VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO OBJEDNÁVANÍ DROBNÝCH SLUŽEB

MOBILE APPLICATION FOR ORDERING SMALL SERVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **PETER VICAN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LUKÁŠ SEMERÁD**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20425/2017/xvican02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Vican Peter**
Obor: Informační technologie
Téma: **Mobilní aplikace pro objednávání drobných služeb**
Mobile Application for Ordering Small Services
Kategorie: Informační systémy

Pokyny:

1. Nastudujte platformu Android a technologii Google Firebase.
2. Navrhněte aplikaci, která bude sloužit jako platforma pro objednávání drobných služeb. Poskytovatel svoji službu vystaví pro zákazníky, kteří si ji budou přes mobilní zařízení objednávat. Obdobně zákazník vystaví požadavek na drobnou službu, kterou bude moci poskytovatel následně obsadit. Návrh aplikace proveďte ve standardu UML 2. Zejména jej podpořte diagramem případu užití, tříd a sekvenčním diagramem.
3. Implementujte aplikaci na platformě Android. Na implementaci serverových prvků aplikace, kterými jsou zejména databázová vrstva a komunikační rozhraní, využijte platformu Google Firebase.
4. Aplikaci otestujte na reálném případu objednávky konkrétní služby a na případech užití definovaných v bodě 2.

Literatura:

- ROGERS, R., LOMBARDO, J., MEDNIEKS, Z., MEIKE, B. *Android application development*. 1st ed. Sebastopol, Calif.: O'Reilly, 2009. ISBN 05-965-2147-2.
- SCHILDT, H. *Java: the complete reference*. Ninth edition. New York: McGraw-Hill Education, 2014. ISBN 978-007-1808-552.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Semerád Lukáš, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto bakalárska práca sa zaoberá vývojom mobilnej aplikácie pre objednávanie drobných služieb. Aplikácia je vyvíjaná pre mobilnú platformu Android. Mobilná aplikácia je písaná podľa najnovších trendov pomocou jazyka Kotlin a cloudového riešenia Google Firebase. V prvej časti sú definované špecifikácie s návrhom mobilnej aplikácie, druhej časti je predstavená platforma Android a posledná časť je venovaná samotnej implementácii a testovaniu.

Abstract

This bachelor thesis deals with the development of a mobile application for ordering small services. The application is developed for the Android mobile platform. The mobile application is written according to the latest trends using Kotlin and the Google Firebase cloud solution. The first part defines mobile application design specifications, the second part introduces Android platform and the last part is about implementation and testing

Kľúčové slová

Android, mobilná aplikácia, mobilné zariadenie, TouchDeal, RxJava, Kotlin, Firebase

Keywords

Android, mobile app, mobile device, TouchDeal, RxJava, Kotlin Firebase

Citácia

VICAN, Peter. *Mobilní aplikace pro objednávaní drobných služeb.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Semerád

Mobilní aplikace pro objednávaní drobných služeb

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením Ing. Lukáša Semeráda. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Peter Vican
17. mája 2018

Poďakovanie

Týmto by som sa chcel poďakovať svojmu vedúcemu práce, pánovi Ing. Lukášovi Semerádovi, za odbornú pomoc pri písaní bakalárksej práci.

Obsah

Kapitola 1

Úvod

V dnešnej modernej dobe už skoro každý človek vlastní aspoň jedno chytré zariadenie v podobe mobilu, tabletu alebo chytrých hodiniek. Preto mnohé firmy či samostatní vývojári prichádzajú s novými nápadmi na aplikácie, ktoré by zaujali širokú škálu používateľov týchto zariadení a zjednodušili alebo spríjemnili im ich život. Celosvetová najpoužívanejšia platforma pre inteligentné zariadenia je Android. Preto som sa rozhodol svoju aplikáciu postaviť na tejto platforme.

Mojim novým nápadom pre svet Android je aplikácia TouchDeal, ktorá by mala svojím používateľom sprostredkovať dohodu ohľadne drobných služieb a to v podobe prevozu, vyzdvihnutia alebo nakúpenia vecí. V dobe, keď táto myšlienka vznikla, tak na trhu nebola aplikácia, ktorá by svojim používateľom dala priestor pre tieto služby. Bohužiaľ, behom môjho vývoja jedna taká aplikácia vznikla.

Ako by mala aplikácia fungovať v praxi? Predstavme si, že jeden z používateľov je napríklad študent v Brne pochadzajúci z Popradu zo Slovenskej republiky. Tento študent si doma zabudne svoje poznámky ku skúške, ktoré bude v najbližšej dobe potrebovať, ale už nebude mať čas si ísť pre ne domov. Vystaví ponuku v aplikácii, vďaka ktorej sa bude môcť dohodnúť s iným používateľom na prevoze poznámok do Brna.

Ďalším prípadom, kedy môže byť táto aplikácia užitočná, je pri ochorení alebo úraze človeka. Chorý používateľ, ktorý nemá v dosahu svojich blízkych, si potrebuje nakúpiť alebo zájsť do lekárne. V jeho zdravotnom stave je táto činnosť však náročná. Aplikácia mu umožní uverejniť ponuku s tým, čo by potreboval nakúpiť.

V neposlednom rade z mnohých ďalších prípadov môže byť dopyt. Predstavme si, že používateľ cestuje z Brna do Prahy napríklad vlakom a má miesto v batožine, chce si zlacniť cestovanie, tak uverejní dopyt a používateľ, ktorý potrebuje niečo previesť, tak sa mu ozve.

Takýmto spôsobom môžu používatelia tejto aplikácie jednoducho vyriešiť svôj problém alebo naopak si privyrobiť peniaze naviac, pri ich cestách alebo prechádzkach.

Aplikácia TouchDeal je vytvorená v duchu zdarma dostupných moderných technológií a trendov vo svete vývoja mobilných aplikácii pre platformu Android. Aplikácia je vyvíjaná v jazyku Kotlin, ktorý sa stal na Google I/O 2017 oficiálne podporovaný pre vývoj aplikácii popri jazyku Java a C++. Ďalej je použítá RxJava a pre databázu sa používa Firebase. Podrobnejší popis pre Google Firebase kapitola 3.6

Kapitola 2

Špecifikácia požiadavkov

Kapitola detailne rozoberie a špecifikuje požiadavky pre mobilnú aplikáciu. Na nasledujúcich riadkoch bude rozobraté čo je UML jazyk, základné prvky a aké diagramy poznáme. Tieto poznatky budú využité pre špecifikáciu požiadavkov pre mobilnú aplikáciu na objednávanie drobných služieb. V špecifikácii sa budú používať výrazy aktivita 4.2 a fragment 4.2

2.1 UML jazyk

Na nasledujúcich riadkoch bude vysvetlený pojem *UML (Unified Modeling Language)*, jeho základná štruktúra, stavebné bloky a kde sa využíva. V krátkosti budú predstavené diagramy, ktoré sa budú využívať pre špecifikáciu jednotlivých akcií v mobilnej aplikácii. Informácie sú z knihy [7] a OMG, ktorá špecifikuje štandard UML. [8]

UML

Unifikovaný [1] modelovací jazyk *(Unified Modeling Language)* je univerzálny jazyk pre vizuálne modelovanie systémov. Najčastejším využitím je pre modelovanie objektovo orientovaných softvérových systémov, pretože má objektovo orientovaný prístup. Avšak má širšie použitie, výplyva to z jeho rozšírovacích mechanizmov. Najnovšia špecifikácia je 2.5.1 aktualizovaná v decembri 2017. Nižšie sú uvedenné dva aspekty modelu UML:

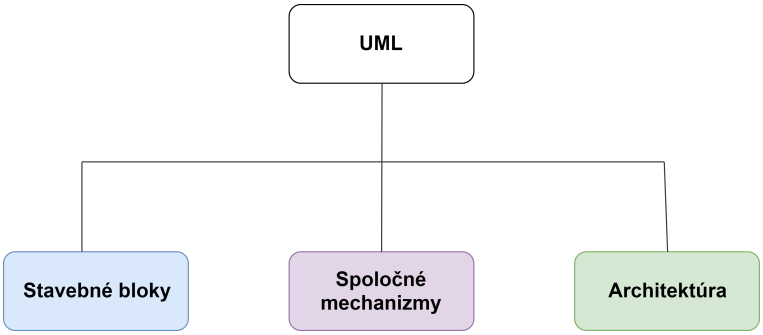
- **Statická štruktúra** popisuje aké typy objektov sú pre modelovanie daného systému dôležité. Taktiež popisuje ako tieto objekty spolu súvisia.
- **Dynamické chovanie** popisuje životný cyklus zmieňovaných objektov a spôsob ich vzájomnej spolupráce s cieľom dosiahnuť funkcie daného systému.

2.1.1 Štruktúra jazyka UML

Štruktúra jazyka UML je zobrazená na obrázku 2.1 . Z obrázku je vidieť, že jazyk UML sa skladá z týchto častí:

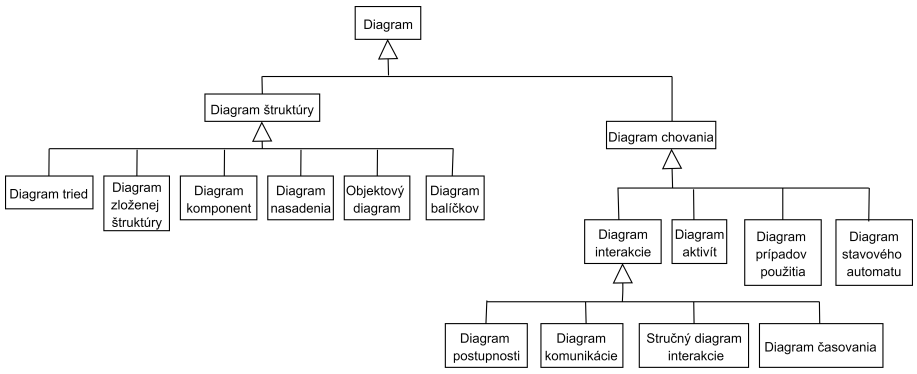
- **Stavebné bloky** sú to základné prvky modelu relácie a diagramy

[1] ARLOW, Jim a Ila NEUSTADT, UML 2 a unifikovaný proces vývoje aplikací (Brno: Computer Press, 2007), 33.



