselogimislehe arendamisega. Järgnevad vaateid arendades võtan aluseks sisselogimislehe kuid vajadusel otsin lisaabi internetist.

4.1.3. Android arendus

Android rakendusi on võimalik arendada nii macOS, Windows kui ka Linux operatsioonisüsteemidega arvutitel, erinevalt iOS arendusest. Android rakenduse arendamiseks on vaja alla laadida Android Studio tarkvara. Arenduse alustamiseks uurin dokumentatsiooni ning internetiõpetusi Android rakenduse loomise kohta.

4.1.4. Hübriidse lahenduse arendus

Hübriidselt mobiilirakenduse arendamiseks läheb sarnaselt iOS arendamisele vaja arvutit, millel on macOS operatsioonisüsteem. Flutter süsteemi installeerimiseks sellisel viisil, et saaks arendada mobiilirakendust iOS ja Android süsteemidele korraga, tuleb allalaadida Xcode ning Android Studio, lisaks ka Flutter SDK. Flutter rakenduse arendamiseks alustan dokumentatsioonis leitava “Write your first Flutter app” õpetusega. Kasutades Flutter õpetust ning iOS õpetust alustasin arendamisega ning jooksvalt otsisin abi ka internetist.

4.2. Võrdlus

Siinse lõputöö praktilise osa üheks eesmärgiks on selgitada välja antud ettevõttele parima viis mobiilirakenduse loomiseks ning välja tuua arendusraamistikude erinevused, et tulevikus mobiilirakendust luues aidata õige arendusraamistiku valikuga. Võrreldakse on loodud koodi pikkust, valmis rakenduse ülesehitusaega, arendusteekide lisamist rakendusele ning algaja arendaja isiklikku arvamust arendusteekidest ning nende keerukusest.

[2] https://blckbirds.com/post/login-page-in-swiftui-1/

4.2.1. Koodi pikkus

Koodi pikkus on rakenduse arendamisel tähtis faktor, kuna mida pikem on kood, seda keerulisem on koodi tulevikus hallata. Selleks on olemas ka mugav tarkvara nimega Count Lines Of Code (cloc). Cloc tarkvara kasutamine on lihtne. Tuleb installerida tarkvara ning sisestada terminali kask "cloc /*projekti kausta asukoht arvutis*/" näiteks "cloc ~/user/desktop/kaust". Antud tarkvara loeb läbi kaustas olevad failid ning väljastab informatsiooni iga failitüübi kohta eraldi. Kuna cloc väljastab informatsiooni iga failitüübi kohta eraldi, on lihtne leida ridade arvu failidelt, mida kindlasti on muudetud. Lisaks arendaja enda kirjutatud koodile tekivad ka automaatsed failid projektidega kaasa. Selle vastu on cloc tarkvaral ka võimalus valida faile välja, mida lugeda ja mida mitte. Projekti süsteemifailid ei lähe arvesse, kuna arendaja neid enamjaolt ise ei muuda. Võrdluse käigus leitakse iga mobiilirakenduse koodipikkuse antud tarkvaraga ning koodipikkuseid võrredlakse omavahel.

4.2.2. Kompileerimisaeg

Valitud keeltes loodud mobiilirakendusi tehes, peab arvuti rakendust jooksutades seda enne ka kompileerima. Kompileerimine tähendab kirjutatud koodi teisendamist kasutatavaks rakenduseks. Kompileerimisaja mõõtmine toimub igas arenduskeskkonnas automaatselt, ehk Xcode ja Android Studio mõlemad väljastavad jooksutades informatsiooni kompileerimisaja kohta, selleks pole vaja kolmanda osapoole rakendusi kasutada. Android Studio puhul avaneb rakendust jooksutades Android Studio akna alumisse poolde "Run" aken, kuhu kompileerimise lõppedes tuleb kompileerimisaeg. Xcode puhul on Xcode akna vasakul küljel erinevad menüüd, sealhulgas projekti kaustade ülevaade. Menüüde ribalt kõige parempoolne valik avab akna, kuhu kompileerimise lõppedes väljustub kompileerimisaeg. Võrdluseks tuleb mõõta iga mobiilirakenduse kompileerimisaega 10 korda ning arvutada aritmeetiline keskmine. Seejärel aritmeelised keskmised tuleb omavahel võrrelda, et leida kiireim kompileerimisaeg.

4.2.3. Arendusteekide lisamine

Kuigi igat rakenduse aspekti on võimalik programmeerijal luua ise, on mugavam ja lihtsam vahepeal kasutada ettevalmistatud arendusteeke. Vastavalt funktsionaalsetele nõuetele, lisatakse

kõikidele rakendustele Blowfish algoritmi järgi räsimine ning JSON kujul andmebaasipäringute saatmine. Parooli räsimise ning JSON päringute saatmiseks lisatakse arendusteegid. Parooli räsides tehakse sisestatud paroolist valitud algoritmi abil räsi, mida on raske tuvastada. Räsimine tagab parooli ning see läbi kasutajaandmete turvalisuse. Blowfish algoritm oli loodud 1993 aastal Bruce Schneieri poolt, tasuta alternatiivina tol ajal olemasolevatele krüpteerimisalgoritmidele. Blowfish algoritm on patenteerimata, litsentsivaba ning tasuta kasutamiseks kõigile [14]. Võrdluse käigus uuritakse teekide lisamise üleüldist keerukust, ehk kui mitu sammu on vaja läbida teegi lisamiseks. Võrdlus toimub subjektiivselt võttes arvesse arendaja enda kogemust.

4.2.4. Arendaja kogemus

Lisaks tehnilistele aspektidele, uurib antud töö ka arendaja enda perspektiivi. Kuna siinse rakenduse arendaja on algaja ning pole eelnevalt varem mobiilirakendusi arendanud, on perspektiiv kasulik lugejale, kes pole varem arendanud mobiilseid rakendusi. Arendaja kogemuse uurimine on subjektiivne vaade arendusprotsessile ning tähelepanekutele,nii positiivsed kui negatiivsed, mis jäid arendajale arendusprotsessis silma. Peamine võrdlusaspekt on isiklik arvamus arenduskäigust erinevatel platvormidel. Tuleb võrrelda ka platvormide vahelisi erinevusi informatsiooni poolest. Ehk kui lihtne on vajadusel abi ja informatsiooni leida internetist.

5. Tulemused

Järgnevates peatükkides tuuakse välja meetodis püstitatud sammude lõplikku tulemust. Lisaks vaadeldakse ka läbiviidud võrdluste tulemusi.