Obr. 2.1: UML

– **Z predmetov (things)** čo sú samotné prvky, ktoré sú:
  * **Štrukturálná abstrakcia (structural things)** - podstatné mená modelu UML ako sú triedy, rozhrania, prípad užitia a ďalšie
  * **Chovanie (structural things)** - slovesá modelu UML, napríklad: interakcia, stav
  * **Zoskupenie (grouping things** - balíčky používané k zoskupovaniu sémanticky súvisejúcich prvkov
  * **Poznámky (annotational things** - anotácia, ktorú je možné pripojiť k modelu
– **Diagramy** sú pohľady na model. Existuje celkom 13 typov diagramov UML, ktoré sú znázornené na obrázku 2.2. Na nasledujúcich riadkoch budú v krátkosti predstavené diagramy, ktoré sa využívajú pre špecifikáciu požiadavkov, návrhu aplikácie a samotnej implementácie.



Obr. 2.2: Štruktúra diagramov

Diagram prípadov užitia

Diagram prípadov užitia *(use case diagram)* zobrazuje chovanie systému alebo jeho časť z hľadiska používateľa.

* účastník *(actor)* - reprezentuje ho ktokoľvek alebo čokoľvek mimo sytém, kto alebo čo komunikuje so systémom.
* hranice *(boundary)* - vymedzujú hranice systému alebo podsystému

* vzťahy *(relationship)* - využíva vzťahy závislosť alebo asosiáciu, ktoré sú vysvetlené v tabuľke 2.1

Sekvenčný diagram

Sekvenčný diagram *(sequence diagram)* znázorňuje interakcie medzi čiarami života ako časovú usporiadanú postupnosť udalostí. Diagramy sú najbohatšie a najpružnejšie formy diagramov interakcie.

Diagram aktivít

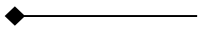
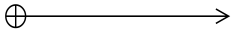
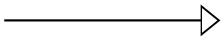
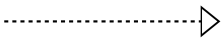
Diagram aktivít (activity diagram) je objektovo orientovaný vývojový diagram. Vďaka tomu je možné procesy modelovať ako aktivitu, ktorá sa skladá z kolekcie uzlov spojených hranami.

Diagram tried

Diagram tried *(class diagram)* znázorňuje dátové štruktúry a oprácie pri objektoch a súvislosti medzi objektami. Napríklad, ktoré atribúty budú viditeľné pre všetky triedy alebo aké metódy existujú v triedach.

– **Zo vzťahov (relationships)** relácie určujú, ako budú spolu dva predmety významovo súvisieť. Prehľad vzťahov je znázornené v tabuľke 2.1.

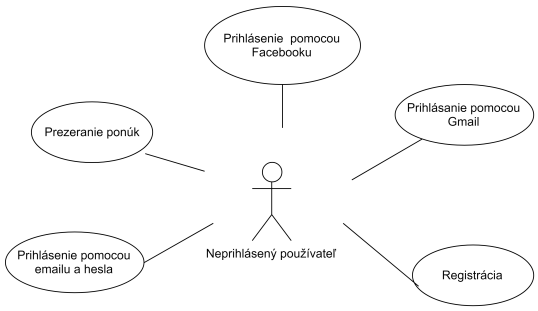
Tabuľka 2.1: Vzťahy v reláciach

Typ relácie	Syntax UML zdroj cieľ	Stručný popis
Závislosť (Depemdncy)		Zmena v určitom predmete ovplyvňuje význam závislého predmetu
Asociácia (Association)		Popis množiny spojenia medzi objektami
Agregácia (Agregation)		Cieľový prvok je súčasťou zdrojového prvku
Kompozícia (Composition)		Silnejšia forma agregácie (má viac obmedzení)
Ochranná nádoba (Containment)		Zdrojový prvok obsahuje cieľový prvok
Zovšeobecnenie (Generalization)		Jeden prvok je špecializáciou iného prvku a je ho možné nahradiť všeobecnejším prvkom
Realizácia (Realization)		Asociácia medzi klasifikátormi, kde jeden klasifikátor určuje dohodu a jeho uskutočnenie zaručuje druhý klasifikátor

2.2 Špecifikácia pomocou diagramov prípadov použita

Pre základný návrh mobilnej aplikácie sa použijú *diagramy prípadov užitia (use case diagrams)*. V prvej podkapitole bude znázornený diagram prípad použitia pre neprihláseného používateľa a v druhej sa znázorní prípad použitia pre prihláseného používateľa.

2.2.1 Neprihlásený používateľ



Obr. 2.3: Diagram prípadu použitia neprihláseného používateľa

Na diagrame prípadu použitia 2.3, je znázornené, čo môže neprihlasený používateľ robiť. Z diagramu vyplýva, že si môže pozrieť ponuky, ktoré pridávajú prihlásení používatelia, aby mal motívaciu sa registrovať alebo sa prihlásiť do mobilnej aplikácie.

Ako neprihlásený používateľ má umožnené sa prihlásiť pomocou sociálnej siete Facebook ako aj pomocou emailu od spoločnosti Google. Pre používateľov, ktorí nedisponujú týmito účtami alebo sa nechcú prihlasovať pomocou týchto účtov, majú možnosť sa prihlásiť pomocou emailu a hesla. Pre poslednú možnosť prihlásenia, je umožnená registrácia.

2.2.2 Prihlásený používateľ

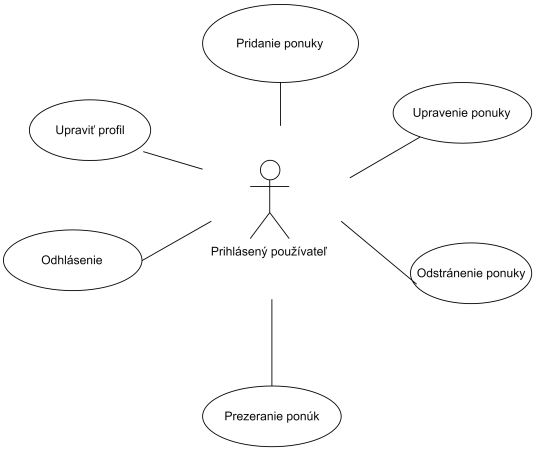
Prihlásený používateľ má definované prípady použitia na diagrame 2.4. Na prvý pohľad je vidieť, že má omnoho viacej prípadov použitia ako neprihlásený používateľ. Z diagramu je zrejmé, že prihlásený používateľ môže vytvoriť ponuku, potom následne ju môže upraviť a ak zistí, že o ponuku, ktorú zadal nemá záujem alebo je neaktuálna môže ju odstrániť. Ako prihlásený používateľ môže upravovať svoj profil.

2.3 Špecifikácia prihlásenie a odhlásenie používateľa

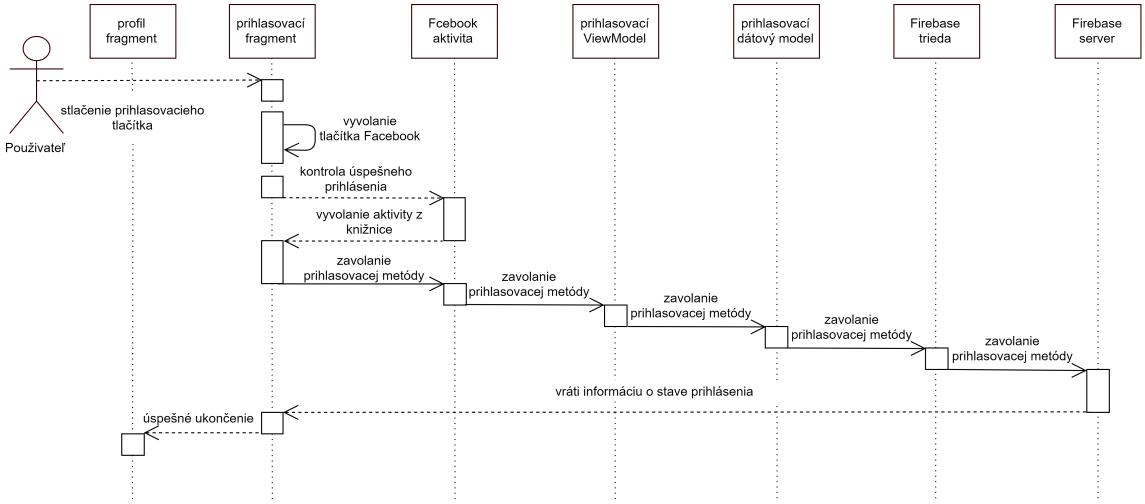
Podkapitola sa zameriava na špecikáciu prihlásenia používateľa pomocou Facebooku, Gmailu a nakoniec pomocou emailu a hesla, ktoré je riešené cloudovým riešením od Google Firebase, ktoré je vysvetľované v kapitole **??**. Špecifikácia je podporená sekvenčnými diagramami, ktoré špecifikujú tok prihlásenia do aplikácie.

2.3.1 Prihlásenie pomocou Facebooku

Sekvenčný diagram zobrazený na obrázku 2.5, špecifikuje chovanie prihlásenia aplikácie pomocou Facebooku. Ako je vidieť z diagramu, používateľ, ktorý sa chce prihlásiť musí v prvom rade kliknúť na prihlásiť sa pomocou Facebooku. To vyvolá správu na stlačenie tlačidla od spoločnosti Facebook.



Obr. 2.4: Diagram prípadu použitia prihláseného používateľa



Obr. 2.5: Sekvenčný diagram pre prihlásenie pomocou Facebooku

Pokiaľ je používateľ prihlásený na svojom mobilnom zariadení, tak sa spustí komunikácia so serverom. V opačnom prípade sa otvorí aplikácia Facebook, cez ktorú sa používateľ prihlási do mobilnej aplikácie od Facebooku a následne sa spustí komunikácia so serverom pre prihlásenie do mobilnej aplikácie na objednávanie drobných služieb.

V prípade, že bolo prihlásenie úspešné, tak podľa diagramu 2.5, sa pošle správa na spracovanie požiadavky do ViewModelu, následne sa prepošle správa do prihlasovacieho dátového modelu, dátový model pošle správu do Firebase triedy, odkiaľ je poslaný asynchrónny požiadavok na prihlásenie do cloudového riešenia prihlásenia aplikácie cez poverenia *(credential)*

Asynchrónny požiadavok sa naspäť vráti do prihlasovacieho fragmentu, kde je výsledok prihlásenia do Firebase. Ak bolo úspešné, tak sa zmení prihlasovací fragment na profil fragment.

2.3.2 Prihlásenie pomocou Gmailu

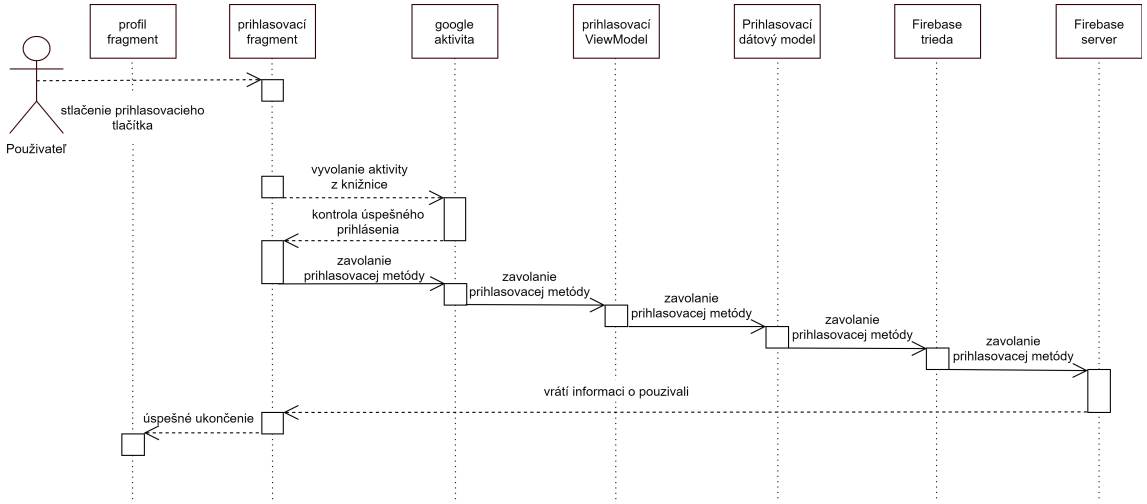
Prihlásenie pomocou služby Gmail je vidieť z diagramu 2.6. Používateľ, ktorý sa chce prihlásiť pomocou tejto služby, musí ako prvé stlačiť prihlásiť sa pomocou Gmailu. To pošle správu na otvorenie aktivity od Google aby si mohol používateľ vybrať účet, pod ktorým sa chce prihlásiť.

Po vybratí účtu sa skúsi spojiť so serverom a overiť používateľa so zadaným gmailovým účtom. Ak nastane chyba pri komunikácii so serverom, prihlasovanie končí. Po odoslaní správy do fragmentu sa pošle správa pre kontrolu, či prišli dáta o používateľovi.

Prihlasovanie končí a používateľ zostáva na prihlasovom fragmente pri neprijatí dát o používateľovi.

V opačnom prípade prihlasovací fragment odosiela správu do ViewModelu. Z ViewModelu sa správa posiela do prihlasovacieho dátového modelu, odkiaľ správa posiela správu do triedy pre asychrónne zavolanie prihlásenia pomocou poverania do cloudového riešenia podobne ako v prípade prihlásenia sa pomocou Facebooku.

Asynchrónny požiadavok pošle správu do prihlasovacieho fragmentu a vyhodnotí zostávanie na prihlasovacom fragmente alebo presunie používateľa do profil fragmentu.



Obr. 2.6: Sekvenčný diagram pre prihlásenie pomocou Gmailu

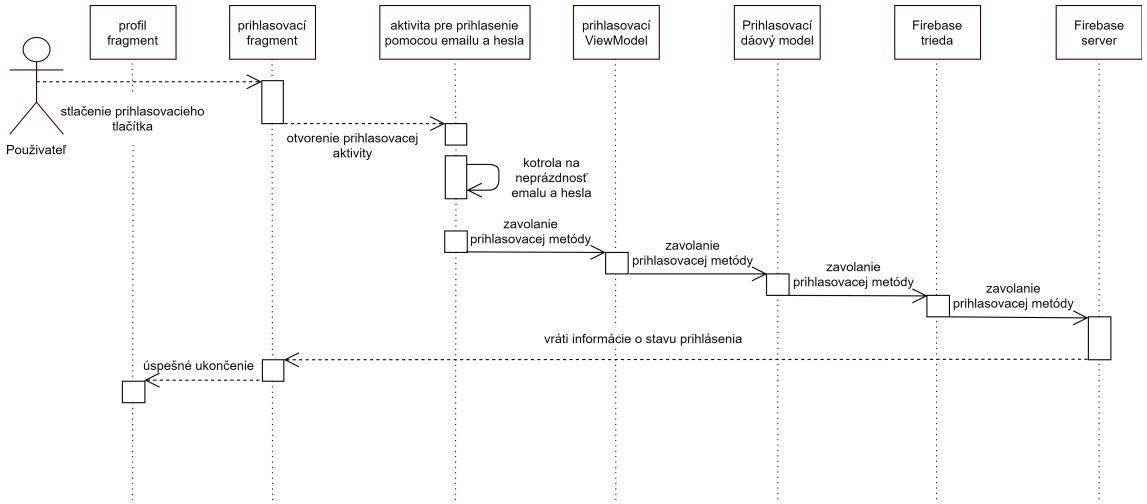
2.3.3 Prihlásenie pomocou emailu a hesla

Používateľ, ktorý nechce využiť ani jednu z možnosti prihlásenia sa vyššie, môže využiť prihlásenie pomocou emailu a hesla. Toto prihlásenie znázorňuje diagram 2.7, kde je na prvý pohľad vidieť, že má odlišný tok správ ako prihlásenie sa pomocou Facebooku alebo Gmailu.

Z diagramu 2.7 je vidieť, že používateľ, ktorý sa chce prihlásiť, musí stlačiť tlačidlo prihlásiť sa pomocou emailu. V tomto okamžiku prihlasovací fragment pošle správu na otvorenie novej aktivity, aby používateľ mohol zadat prihlasovacie meno a heslo. Používateľ sa vráti na prihlasovací fragment pomocou tlačítka späť ako je znázornené na diagrame 2.7.

Pre prihlásenie, používateľ musí stlačiť tlačítko prihlásiť sa. Aktivita pošle správu na kontrolu na neprázdnosť polí pre zadávanie emailu a hesla. V prípade, že polia sú prázdne končí prihlasovanie a používateľ zostáva v tejto aktivite. V opačnom prípade aktivita na prihlásenie pomocou emailu odošle správu do ViewModelu. ViewModel odošle správu do

prihlasovacieho dátového modelu, odkiaľ pošle správu do Firebase triedy, kde sa posiela asynchrónny požiadavok na prihlásenie pomocou emailu. Po prijatí správy zo servera sa odošle asynchrónne správa do aktivity pre prihlásenie pomocou emailu, kde ak prihlásenie skončilo úspešne, sa pošle správa pre ukončenie aktivity a zobrazenie profil fragmentu, inak sa zostáva v aktivite.



Obr. 2.7: Sekvenčný diagram pre prihlásenie pomocou Emailu a hesla

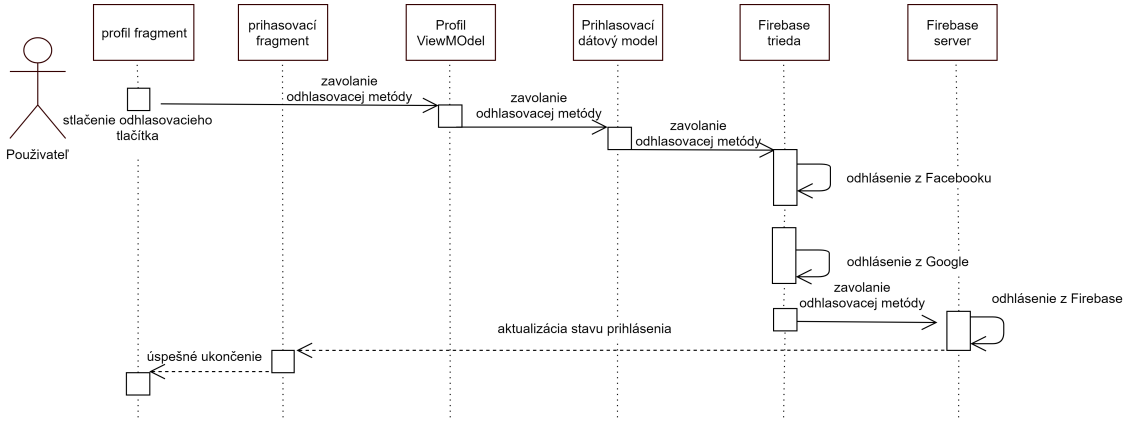
2.3.4 Odhlásenie používateľa

Sekvenčný diagram znázornený na obrázku 2.8, špecifikuje odhlásenie používateľa. Z diagramu 2.8, je vidieť, že pre odhlásenie musí používateľ kliknúť na odhlásenie v profil fragmente.