5.1. Rakendus

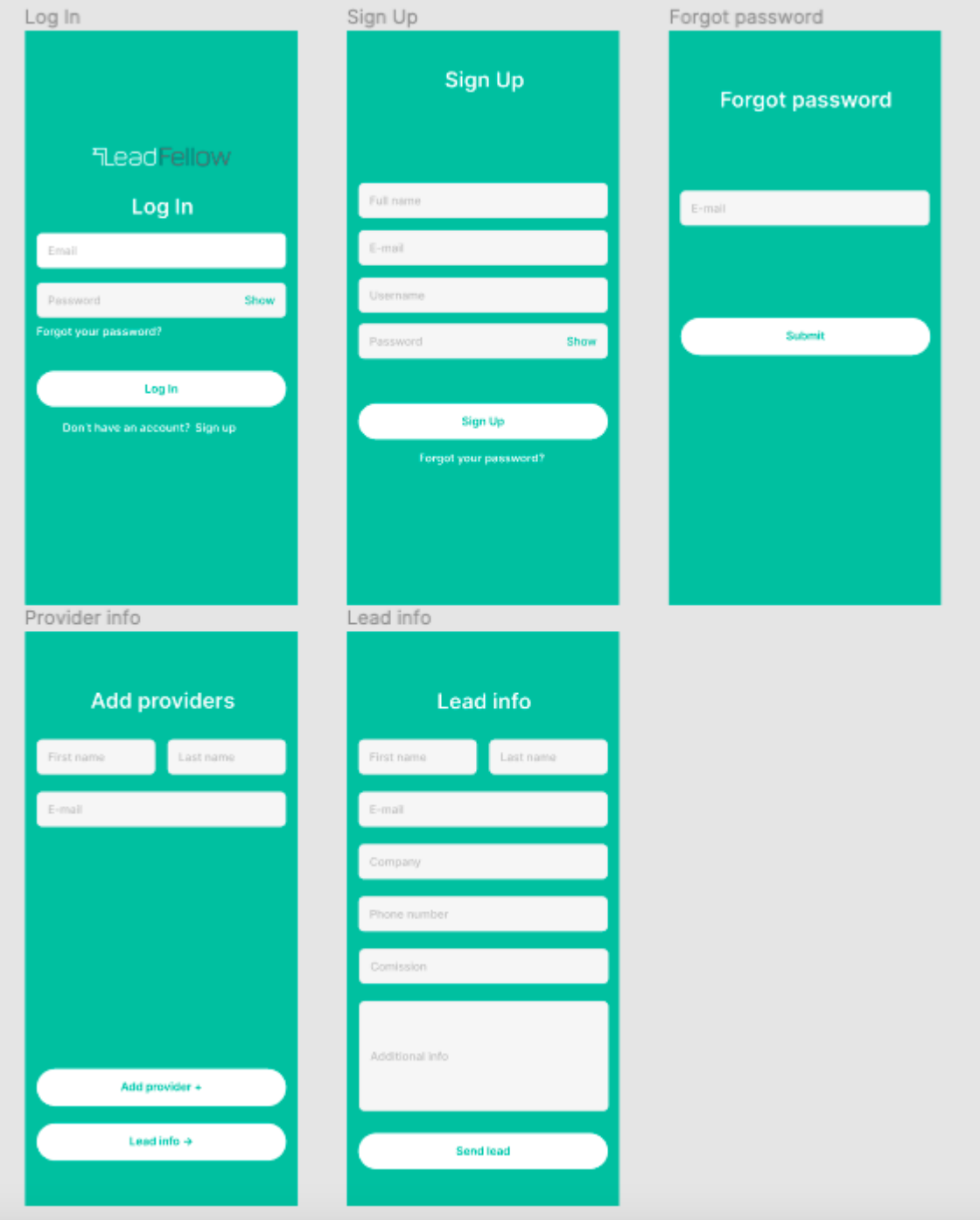
Järgnevates alapeatükkides uuritakse mobiilirakenduste arenduse tulemusi. Mobiilirakendused loodi ning võrreldi kasutades 2017 aasta Apple MacBook Air, millel on operatsioonisüsteemiks macOS Big Sur versioon 11.5.2. Arvuti protsessor on 1,8 GHz Dual-Core Intel Core i5. Vahemälu on 8 GB 1600 MHz DDR3 ning graafikakaart on Intel HD Graphics 6000 1536 MB.

5.1.1. Üldine arhitektuur

Mobiilirakendus on loodud Leadfellow kasutajatele, kes on soovitaja rollis. Läbi mobiilirakenduse saab rakenduse kasutaja saata teenusepakkujatele aktsiatehingu info. Mobiilirakendus on ühendatud Leadfellow andmebaasiga läbi PHP (PHP: Hypertext Processor) API-ga (*application programming interface* ehk rakendusprogrammiliides). Läbi API saab edastada andmebaasile päringud sisselogimiseks ning tehinguinfo saatmiseks. Allpool on näide API päringust, mis tegeleb sisselogimisega.

```
url: https://app.leadfellow.dev/api/login
{
        “email”: “xxx@xxx.xxx”,
        “password_hash”: “abcdefg…”
}
```

Mobiilirakendus on jaotatud viite vaatesse: sisselogimisleht, parooli unustamise leht, registreerimisleht, teenusepakkujate lisamise leht ning tehingu info leht. Joonisel 1 on näidatud vaadete välimus.