Po stlačení tlačítka fragment posiela správu, do ViewModelu. ViewModel odošle správu do dátového modelu, odtiaľ do Firebase triedy. Firebase Trieda odošle asynchrónny požiadavok na odhlásenie z Facebooku, ak bol používateľ prihlásený pomocou Facebooku alebo pošle asynchrónny požiadavok na odhlásenie z Gmailu, ak bol používateľ touto metódou prihlásený. Po tomto odošle asychrónne požiadavok na odhlásenie z Firebase a posiela asychrónne správu otvrenie fragmentu s prihlásením.

2.4 Špecifikácia registrácia

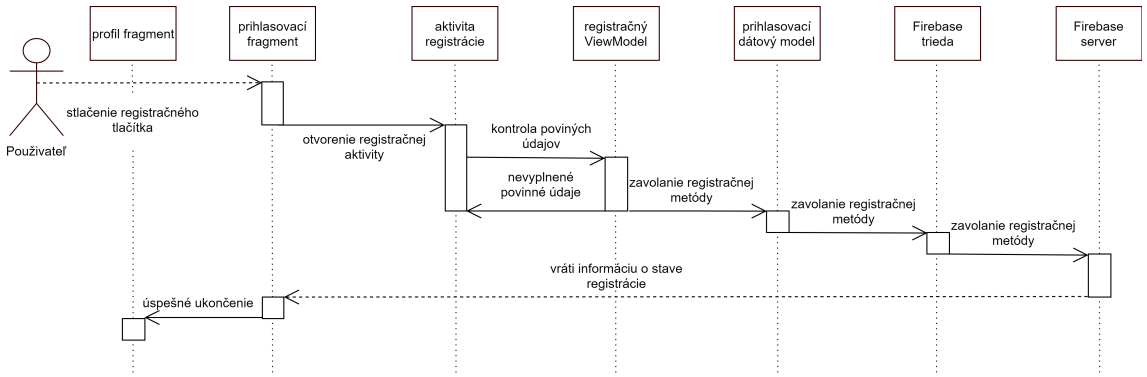
Registrácia používateľa sa jedná iba, ak používateľ sa chce prihlasovať pomocou emailu a hesla. Diagram tejto špecifikácii je znázornený na obrázku 2.9. Používateľ, ktorý sa chce zaregistrovať, klikne na tlačítko registrovať sa
v prihlasovacom Fragmente. Ten pošle správu na otvorenie aktivity s registráciou. Pre registráciu používateľ klikne na registrovať sa. V tomto okamžiku aktivita na registráciu odosiela správu pre kontrolu na neprázdnosť povinných údajov. Ak táto kontrola prejde úspešne, odosiela správu do ViewModelu. Následne do prihlasovacieho dátového modelu. Synchrónne sa pošle požiadavok do Firebase triedy, odkiaľ sa asynchrónnne posiela požiadavok na registráciu. Odpoveď zo servera príde do registračnej aktivity, ktorá buď zostane, pokiaľ registrácia je neúspšná. V opačnom prípade sa aktivita zavrie a zobrazí sa proifil fragment.



Obr. 2.8: Sekvenčný diagram odhlásenia

Pri obrdžaní správy o registráciu z aktivity do ViewModel posiela ViewModel správu do dátového modelu pre prihlásenie, kde tento model odosiela správu do Firebase Triedy, ktorá posiela asynchrónny požiadavok.

Firebase trieda dostane odpoveď zo servera, ktorá je asynchrónne poslaná do aktivity na registráciu. Ak prebehla registrácia úspešne, tak sa zavrie registračná aktivita a zobrazí sa profil fragment. Pokiaľ pokus o registráciu bol neúspešný, používateľ zostáva na aktivite.

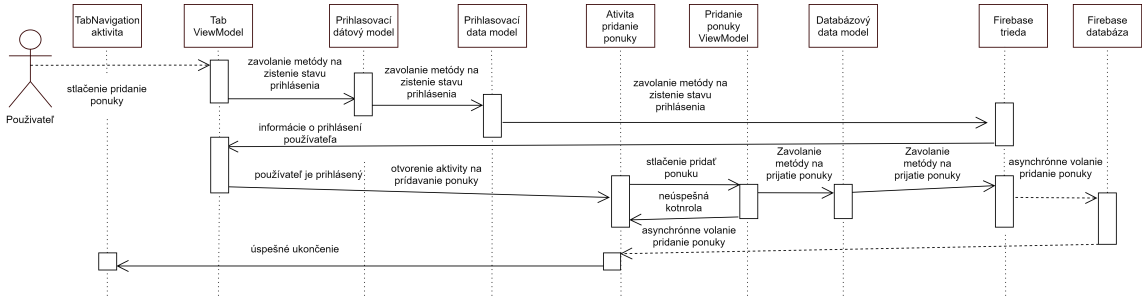


Obr. 2.9: Sekvenčný diagram pre registráciu

2.5 Špecifikácia pridávanie ponuky

Nasledujúca kapitola špecifikuje pridávanie ponuky, ktorá je znázornená sekvenčným diagramom 2.10. Z tohto diagramu vyplýva, že ak chce používateľ pridať ponuku, tak musí stlačiť tlačidlo *pridať ponuku*, ktoré pošle správu do aktivity s navigáciou, ktorá následne pošle správu do ViewModelu na otvorenie aktivity s pridaním ponuky.

ViewModel odošle správu do dátového modelu, ktorý je znázornený na diagrame 2.10, ktorý následne pošle správu do Firebase triedy. Firebase trieda synchrónne pošle naspäť informáciu o tom, či je alebo nie je používateľ prihlásený. Odpoveď sa dostane naspäť do aktivity s navigáciou. V prípade, že používateľ je neprihlásený, proces na pridávanie ponuky sa končí. Prihlásenému používateľovi aktivita s navigáciou pošle správu na otvorenie aktivity s pridaním ponuky.



Obr. 2.10: Sekvenčný diagram pre pridávanie ponuky

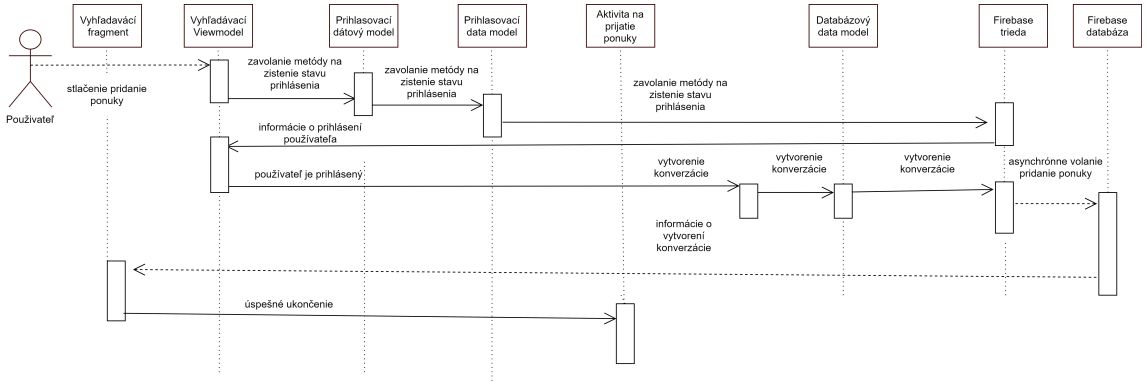
Pre pridanie ponuky, stlačí používateľ tlačidlo pridať. Po stlačení tohto tlačidla aktivita na pridávanie ponuky odošle správu do viewModelu na pridávanie ponuky pre kontrolu, či používateľ vyplnil všetky potrebné informácie. Tieto kontroly sa posielajú pre každú položku zvlášť. Keď prejde posledná výmena správ pre kontrolu povinných položiek, ViewModel pre pridávanie ponuky odošle správu do dátového modelu pre firebase databázu. Dátový model následne prepošle správu do Firebase triedy, ktorá pošle asynchrónny požiadavok na pridanie ponuky. Fireabse trieda pošle asynchrónne správu do aktivity pre pridávanie ponúk, aby sa ukončila. Týmto končí proces pridávania ponuky.

2.6 Špecifikácia prijímanie ponuky

V podkapitole sa naŠpecifikuje pomocou sekvenčného diagramu 2.11, ako používateľ bude môcť prijať ponuku. V prvom rade musí kliknúť na tlačidlo spáva, ktoré je v každej zobrazenej ponuke. Kliknutie vyvolá poslanie správy do vyhľadávacieho fragmentu. Vyhľadávací fragment podobne ako pri pridávaní ponuky posiela správu o zistení stavu prihlásenia.

Ak používateľ je neprihlásený, tak proces prijatia ponuky sa končí.

V prípade, že používateľ je prihlásený, pošle správu ViewModelu. Ten následne pošle správu do dátového modelu pre Firebase databázu. Dátový model, pošle správu do Firebase triedy, odkiaľ sa pošle asynchrónny požiadavok na získanie informácii o konverzácii.