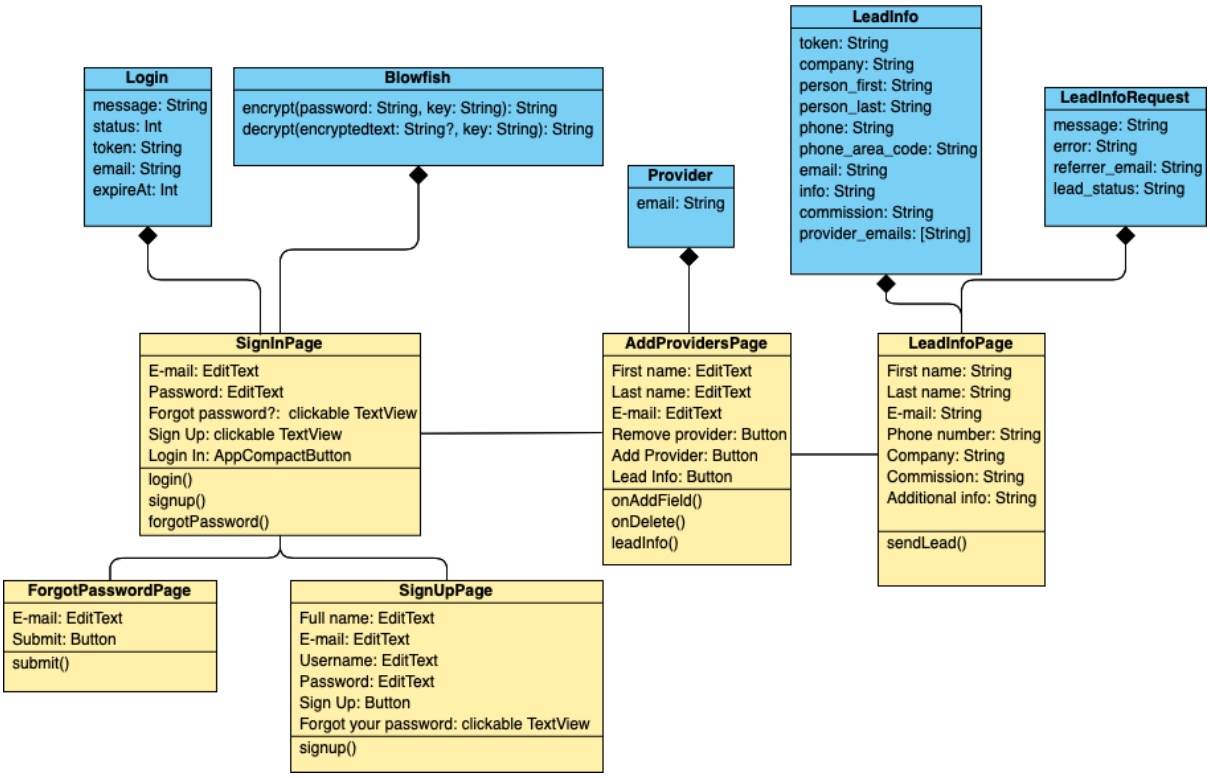


Joonis 1 Mobiilirakenduste vaated

5.1.2. Android

Androidi süsteemile loodud mobiilirakendus oli tehtud programmeerimiskeeles Kotlin Android Studio keskkonnas. Nagu eelnevalt oli mainitud, koosneb rakendus viiest vaatest:

sisselogimisleht(SignInPage), unustatud parooli leht (ForgotPasswordPage), registreerimisleht (SignUpPage), teenusepakkujate lisamise leht (AddProvidersPage) ja tehingu info leht (LeadInfoPage). Lisaks on loodud rakendusele ka klassid, mida vaated kasutavad andmete hoiustamiseks. Blowfish klass sisaldab meetodit parooli räsimiseks. E-maili ja räsitud parooliga sisselogides, tehakse API päring, mille vastuseks saadud info salvestatakse Login klassi. AddProvidersPage-l, kui lisatakse teenusepakkuja, salvestatakse ta meil ka eraldi Provider klassi. Tehingu info lehel täidetud väljad salvestatakse LeadInfo klassi, kuhu lisatakse ka eelnevalt loodud teenusepakkujate klassidest saadud e-mailide massiivi. LeadInfo klassi sisu saadetakse API päringuna edasi ning saadud vastus salvestatakse LeadInfoRequest klassi. Joonisel 2 on näha vaadete ning klasside vahelist suhtlust. Joonisel on kollaselt märgitud vaated ning siniselt klassid. Kotlinis, luues uue vaate tekib .kt fail ning .xml fail. Kotlin fail ehk .kt fail sisaldab endas nii-öelda vaate loogikat. Kotlin failis pannakse paika selles vaates toimuvad funktsiooni. Vaatega loodud teises failis ehk .xml failis toimub kogu välimuslik pool. Xml failis pannakse paika taustavärv, lisatakse erinevad tekstiväljad ning nupud, seadistatakse kõik vahemikud ning suurused. Arenduse käigus lähtusin sisselogimislehe loomis õpetusest, mille baasil lõin ka teised vaated. Jooksvalt tekkinud küsimustele leidsin vastused internetist, peamiselt StackOverflow-ist. Android rakenduse repo link: https://github.com/kimleadfellow/AppAndroid .

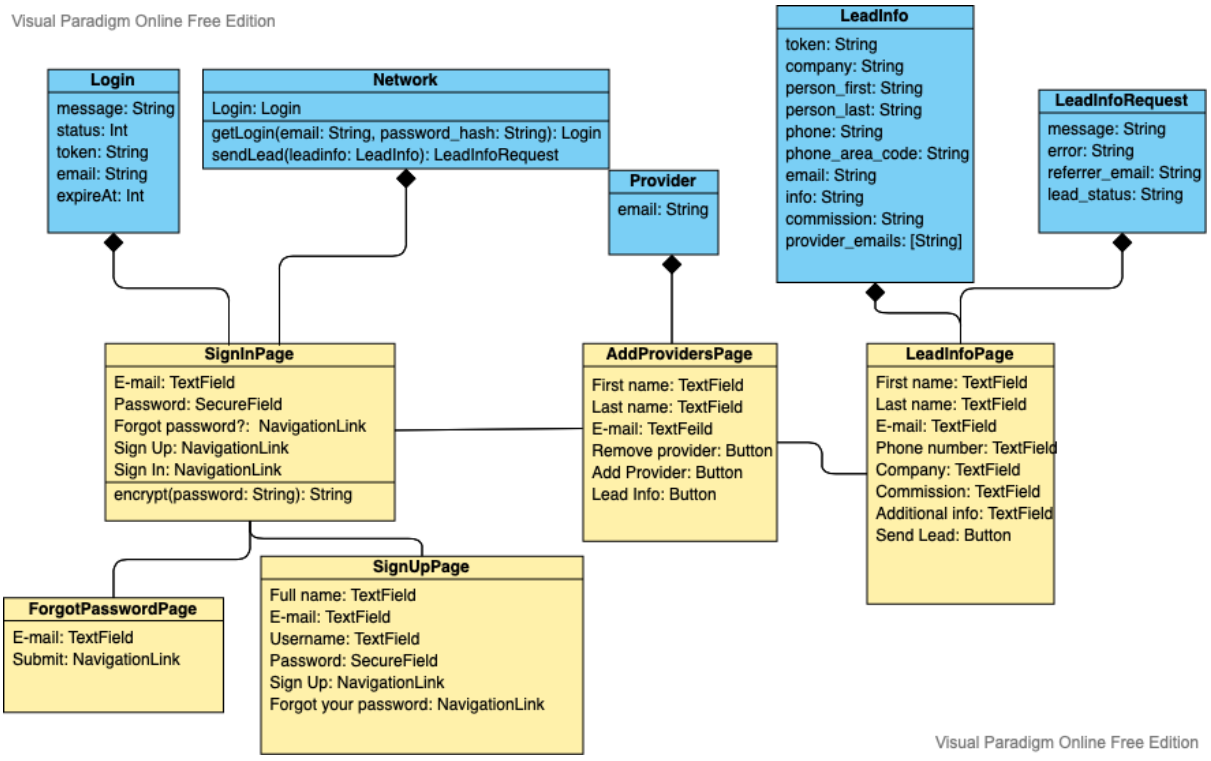


Joonis 2 Android rakenduse klassidiagramm

5.1.3. iOS

Apple iOS süsteemile loodud mobiilirakendus oli tehtud programmeerimiskeeles Swift Xcode keskkonnas. IOS Platvormile loodud rakendus koosneb viiest vaatest: SignInPage, ForgotPasswordPage, SignUpPage, AddProvidersPage, LeadInfoPage. SignInPage vaates toimub rakendusse sisselogimine. Sisselogides saadetakse sisestatud e-mail ja räsitud parool läbi API andmebaasi, kust tagastatud info salvestatakse Login klassi (message: sõnum sisselogimise õnnestumise/ebaõnnestumise kohta, status: numbriline staatus sisselogimise õnnestumisest, token: pikk sõne, mida kasutatakse sisselogimise autentimiseks, email: e-mail millega on sisse logitud, expireAt: numbriline näitaja, mis annab teada kaua sisselogimine kehtib). SignInPagega on ühendatud ka Network klass. Network klass hoiustab endas getLogin() meetodit, mida kasutades saadetaksegi sisselogimisinfo API kaudu andmebaasi ning vastus salvestatakse Login klassi. SignInPagelt saab liikuda ForgotPasswordPagele ning SignUpPagele vajutades vastavat nuppu. SignInPagelt sisselogides avaneb järgmisena AddProvidersPage, kuhu saab sisestada teenusepakkujate nimed ning e-mailid, ning vajadusel teenusepakkujaid lisada juurde. Iga lisatud

teenusepakkuja salvestatakse Provider klassi. Kui soovitud arv teenusepakkujaid on lisatud, saab vastavat nuppu vajutades edasi liikuda leheküljele LeadInfoPage. Leheküljel LeadInfoPage saab täita tehingu informatsiooni, mida saadetakse API kaudu edasi andmebaasi. Läbi API saadetakse andmebaasi järgneva informatsiooni: tehingu pakkuja ees- ja perekonnanimi, tehingu pakkuja e-mail, tehingu pakkuja mobiilitelefoninumber, tehingu pakkuja telefoninumbri suunakood, tehingu pakkuja ettevõtte nimi, vahetasu, mida soovitaja soovib ning lisainfo. Saadetav info salvestatakse LeadInfo klassi. Andmebaasilt saab läbi API vastuseks info, mida salvestatakse LeadInfoRequest klassi. Vastusena saadab API järgneva info: sõnumi tehingu kohta, juhul kui tekib mingi viga ka error sõnumi, soovitaja e-maili, tehingu staatuse. IOS rakenduse vaadete ning klasside suhtlust on näha joonisel 3, kus kollaselt on märgitud vaated ning siniselt on märgitud klassid.



Joonis 3 iOS rakenduse klassidiagramm

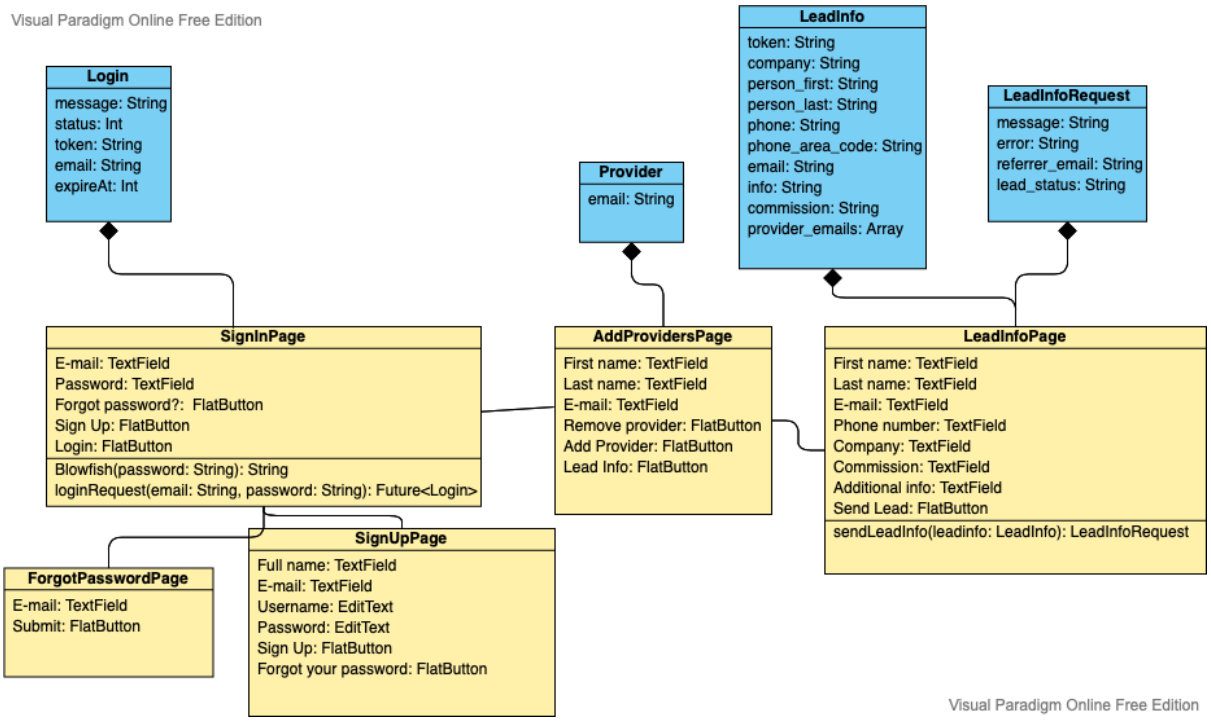
Rakenduse arendus algas internetiõpetuste jäglimisest. “How To Build A Login Page In SwiftUI #1 – Mastering Text Fields And Understanding @State“[3] järgi lõin esialgse SignInPage, mille baasil said loodud ka teised vaated. Erinevalt Android arendusest, toimub nii loogika kui ka välimuslik pool ainult .swift failides. Erinevalt Android arendamisest, Swift rakendus koosneb

[3] https://blckbirds.com/post/login-page-in-swiftui-1/

lisaks klassidele ka struktuurides ehk *struct*-idest. Iga *struct* on nii-öelda objekt (näiteks UsernameTextField on *struct*, mis väljastab ekraanile tekstivälja). Erinevate "objektide" *struct*-e lisatakse vaate peastruktuuri. iOS rakenduse repo link: https://github.com/kimleadfellow/AppIos .