Obr. 2.11: Sekvenčný diagram pre prijímanie ponuky

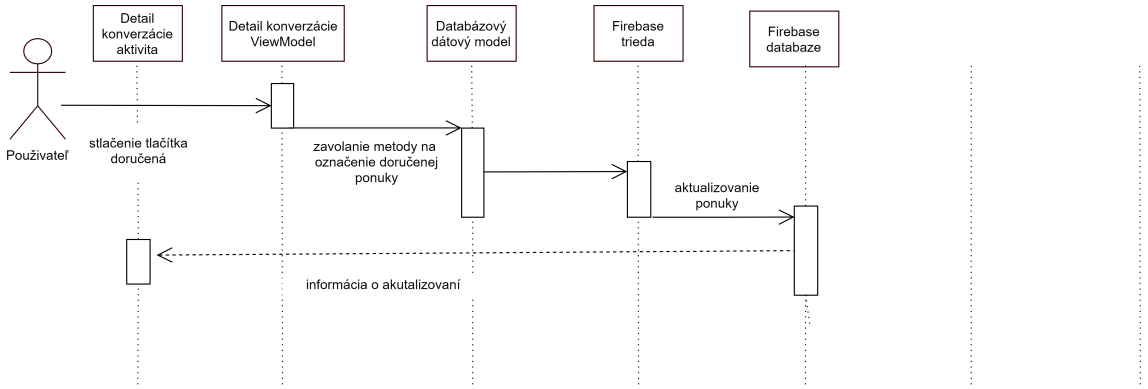
Ak tento požiadavok skončí neúspechom používateľ musí znova stlačiť na tlačítko správa. Ak príde odpoveď so servera, že konverzácia sa úšpesne vytvorila, pošle sa asynchrónny

požiadavok do vyhľadávacieho fragmentu, ktorý následne pošle požiadavok na otvorenie aktivity s konverzáciou.

V tomto momente požiadavok na prijatie ponuky končí, pretože sa používatelia dohadujú pomocou správ na podrobnostiach.

2.7 Špecifikácia označenia doručenej ponuky

Používateľ, ktorý bude chcieť označiť ponuku ako doručenú, tak v aktivite detailu konverzácie, klikne na tlačidlo doručená, ako je to znázornené na sekvenčnom diagrame 2.12. Aktivita detail konverzácie pošle správu do ViewModelu. Ten následne prepošle správu do dátového modelu pre databázu Firebase. Správa sa následne pošle do Firebase triedy, kde sa pošle asynchrónny požiadavok na aktualizovanie ponuky, pre stav doručená. Asynchrónne sa pošle požiadavok naspäť do aktivity detail konverzácie, kde sa pošle výsledok o úspechu alebo neúspechu označenia.



Obr. 2.12: Sekvenčný diagram pre prijímanie ponuky

Kapitola 3

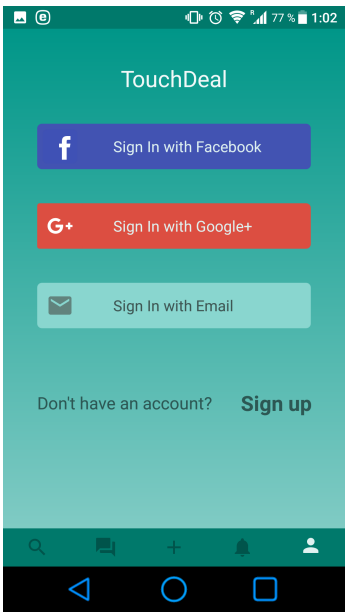
Návrh mobilnej aplikácie

Nasledujúca kapitola obsahuje podrobne návrh mobilnej aplikácie na objednávanie drobných služieb. Prvá podkapitola sa zaoberá UI/UX. V Druhej podkapitole bude vysvetlený pojem architektonický vzor a jeho 3 najbežnejšie varianty, ktoré sa používajú pri tvorbe aplikácii. Na konci tejto podkapitoly sa vyberie architektonický vzor pre návrh a následnú implementáciu. Tretia podkapitola sa zameriava na serverové riešenie Google Firebase, ktorá sa využije na implementáciu serverovej časti.

3.1 Návrh UI/UX

V podkapitole návrh UI/UX budú navrhnuté jednotlivé obrazovky, ktoré vyplývajú z kapitoly špecifkácia požiadavkov. 2. V nasledujúcich podkapitolách bude návrh obrazoviek prihlásenia prijatia pounky a pridanie ponuky.

3.2 Prihlasovacia obrazovka

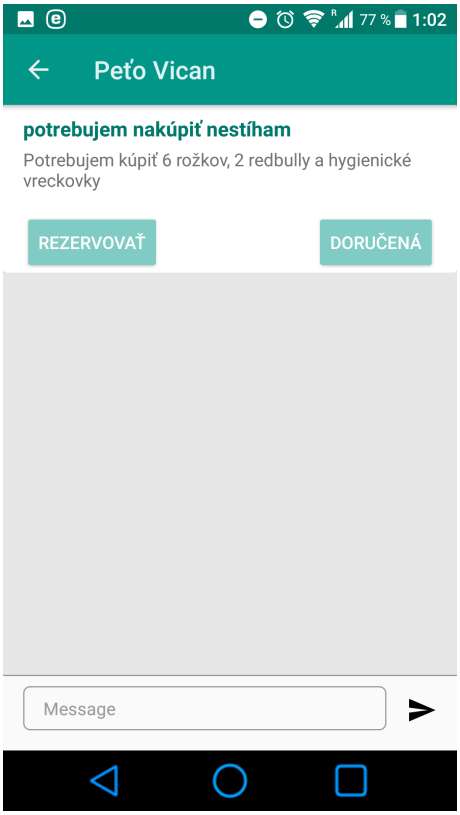


Obr. 3.1: Prihlasovacia obrazovka

V podkapitole bude ukazaný návrh, prihlasovacej obrazovky, ktorý je rovnako aj naimplementovaný. Na obrázku 3.1 vidíme, že má tlačidlá na prihlásenie sa pomocou Facebooku, Gmailu, Emailu a hesla. Tok udalsotí je popísany v špecifikácii požiadavkov pre prihlásenie 2.3

3.3 Prijatie ponuky obrazovka

V podkapitole bude zobrazený návrh na prijatie ponuky, ktorý je znázornený na obrázku 3.2.



Obr. 3.2: Obrazovka prijatie ponuky

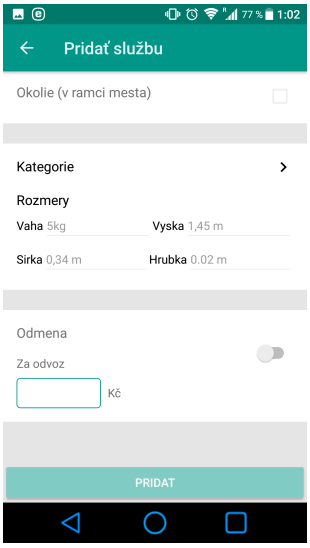
3.4 Pridávanie ponuky

Keďže aplikácia má hlavnú fnkcionalitu pre pridávanie a prijatie ponúk tak posledným návrhom, ktorý si ukázeme bude návrh pridania ponuky, ktorý je znázorený na obrázkoch 3.3 a 3.4.

3.5 Architektonický vzor

Pojem *architektonický vzor (anglicky architectural pattern)* je často zamieňaný za *návrhový vzor*. *Návrhové vzory* (anglicky *design patterns*) popisujú doporučené riešenie často sa vyskytujúcich úloh. Tieto vzory sú určené pre objektovo-orientované programovacie jazyky.

Obr. 3.3: Obrazovka na pridávane ponuky



Obr. 3.4: Obrazovka pre pridávanie ponuky 2

Naproti tomu *architektonický vzor* popisuje podstatné štrukturálne komponenty pri vývoji softwéru a väzby medzi nimi. Zaoberá sa, ktorá komponenta je zodpovedná za získanie dát z databázy alebo servera, ich spracovanie podľa určenej logiky a následné zobrazenie používateľovi. Zároveň rieši následné znovupoužitie komponentu, ich bezpečnosť a rozšíriteľnosť. Najbežnejšími vzormi sú: MVC(anglicky *Model-View-Controller*), MVP(anglicky *Model-View-Presenter*)) a MVVM (anglicky *Model-View-ViewModel*). V podkapitole sú čerpané informácie z wikipeédie [11], [10] a z elektronického článku [4]

3.5.1 Model-View-Controller

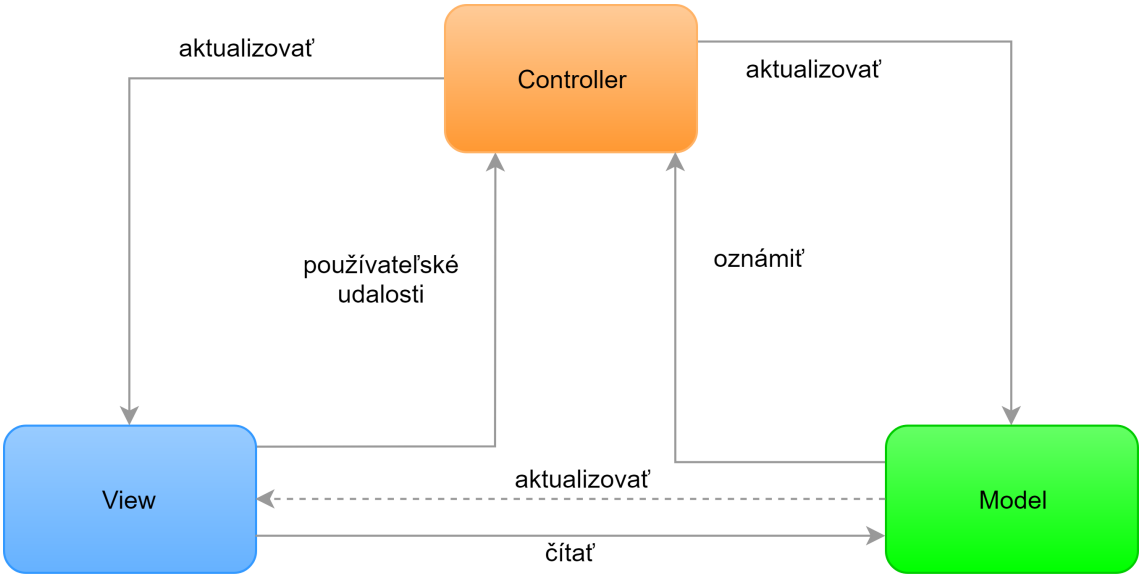
Model-View-Controller (niekedy taktiež nesprávne označovaný ako *Model-2*) je softwarová architektúra, ktorá rozdeľuje dátový model aplikácie, uživateľské rozhranie a riadiacu logiku do troch nezávislých komponentov, tak že modifikácia niektorej z nich má iba minimálny dopad na ostatné. Na obrázku 3.5, je vidieť ako komponenty medzi sebou interagujú. Jednotlivé komponenty sú:

Model

Model je doménová špecifická reprezentácia informácií, s ktorými aplikácia pracuje. Model môže mať logiku na výpočty, databázové dotazy, validácia a podobne. Model nevie, odkiaľ dáta v parametroch prišli a ani ako budú výstupné dáta sformátované a vypísané.