5.1.4. Hübriid

Hübriidselt ehk nii Android kui iOS süsteemile loodud rakendus oli tehtud programmeerimiskeeles Dart Android Studio keskkonnas kasutades Flutter raamistikku. Flutter rakendus koosneb viiest vaatest: SignInPage, ForgotPasswordPage, SignUpPage, AddProvidersPage, LeadInfoPage. Rakendust jooksutades, avaneb esialgu SignInPage. SignInPagel on väljad e-maili ja parooli jaoks ning nupud sisselogimiseks, ForgotPasswordPagele liikumiseks ning SignUpPagele liikumiseks. SignInPage failis on ka meetodid Blowfish algoritmiga parooli räsimiseks ning sisselogimis API päringu tegemiseks. Sisestatud e-mail ning räsitud parool saadetakse "Login" nupule vajutades läbi API andmebaasi, mille tagastatud vastus salvestatakse Login klassi. Login klassi salvestatakse järgnev info: message: sõnum sisselogimise õnnestumise/ebaõnnestumise kohta, status: numbriline staatus sisselogimise õnnestumisest, token: pikk sõne, mida kasutatakse sisselogimise autentimiseks, email: e-mail millega on sisse logitud, expireAt: numbriline näitaja, mis annab teada kaua sisselogimine kehtib. Sisse logides avaneb AddProvidersPage, kuhu saab sisestada teenusepakkuja andmed ning vajadusel ka lisada teenusepakkujaid, kellele tehinguinfo saata. Kui soovitud arv teenusepakkujaid on lisatud, avaneb vastavat nuppu vajutades LeadInfoPage. LeadInfoPagel täidetud väljad salvestatakse LeadInfo klassi. LeadInfo klassi andmeid kasutades saadetakse vastavat nuppu vajutades informatsioon läbi API andmebaasi, pärast mida salvestatakse päringu vastus LeadInfoRequest klassi. Joonisel 4 on välja toodud rakenduse vaadete ning klasside suhted, kus kollaselt on märgitud vaated ning siniselt klassid. Flutter rakenduse vaate ülesehitus koosneb *widgetitest* ehk vidinatest. Vaate keskmeks on Scaffold vidin, mille sees on erinevad vidinad, mis loovad lõpliku vaate. Iga vidina välimus deklareeritakse vidina enda sees, ehk värvus, kirjasuurus jms. Hübriidse rakenduse repo link: https://github.com/kimleadfellow/AppFlutter .



Joonis 4 Hübriidse rakenduse klassidiagramm

5.2. Võrdlus

Järgnevates alampeatükkides esitatakse rakendustevaheliste võrdluste tulemusi. Võrreldi koodipikkust, kompileerimisaega, arendusteekide lisamist ning arendaja enda kogemust.

5.2.1. Koodipikkus

Kasutades cloc (Count Lines Of Code) tarkvara leidsin, et Swift koodi pikkus on 1466 rida, Kotlin koodi pikkus on 925 rida ning Dart koodi pikkus on 704 rida. Kokkuvõttes, kui rakendust luua eraldi kahele platvormile, tekib antud kontekstis 1687 rida koodi rohkem.

5.2.2. Kompileerimisaeg

Kompileerimisaega saab näha igas arenduskeskkonnas rakendust kompileerides. Mõõtmiseks ei ole vaja lisada ühtegi kolmanda osapoole teeki. Mobiilirakendusi arendati ning käivitati 2017 aasta Macbook Air arvutis. Mõõtmiseks kompileeriti igat rakendust 10 korda ning arvutati aritmeetiline keskmine. Kõige kiirem kompileerimisaeg oli puhtalt iOS platvormile arendatud rakenduse

kompileerimisaeg, ajaks oli keskmiselt 45 sekundit. Puhtalt Android süsteemile arendatud rakendus võttis kompileerimiseks keskmiselt 50 sekundit aega. Hübriidset lahendust kontrolliti nii Android süsteemil kui ka iOS süsteemil. Hübriidse rakenduse Android versiooni kompileerimisaeg võttis keskmiselt 55 sekundit, ning sama rakenduse iOS versiooni peale kulus keskmiselt 56 sekundit.

5.2.3. Arendusteekide lisamine

Apple iOS arenduse jaoks arendusteekide lisamine koosneb mitmest sammust, kuid neid ei pea iga kord kordama. IOS arenduseks on erinevaid arendusteekide haldusviise: CocoaPods, Carthage, Swift Package Manager. Siinse rakenduse jaoks sai valitud CocoaPods kuna krüpteerimisteeki lisades oli juhistes kasutatud just CocoaPodsi. Esialgu on vaja arvutisse installeerida CocoaPods. Selleks on vaja arvuti terminalis jooksutada järgnev käsk “sudo gem install cocoapods”. Seejärel on vaja luua projekti kaustas pod-file, mis haldab erinevaid teeke, mida võidakse lisada. Joonisel 5 on näidatud, milline näeb välja CocoaPods .pod fail.

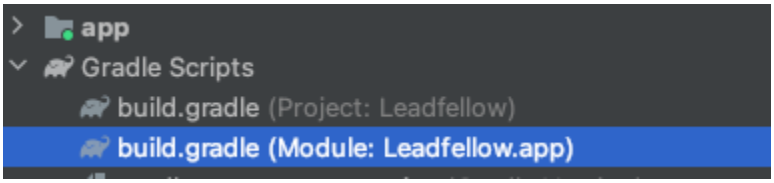
Podfile

```
platform :ios, '9.0'

target 'MobileApp' do
  pod "BCryptSwift"
  target 'MobileAppTests' do
    inherit! :search_paths

  end

  target 'MobileAppUITests' do
  end

end
```

Joonis 5 CocoaPod fail

Et lisada projektile soovitud teek, on vaja lisada pod-file-i rida “pod ____” ehk siinses näites “pod “BCryptSwift”. Loodud fail salvestatakse projektikausta, ning arvuti terminalis suundutakse samuti projekti kausta. Kui terminal on projekti kaustas (“cd projekti/kausta/asukoht/arvutis” - käsk terminalis kaustade vahel liikumiseks) jooksutatakse käsk “pod install”, mis loob uue App.xworkspace faili. App.xworkspace faili jooksutades avaneb arenduskeskkond, mis on samasugune nagu enne teekide lisamist, kuid nüüd saab projektis kasutada ka lisatud teeke. Uute teekide lisamiseks on vaja vaid lisada pod faili rida “pod teeginimi” ning jooksutada “pod install” käsk.