View (pohľad)

Pohľad prevádza dáta reprezentované modelom, do podoby vhodné k interaktívnej prezentácii používateľovi. Cez Model pristupuje k dátam a k stavu aplikácie a špecifikuje ako dáta majú byť reprezentované. Po zmene dát v modely sa zaktualizuje pohľad s dátami.



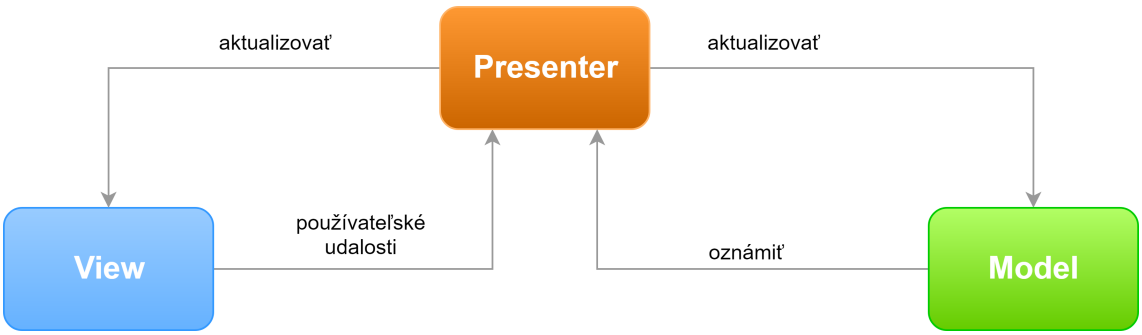
Obr. 3.5: Model-View-Controller

Controller(radič)

Radič reaguje na udalosti typicky pochádzajúce od používateľa. Je to chýbajúci prvok, ktorý spája celý vzor dokopy a rieši formátovanie dát na zobrazenie alebo zmenu v samotnom modely. Najčastejšie jedna entita má jeden radič.

3.5.2 Model-View-Presenter

Model-View-Presenter je softwérová architektúra, ktorá vznikla deriváciou od architektúry MVC, ktorú sme ako prvú rozoberali. Na obrázku 3.6 vidíme rozdiel oproti architektúre MVC, ktorá je znázornená na 3.6. Táto architektúra je rozdelená na tri nezávislé komponenty, Model, View(pohľad)
a radič nahradil presenter. Pri zmene pohľadu sa musí zmeniť aj presenter.



Obr. 3.6: Model-View-Presenter

Model

Model obsahuje opis takzvanej "business logic", čo znamená, že celá logika je oddelená od zobrazovacej vrstvy.

View (pohľad)

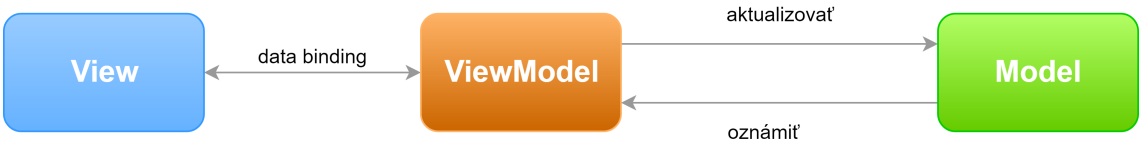
Pohľad reprezentuje komponenty, pomocou ktorých sa len zobrazujú dáta používateľovi a vyvoláva akcie na zmenu dát, ktoré sa vykonávajú v presentri.

Presenter

Presenter je zodpovedný za manipuláciu zobrazovacích komponent. V presentri je logika na pretransformovanie dát z modelu pre zobrazenie. Presenter si udržuje referenciu na view.

3.5.3 Model-View-ViewModel

Model-View-ViewModel je softwérová architektúra, ktorá je silná a robustná kvôli typickému využitiu väzby údajov(anglicky *data biding*). Táto väzba údajov, nám umožnuje viazať pohľady s view-modelmi, to znamená, že jeden view-model, môže mať na seba napojených N pohľadov. Schéma je znázornená na obrázku 3.7



Obr. 3.7: Model-View-ViewModel

Model

Model obsahuje reprezentáciu informácií, ktoré typicky sú operácie v databáze alebo spracovanie dát z iného úložiska napríklad servera.

View (pohľad)

Pohľad reprezentuje komponenty, pomocou ktorých sa len zobrazujú dáta používateľovi, avšak vyvoláva aj akcie na zmenu dát, ktoré sú previazané pomocou väzby údajov s view-modelom.

ViewModel

ViewModel je spájajúca komponenta medzi pohľadom a modelom. Vystavuje rôzne príkazy alebo vlastnosti, ktoré pomáhajú pri udržiavaní aktuálnosti pohľadu, pripravuje a transformuje dáta na zobrazenie alebo na manipuláciu s modelom.

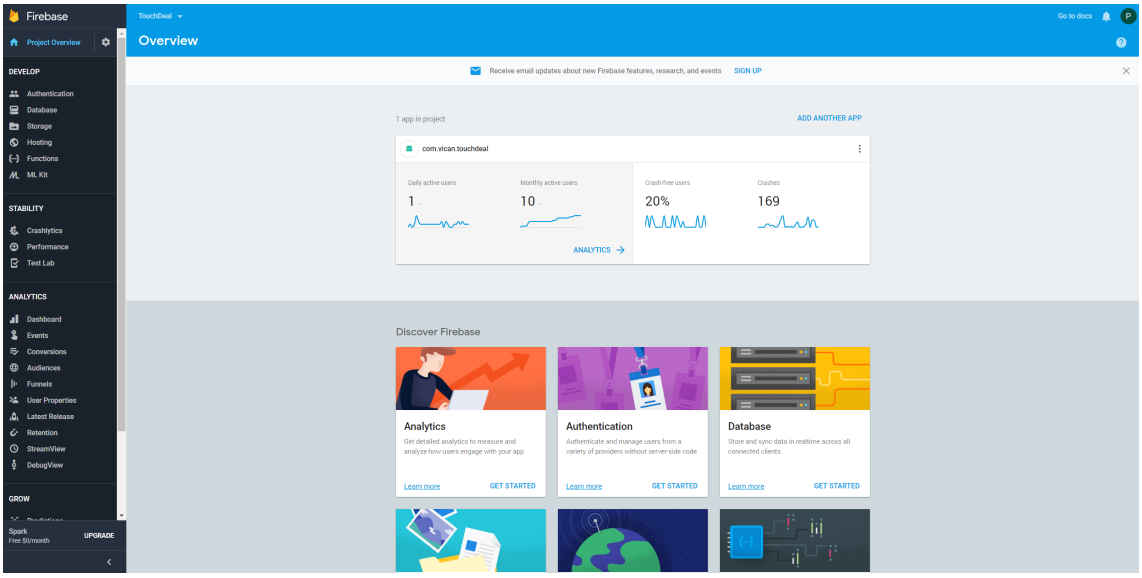
3.5.4 Zhrnutie a záver

V podkapitole sme si v skratke vysvetlili MVC, MVP a MVVM vzor. Tieto informácie pomohli pri následnom výbere architektonického vzoru. Pre návrh a následnú implementáciu sa vybral MVVM vzor. Vybratý vzor sa vybral, pretože na Google I/O v roku 2015 sa predstavila knižnica pre prácu väzby pohľadu na objekt, táto knižnica sa volá data binding. Obrovskou výhodou je jednoduché testovanie logiky pre transformáciu dát pre používateľa, pretože sa tam nedrží referencia na pohľad a pri každej zmene sa nemusí meniť aj samotný

view-model. S týmto sa viaže ďalšia dobrá vlastnosť a to, že jeden ViewModel môže obsluhovať viacero pohľadov, a tým sa redukuje kód a zvyšuje znovupoužiteľnosť kódu. S ďalšou z mnohých výhod prečo sa vybral vzor MVVM je, že na Google I/O v roku 2017 vyšla oficiálna podpora pre architektonický vzor MVVM pod balíčkom Ärchitecture components". Tento balíček zahŕňa životný cyklus view-modelu pre Android aplikáciu ako aj nadstavbu nad SQL lite databázou v knižnici s názvom "Room".