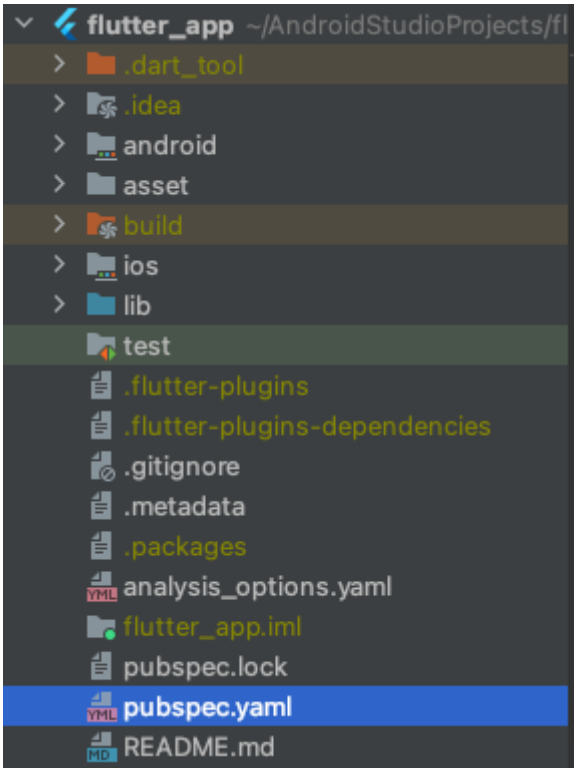
Android süsteemil teekide lisamiseks on vaja avada projektis build.gradle(Module) fail, mis asub gradle scripts all. Joonisel 6 on märgitud, kus Android Studio IDE-s leiab build.gradle faili.



Joonis 6 Android Studio build.gradle faili asukoht

Antud failis tuleb lisada “dependency” lõiku rida enda soovitud teegiga. Näiteks “implementation 'com.android.volley:volley:1.2.1'”. Uue teegi lisamisel tekib ka kohe teade, et kogu projekt üles ehitada. Pärast ülesehitamist on teek lisatud ning saab seda enda koodis ka kasutada.

Hübriidselt toimub teekide lisamine sarnaselt Android süsteemile. Teegi lisamiseks on vaja avada pubsec.yaml fail. Joonisel 7 on märgitud, kus asub pubsec.yaml fail Android Studio arenduskeskkonnas.



Joonis 7 Android Studio pubsec.yaml faili asukoht

Avatud failis tuleb lisada teegi nimi ning versiooni. Näiteks “dbcrypt: ^2.0.0”. Pärast lisamist tuleb teade, et üles ehitada kogu projekt uuesti ning pärast ülesehitust ongi teek kasutusvalmis projektis.

5.2.4. Arendaja kogemus

Antud rakenduse puhul ei tekkinud õnneks ka küsimusi, mida oleks vaja uurida internetist, kuid juhul kui tulevikus tekivad mõned küsimused, ei pruugi hübriidsel lahendusel olla nii palju vastuseid kui puhtalt Android või iOS lahendustel. Androidi Kotlin ning iOS Swift on vanemad ning populaarsemad programmeerimiskeeled kui Dart, mistõttu leiab internetist vähem informatsiooni selle kohta. Enamus küsimuste jaoks on siiski kõigil olemas ka dokumentatsioon. Flutteriga tekkis ka ülekuumenemise probleem mitmel korral. Flutter rakendust arendades kuumenes arvuti mitmel korral üle ning pidi ajutiselt arvuti välja lülitama ja laskma sellel puhata. Internetist uurides ei leidnud, et tegu oleks levinud probleemiga, kus Flutter ning Flutteri rakendust jooksutav simulaator põhjustaksid arvuti ülekuumenemist.

6. Arutelu

Flutter-is loodud rakendus on lühem ehk vajab vähem kirjutamist ning hiljem ka vähem ülalpidamist. Märkimisväärseim erinevus Android arenduse ning iOS ja hübriidse arenduse vahel oli vaadete loomisel tekkinud .xml failid. Võrreldes Android *native* arendusega, nõuab Flutter arendus vähem nii öelda standartkoodi, ehk koodi, mida on vaja lisada ainult selleks, et kõik töötaks. Näiteks ühe nupu loomiseks ning sellele funktsionaalsuse lisamine on Androidi puhul rohkem koodi, mida kirjutada. Android rakendusel on vaja .xml faili lisada soovitud nupp ning samas failis selle välimuse meelepäraselt seadistada. Seejärel on vaja nupp deklareerida vaate klassis, kuhu nupp lisati. Deklareeritud nupule on vaja lisada funktsionaalsus. Flutteril sama tulemuse jaoks on vaja lisada vaate .dart faili nupu vidin, ning vidinale lisama meelepärased funktsionaalsused ja välimuse. Selle aspekti poolest on mõistlikum kasutada hübriidset lahendust, kuna koodi mida kirjutada ning hiljem hallata on märkimisväärselt vähem.

Kompileerimisaja arvestuses oli hübriidne lahendus aeglasem kui *native* kuid kompileerimisaja vahe oli nii väike, et lõppotsust see ei mõjutanud. Rakenduse kompileerimine on koodi teisendamine rakenduseks, mistõttu võtab hübriidse rakenduse kompileerimine natukene rohkem aega. Kui native rakendused peavad enda kindlat koodi kompileerima enda kindla platvormi peale, peab hübriidne rakendus ühte koodi kompileerima vastavalt vajadusele mingile platvormile. Selleks peab ta lisaks teisendama koodi õigele kujule, et valitud platvormil töötaks.

Arendusteekide lisamine on üldiselt kõikide arendusraamistikude puhul arusaadav ning lihtne, kuid iOS teekide lisamine nõuab veidi rohkem tööd. Võrreldes iOS teekide lisamisega, on Android ning hübriidsel lahendusel teekide lisamine lihtsam. Enne valitud “BCryptSwift” arendusteegi lisamist, proovisin kasutada arendusteeki “cryptoswift”. “Cryptoswift” arendusteegi lisamiseks kasutasin Swift Package Manageri. Swift Package Manager on Xcode-i sisseehitatud arendusteekide lisamise tööriist. Swift Package Manageri kasutamine on lihtne: File -> Add Package… -> sisestada teegi nimi või URL ning vajutad “Add Package”. Sõltuvalt vajaminevast arendusteegist, on võimalik ka kasutada algajale sõbralikumat Swift Package Manageri, kuid siinses olukorras oli vaja kasutada arendusteeki “BCryptSwift”, mistõttu tuli kasutusele võtta CocoaPods. Androidile ja hübriidsele lahendusele arendusteekide lisamine nõuab vähem samme

kui iOS lahendusele lisamine, mistõttu võib öelda, et Androidile ja hübriidsele lahendusele on arendusteeke lihtsam lisada.