3.6 Google Firebase

Google Firebase je komplexné riešenie pre vývoj mobilných a webových aplikácii. *Firebase* bola firma, ktorá sa zaoberala cloudovým riešením databázy v reálnom čase, ktorú v roku 2014 odkúpila spoločnosť Google a vytvorila tak komplexnú službu pre vývoj. Na obrázku 3.8 je zobrazené administračné rozhranie Google Firebase. Do tejto sekcie majú prístup všetci používatelia s emailovým účtom od Google a majú vytvorený svoj vlastný projekt. Použité informácie sú z oficiálnych stránok [5]. V tejto službe sa nachádzajú:



Obr. 3.8: Firebase administračné rozhranie

- Analytika
  - Firebase analytics - Nástroj na zber ananlytických informácii, ktoré môže administrátor využiť, napríklad vek používateľov, na aké miesta v aplikácii klikol a mnoho iného.
- Vývoj
  - Firebase cloud správy
  - Firebase autentifikácia
  - Databáza v reálnom čase
  - Firebase úložisko
  - Firebase hosting

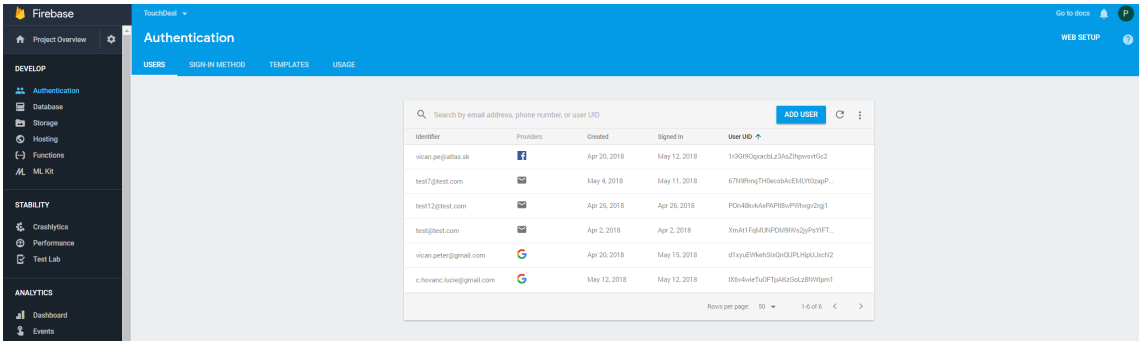
- Stabilita
  - Analytika pádov
  - Výkon
  - Firebase testovacie laboratória pre Android
- Rast
  - Firebase notifikácie
  - Firebase indexácie aplikácie
  - Firebase dynamické linky
  - Firebase pozvánky
  - Firebase vzdialená konfigurácia
  - Adwords
- Zárobok
  - Admob

Pri implemetácii mobilnej aplikácie neboli použité všetky služby, ktoré sú spomenuté ale v podkapitolách nižšie budú spomenuté a v skratke predstavené služby nevyhnutné pre implementáciu.

3.6.1 Firebase autentifikácia

Firebase autentifikácia (anglicky *Firebase Authentification*) je služba pre správu prihlásených používateľov, registrácia nových používateľov, blokovanie používateľov ale zároveň nám pomáha integrovať prihlásenie pomocou Facebooku, Gmailu, Twitter, Instagram, GitHub, prihlásenie emailom a heslom, cez SMS alebo anonymné prihlásenie.

Na obrázku 3.9 je vidieť, že admistrátor, si sám zvolí ktoré prihlásenie chce mať povolené. Detaily ohľadom integrácie Facebooku a Gmailu budú napísané v kapitole dd implementácia autentifikácie.



Obr. 3.9: Firebase rozhranie s autentifikáciou

3.6.2 Databáza v reálnom čase

Databáza v reálnom čase (anglicky *Realtime Database*) pomenovanie dostala kvôli okamžitému doručeniu informácii o modifikácii, pridaniu alebo odstráneniu položky v databáze. Vytvorené záznamy sa prezentujú ako JSON (anglicky *JavaScriptObject Notation*) ale zároveň administrátor v rozhraní môže vytvárať databázu v štruktúre JSON. Konkrétne v tomto prípade ide o ukladanie ponúk, správ medzi používateľmi, profil používateľa, hodnotenie používateľa a rôzne nastavenia, ktoré sa budú prenášať na zariadenie po prihlásení.

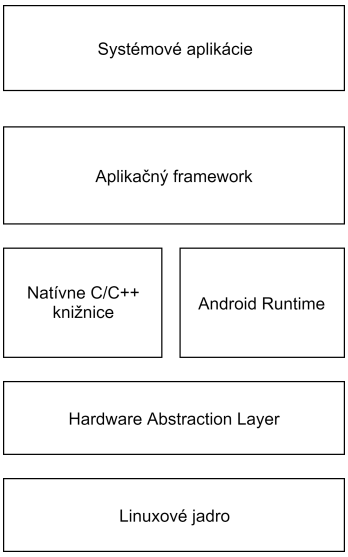
Kapitola 4

Android architektúra

Kapitola sa zaoberá samotnou architektúrou systému Android, jeho základnými funkčnými a grafickými blokmi. Tieto nabudnuté informácie sú ďalej použité pri návrhu a následnej implementácie. Jedna z podkapitol bude venovaná aj grafickému rozhraniu. Informáciami, ktorým, sa kapitola bude venovať sú čerpané z oficiálnej vývojárskej dokumentácie. [2] a z bakalárskej práce [3]

4.1 Popis architektúry

Dôležitou časťou pri vývoji mobilnej aplikácie pre platformu Android je potrebné poznať architektúru tejto platformy. Na obrázku 4.1 znázornené členenie jednotlivých blokov.



Obr. 4.1: Architektúra platformy Android

- **Linuxové jadro (Kernel)** tvorí základ operačného systému (platformy) Android. Zabezpečuje základnú správu procesov, pamäti a sieťovú vrstvu. Taktiež sa stará o bezpečnosť systému, ovládače a rôzne vstupno-výstupné operácie.

- **HAL (Hardware Abstraction Layer)** bola vytvorená ako rozhranie medzi špecifickým typom hardwaru a API frameworkom. Obsahuje viacero knižnicových modulov pre špecifický hardware ako je napríklad"Kamera, Bluetooth, Senzory a iné.

- **Natívne C/C++ knižnice (Native C/C++ Libraries)** obsahuje knižnice pre správu 2D a 3D grafiky pomocou knižnice *OpenGL ES*. Ďalej sú tú napísané systémove komponenty a služby ako napríklad *ART* a *HAL*. Pre vývoj aplikácie ktorá potrebuje natívnu podporu C/C++ sa môže použiť *Android NDK*, ktorý sprístupňuje C/C++ knižnice priamo z nativného kódu aplikácie.

- **Android Runtime** je virtuálny stroj, ktorý existuje od verzie Android 5.0 a vyšší. Každá aplikácia sa spúšťa ako vlastný proces inštancie virtuálneho stroja. ART používa AOT (*ahead-of-time*) kompiláciu, čo znamená, že pri inštalácii sa bytekód prevedie na natívne inštrukcie a tie vykonáva virtuálny stroj. Taktiež využíva JIT (*just-in-time*) kompilácie, čo znamená, že prekladá za behu aplikácie. Android 4.4 a nižší používali *Dalvik Virtual Machine* na vykonávanie bytekódu.

- **Aplikačný framework (Java API Framework)** je sada funkcií operačného systému Android, ktoré je dostupné pre vývojára prostredníctvom rozhrania *API*, ktoré je napísané v jazyku *Java*. Rozhranie obsahuje znovupoužiteľné stavebné bloky, ktoré sú potrebné pre vytvorenie aplikácie pre systém Android medzi ktoré patria prepojenie s jadrom systému, modulárne systémové komponenty a služby.

- **Systémové aplikácie a aplikácie (System Apps and Apps)** je blok v ktorom sa nachádzajú aplikácie inštalované používateľom ale aj systémové aplikácie ako napríklad Kamera, Správy, Email, Kalendár a mnoho iných, ktoré sú implementované pomocou *aplikačného frameworku*. Tieto aplikácie sa správajú ako naištalované aplikácie používateľom, len s tým rozdielom, že sú inštalované s operačným systémom Android.

4.2 Popis komponentov

V podkapitole sú rozobraté jednotlivé komponenty, ktoré sú obsiahnuté v *aplikačnom frameworku*. Komponenty slúžia na zabezpečenie potrebnej funkčnosti a sú základom alikačného jadra

AndroidManifest

AndroidManifest je hlavný konfiguračný súbor aplikácie a musí mať presný názov *AndroidManifest.xml*. V súbore sa definújú bežiace služby, hlavné triedy (Activity), oprávnenia, ktoré aplikácia potrebuje pre svoj beh pri inštalácii aplikácie ale od verzie Android 6.0 (Marshmallow) sa oprávnenia vyžadujú pri behu aplikácie nie pri inštalácii.

Poskytovateľ obsahu (Content Provider)

Poskytovateľ obsahu môže pomôcť aplikácii spravovať prístup k svojím uloženým dátam, dátam uložených inými aplikáciami a poskytnúť spôsob zdieľania údajov s inými aplikáciami. Zapisujú údaje a poskytujú mechanizmy na definovanie bezpečnosti údajov.

Poskytovateľ obsahu musí byť definovaný v konfiguračnom súbore. Pre prístup k dátam sa používa *URI*[1] definovanom tvare *content://authority/path/id*

Odoberateľ obsahu (Broadcast Receiver)

Aplikácie sú schopné vytvoriť udalosti, ktoré môžu pomocou odosieľateľa *(Broadcast)* vysielať alebo prijímača *Receiver* prijímať správy vysielané zo systému Android a iných aplikácií. Tieto vysielania sa posielajú, keď nastane udalosť, ktorá aplikáciu zaujíma. Aplikácie sa zaregistrujú na prijímanie konkrétnych vysielaní. Po odoslaní vysielania systém automaticky smeruje vysielanie do aplikácií, ktoré sa prihlásili na prijímanie daného typu vysielania.

Napríklad, keď je zariadenie na internete pomocou Wi-Fi, tak aktualizuje aplikáciu, pomocou vzdialenej konfigurácie.