Üheks võimalikuks mõjutavaks faktoriks võib tulemuste ning lõpliku töö puhul olla fakt, et tegu oli üksiku arendajaga, kel pole varasemat kokkupuudet mobiilsete rakenduste arendamisega. Algaja arendajal kulub aega arendusraamistiku ning üleüldise mobiilse rakenduse arendamise õppimisele.

Ajapuuduse tõttu jäid lisamata ka testid rakendusele. Siiski sai loodud rakendusi iga sammu tagant manuaalselt testitud, mis tagasid nõuetele vastavuse kontrollimist. Vajaliku riistvara puudumise tõttu ei olnud võimalik kontrollida rakenduste kasutamist päris nutitelefonis, kõik jooksutamised olid tehtud arvutis läbi simulaatori. Arendajal oli vaid Android nutitelefon kuid Apple iOS nutitelefoni pole, mistõttu ei saa võrrelda kõiki rakendusi ning päris nutitelefoniga võrdlemist ei lisatud.

Kuna lõputöö praktilise osa käigus oli tegu nii õppimise kui arendamisega, ei ole mobiilirakendus lõplikult valmis. Rakenduse poolt loodud parooliräsid ei vasta andmebaasile, mistõttu sisselogimine ei tööta õigesti. Rakendus ise ühendub andmebaasiga, mida sai testitud kasutades etteantud räsi.

Kuigi üldiselt rakenduse loomiseks valin isiklikult ja subjektiivselt Flutteri hübriidse arenduse, tekkis hübriidse lahendusega ka probleeme. Kindlat põhjust antud probleemile ei leidnud ning internetist uurides ei tundunud probleem olevat ka väga levinud.

Kui sarnast rakendust valmistada, siis isiklikult soovitan kasutada Flutterit või muud hübriidse lahenduse varianti, kuna võrdlemisi suuremate ja keerulisemate rakendustega on tegu üpriski lihtsa rakendusega, saab hübriidse lahendusega täpselt sama tulemuse lihtsama vaevaga.

7. Kokkuvõte

Selle bakalaureusetöö eesmärk oli luua ning võrrelda mobiilirakendusi, mis olid loodud kasutades erinevaid arendusteeke. Töö teoreetiline osa andis ülevaate kasutatud arendusteekidest ning tehnoloogiatest, rakenduse enda olemusest, varasemalt tehtud uuringutest ning ettevõttest, millele mobiilirakendus oli loodud.

Töö käigus loodi samasugust mobiilirakendused kolmel viisil: iOS *native* kasutades arenduskeskkonda Xcode ning programmeerimiskeelt Swift, Android *native* kasutades arenduskeskkonda Android Studio ning programmeerimiskeelt Kotlin ning hübriidne lahendus kasutades arenduskeskkonda Android Studio ning programmeerimiskeelt Dart. Loodud rakenduste vahel võrreldi koodi pikkust, kompileerimisaega, arendusteekide lisamist ning arendaja enda kogemust. Võrdluse käigus tuli välja, et kirjutatud kood oli hübriidsel lahendusel kõige lühem. Mõõdetud kompileerimisaeg oli kõige kiirem iOS arendusel, teisel kohal oli Android ning kolmandal kohal hübriidne lahendus. Kuid kompileerimisaegade erinevus oli vaid mõni sekund, mistõttu ei mänginud see lõppotsuses olulist rolli. Arendusteekide lisamine, kuigi erinev, ei olnud ühegi puhul kindlalt parem või halvem, kuna tegu on suhteliselt isikliku arvamusega. Arendaja enda kogemus töö käigus kajastab, et isikliku arvamusena on hübriidne lahendus algajale parem.

Töö tulemusena saab väita, et hübriidne lahendus on sarnase rakenduse loomiseks mõistlikum variant.

Viidatud kirjandus

[1] Turner, Ash 2021. How many smartphones are in the world? https://www.bankmycell.com/blog/how-many-phones-are-in-the-world (01.12.2021)

[2] Olsson, Matilda. A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development. Blekinge Institute of Technology, Faculty of Computing, Department of Software Engineering, 2020.

[3] Flutter vs Native: What is Best for Your Project? https://surf.dev/flutter-vs-native/ (31.04.2022)

[4] Dart / Flutter vs. Swift / Native iOS – which one is better in 2021? https://itcraftapps.com/blog/dart-flutter-vs-swift-native-ios-which-one-is-better/ (31.04.2022)

[5] Android. https://developer.android.com/guide/platform (24.11.2021)

[6] Integrated Development Environment (IDE) - Android Studio. https://developer.android.com/studio/intro (24.11.2021)

[7] Heller, Martin 2020. What is Kotlin? The Java alternative explained. https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html (24.11.2021)

[8] Chung, Eddy. What is Xcode and why do I need it? (24.11.2021)

[9] Castor, Fernando; Ebert, Felipe; Pinto, Gustavo; Rebouças, Marcel; Serebrenik, Alexander; Torres, Weslley. An Empirical Study on the Usage of the Swift Programming Language. USA: Institute of Electrical and Electronics Engineers, 2016,

[10] What is Flutter and Why You Should Learn it in 2020. https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/ (21.05.2022)

[11] Frequently Asked Questions https://docs.flutter.dev/resources/faq (07.05.2022)

[12] Dart. https://www.dart.dev/overview (02.02.2022)

[13] Average download speeds for internet connection in Estonia as of December 2020, by type (in Mbps) https://www.statista.com/statistics/1156632/internet-connection-speeds-estonia/

[14] The Blowfish Encryption Algorithm. https://www.schneier.com/academic/blowfish/ (23.03.2022)

Lisad

1. Nõuded

Funktsionaalsed nõuded

Sisselogimisleht

- Kasutajana soovin, et rakenduse esmakordsel avamisel oleks esimene leht sisselogimisleht, et

esimese asjana rakendust avades saaks siseneda enda kontosse.

- Kasutajana soovin, et kui rakendus on vahepeal taustale jäänud (kuid pole täiesti suletud), avaneks viimane aken, mis oli lahti.
- Rakenduse tellijana soovin, et sisselogimislehel oleks "Leadfellow" logo, kiri "Log In".
- Kasutajana soovin, et sisselogimiseks oleks sisselogimislehel kasutajanime väli,parooli väli ning sisselogimisnupp.
- Kasutajana soovin, et parooli unustamise puhul oleks nupp "Forgot your password?", mis viib parooli lähtestamiseks mõeldud lehele, juhul kui parool on ununenud või vajab uuendamist.
- Kasutajana soovin, et uue kasutaja loomiseks oleks registreerimisnupp, mis viib registreerimislehele, juhul kui pole veel kasutajat olemas või on vaja uut kasutajat teha.
- Kasutajana soovin, et kasutajanime/e-maili väljale vajutades saab sisestada

emaili.

- Kasutajana soovin, et oleks kindel nupp, et teha uut kasutajat, juhul kui kasutajat

pole veel tehtud või on vaja uut kasutajat teha.

- Kasutajana soovin, et sisselogides andmetega, mis pole veel "Leadfellow" andmebaasi registreeritud, tuleb vastav sõnum ka ette.
- Kasutajana soovin, et kui kasutatud parool ei ole sama, mis andmebaasis antud

kasutajanimel on, tuleb sisselogimiskatsel vastav sõnum ette.

- Kasutajana soovin, et kui kasutajanimi ning parool on õiged, avaneb "Add providers"

leht.

Unustatud parooli leht

- Kasutajana soovin, et parooli unustamise lehel oleks tagasi mineku nupp, juhul kui ei ole lõpuks vaja ikka parooli muuta.
- Kasutajana soovin, et oleks e-maili väli, millele saata täiendav informatsioon parooli lähtestamise kohta.
- Kasutajan soovin, et oleks kinnitusnupp, mida vajutades saadetakse informatsioon sisestatud e-mailile.
- Kasutajana soovin, et e-maili väljale vajutades saab sisestada e-maili. Kui e-maili ei ole "Leadfellow" andmebaasi registreeritud, siis kinnitusnuppu vajutades kajastub ka vastav sõnum.
- Kasutajan soovin, et kui e-mail on "Leadfellow" andmebaasi registreeritud, saadetakse sisestatud e-mailile järgnevad sammud parooli vahetamiseks.
- Kasutajana soovin, et tagasi mineku nuppu vajutades, avaneb uuesti sisselogimisleht.

Registreerimisleht

- Kasutajana soovin, et registreerimislehel oleks tagasi mineku nupp, täisnime väli, e-maili väli, kasutajanime väli, parooli väli ning registreerimise nupp.
- Kasutajana soovin, et tagasi mineku nuppu vajutades, avaneb uuesti sisselogimisleht.
- Kasutajana soovin, et täisnime väljale vajutades saab sisestada kasutaja täisnime (eesnimi ja perekonnanimi).
- Kasutajana soovin, et e-maili väljale vajutades saab sisestada e-maili.
- Kasutajana soovin, et kasutajanime väljale vajutades saab sisestada kasutajanime.
- Kasutajana soovin, et parooli väljale saab sisestada parooli.
- Kasutajana soovin, et kui sisestatud e-mail on andmebaasis juba olemas, registreerimisnupule vajutades kajastub vastav sõnum ning registreerimine ei lähe läbi.
- Kasutajana soovin, et kui sisestatud kasutajanimi on andmebaasis juba olemas, registreerimisnupule vajutades kajastub vastav sõnum ning registreerimine ei lähe läbi.
- Kasutajana soovin, et kui kõik väljad on õieti täidetud, registreerimisnupule vajutades luuakse uus kasutaja "Leadfellow" andmebaasi ning avaneb uuesti sisselogimisleht.

Teenusepakkuja lisamise leht

- Kasutajana soovin, et “Add providers” lehel on tagasi mineku nupp, väljade plokk (eesnime väli, perekonnanime väli, e-maili väli), “Add providers” nupp ning “Lead info” nupp.
- Kasutajana soovin, et eesnime väljale vajutades saab sisestada eesnime.
- Kasutajana soovin, et perekonnanime väljale vajutades saab sisestada perekonnanime.
- Kasutajana soovin, et e-maili väljale vajutades saab sisestada e-maili.
- Kasutajana soovin, et “Add providers” nupule, lisandub veel üks plokk samasuguseid väljasid. Maksimaalselt saab olla kuni 10 plokki.
- Kasutajana soovin, et “Lead info” nupule vajutades avaneb “Lead info” leht.
- Kasutajana soovin, et tagasimineku nupule vajutades avaneb sisselogimisleht.

Tehingu info leht

- Kasuatjana soovin, et “Lead info” lehel on tagasi mineku nupp, mis avab “Add providers” lehe, kus on eelnevalt sisestatud informatsioon veel alles, et seda muuta vajadusel.
- Kasutajana soovin, et “Lead info” lehel oleks eesnime väli, perekonnanime väli, e-maili väli, firma nime väli, telefoninumbri väli, lisainfo väli.
- Kasutajan soovin, et “Lead info” lehel oleks “Send lead” nupp, mis saadab täidetud informatsiooni “Add providers” lehel sisestatud isikutele.
- Kasutajana soovin, et eesnime väljale vajutades saab sisestada eesnime.
- Kasutajana soovin. et perekonnanime väljale vajutades saab sisestada perekonnanime.
- Kasutajana soovin, et e-maili väljale vajutades saab sisestada e-maili.
- Kasutajana soovin, et firma nime väljale vajutades saab sisestada firma nime.
- Kasutajana soovin, et telefoninumbri väljale vajutades saab sisestada telefoninumbrit.
- Kasutajana soovin, et lisainfo väljale vajutades saab sisestada misiganes teksti.

Mittefunktsionaalsed nõuded

- Kasutajana soovin, et rakenduse avamine võtaks maksimaalselt 1500ms aega.
- Kasutajana soovin, et parooliunustamise lehe avamine võtaks maksimaalselt 500ms aega.
- Kasutajana soovin, et registreerimislehe avamine võtaks maksimaalselt 500ms aega.
- Kasutajana soovin, et tekstiväljad võtaksid vastu vaid tähti ning sidekriipse. Numbrid ei ole lubatud (v.a. “Lead info” lehel “Additional Info” väli)
- Kasutajana soovin, et parooliväljad oleksid varjatud.
- Kasutajana soovin, et telefoninumbri väli võtaks vasti vaid numbreid.
- Kasutajana soovin, et “Add provider” nupule vajutamise ja ploki lisamise vahel läheks maksimaalselt 300ms aega.
- Kasutajana soovin, et lehtede vahel liikumine ei võtaks rohkem aega kui 500ms.
- Andmebaasi administraatorina soovin, et andmebaasile saadetud paroolid oleks Blowfish algoritmiga räsitud.

2. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Erik Kim,

(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

Kuidas mõjutab mobiiliäpi arendusraamistiku valik arendusprotsessi ja lõpptoodet? Juhtumiuuring,

(lõputöö pealkiri)

mille juhendaja Kristiina Rahkema,

(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Erik Kim

10.05.2022