Služby (Services)

Služby slúžia na vykonávanie dlhodobej operácie na pozadí a neposkytuje používateľské rozhranie. Nebeží na hlavnom vlákne, takže nezaťažuje plynulý beh aplikácie. Môže spracovávať sieťové transakcie, prehrávať hudbu, vykonávať vstupy / výstupy súborov alebo komunikovať s poskytovateľom obsahu, a to všetko z na pozadí aplikácie. Od verzie Android 8.0 (Oreo) systém ukladá obmedzenia na spustenie služieb na pozadí, keď samotná aplikácia nie je v popredí. Vo väčšine prípadov, by mala aplikácia použiť *naplánovanú úlohu (JobService)*.

AIDL

Iné služby môžu pracovať nezávisle, používateľ spustí službu, tá sa vykoná a sa ukončí. Druhý prípad je, že služby bežia samostatne a čakajú na požiadavky, ktoré zadá používateľ. Ak chceme využiť druhý prípad, na to slúži práve AIDL[2]

AIDL je rozhranie bez žiadnej implementácie, služba musí implementovať rozhranie a daná aplikácia, následne využíva iba toto rozhranie služby

Príklad využitia môže byť použitie služby mimo svojej aplikácie ako napríklad služba na ukončenie prehrávania hudby.

Uloženie dát

Systém Android umožňuje ukladanie perzistentných dát viacerými možnosťami:

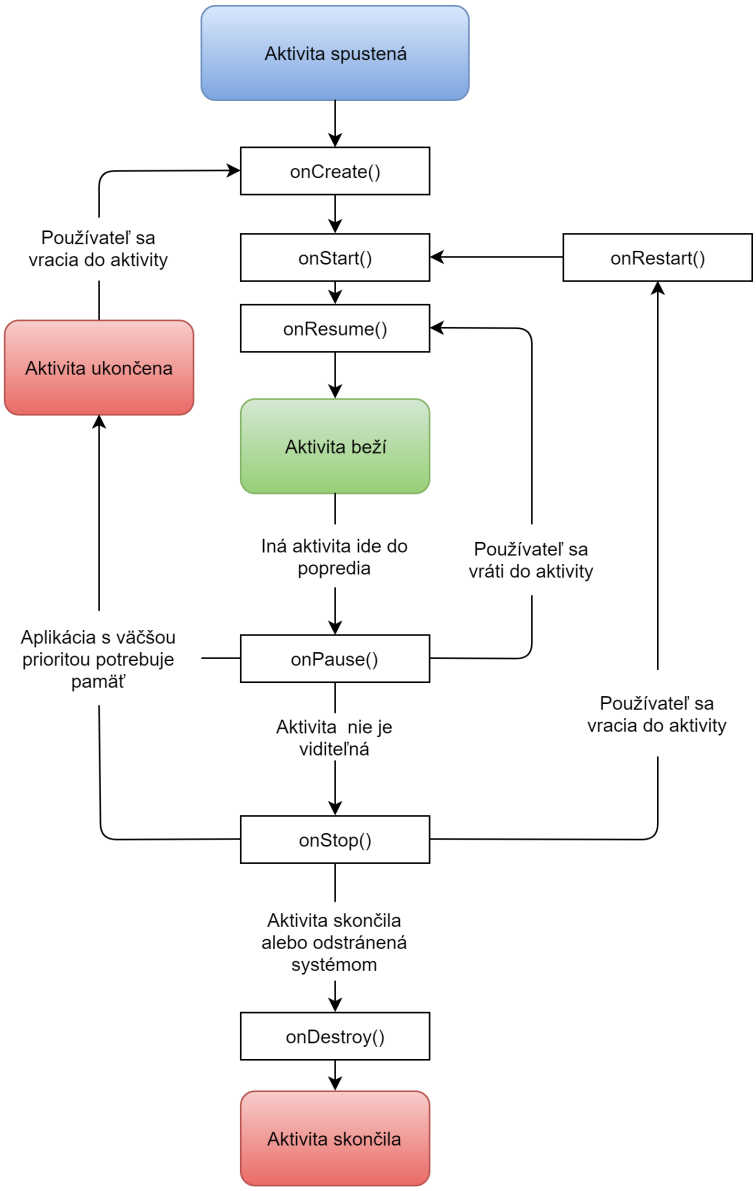
- **Zdieľané nastavenia** sú viditeľné iba pre aplikáciu. Vhodné pre rôzne nastavenia aplikácie, ktoré sa ukladajú vo forme *kľúč-hodnota*
- **Interné a externé úložisko** - dáta, ktoré sú viditeľné iba z aplikácie alebo voľne dostupné pre používateľa
- **SQLite databáza** ukladá dáta formou štrukturovaných dát, ktoré sú viditeľné iba pre aplikáciu. Vhodné pre ukladanie informácii zo servera pokiaľ je aplikácia bez internetového pripojenia.

[1] https://tools.ietf.org/html/rfc3986
[2] https://developer.android.com/guide/components/aidl

- **Sieťové úložisko** môže byť vlastný server alebo cloudové riešenie od rôznych služieb. Vhodné pre zdieľanie informácii medzi zariadeniami sa správajú ako naištalované aplikácie používateľom, len s tým rozdielom, že sú inštalované s operačným systémom Android.

Aktivita

Aktivita je hlavná trieda, ktorá sa ukáže používateľovi po spustení aplikácie. Usporiadanie týchto aktivít je hierarchické. Aplikáciu môže tvoriť viacero aktivít.



Obr. 4.2: Životný cyklus aktivity

Pre realizáciu určitej akcii v jednom zo stavov je potrebné preťažiť jednu z metód z obrázka 4.2. V prípade, že sa otočí displej zariadenia, tak aktivita prejde do stavu *onPause()* a hneď do *onStop()*, kde sa uvoľnia všetky editovateľné používateľské dáta alebo

pozícia v zozname. Pre uloženie editovateľných dát je potrebné preťažiť *onSaveInstanceState()*,
v ktorej sa definujú, ktoré používateľské dáta sa majú uložiť. Keď prejde aktivita do stavu *onCreate()*, tak je možnosť obnoviť všetky používateľské dáta.

Fragmenty

Fragment predstavuje časť používateľského rozhrania, ktorý sa vkladá do aktivity. Má rozšírený cyklus aktivity a jeho životný cyklus priamo závisí od hostiteľskej aktivity v ktorej bol vytvorený. Slúži na zábraneniu redundancie logických častí.

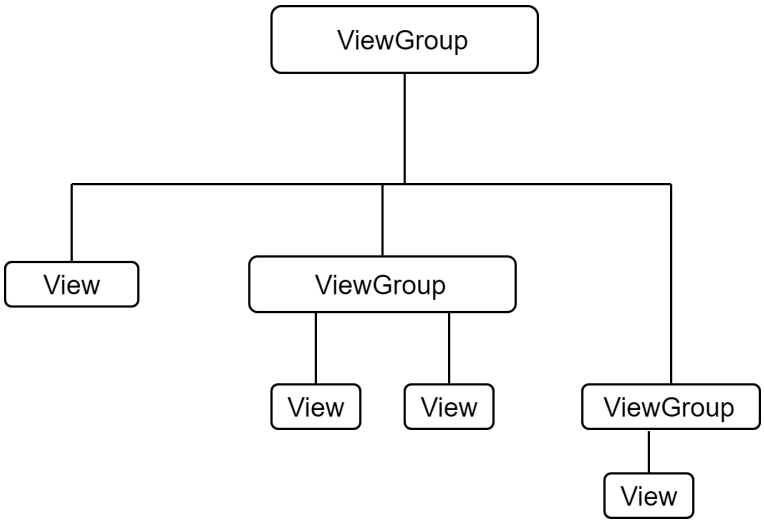
Príkladom fragmentu je vstavaná aplikácia telefónny zoznam kde je jedna aktivita so záložkami a zvyšok aktivity sa zobrazuje uživateľské rozhranie aktuálnej záložky.

4.3 Grafické rozhranie

Podkapitola sa zameriava na grafické rozhranie a jeho prvky, ktoré sa dajú vytvoriť
v systéme Android. Tieto prvky sú potrebné pre UI návrh mobilnej aplikácie ako aj samotnej funkčnosti.

Pohľady (Views)

Trieda Pohľad predstavuje základný stavebný prvok komponentov používateľského rozhrania a je zodpovedná za kreslenie a spracovanie udalostí. Od tejto triedy sú odvodené aj grafické komponenty ako napríklad *ImageView, TextView, EdiText* a mnoho ďalšieho. Pohľad môže byť začlenený do podtriedy *ViewGroup*, ktora môže obsahovať ďalšiu podtriedu ViewGroup alebo iné Pohľady, ktorý znázorňuje napríklad obrázok 4.3



Obr. 4.3: Zobrazenie ViewGroup

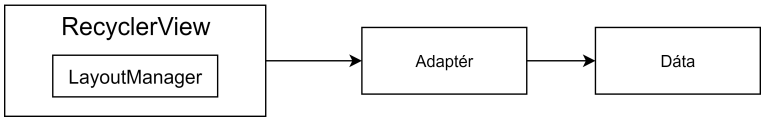
Kontajnery (Layouts)

Kontajnery sú podtriedami triedy ViewGroup a slúžia na jedinečné rozloženie Pohľadov, ktoré sa v nej nahromadili. Najbežnejšie používané:

- **LinearLayout** je rozloženie, ktoré usporiadava svoje potomky vertikálne alebo horizontálne. Svojim potomkom umožňuje nastaviť váhu výšky alebo šírky, a tým určiť relatívnu výšku alebo šírku potomka.
- **RelativeLayout** umožňuje určiť vzájomnú polohu medzi potomkami. Ak potomkovia sa chcú vzájomne ovplyvňovať v polohe zobrazenia, tak každý prvok musí mať unikátny identifikátor
- **FrameLayout** je navrhnutý tak, aby zablokoval oblasť na obrazovke a zobrazil jednu položku. Pohľady na deti sú nakreslené v zásobníku, pričom posledné pridané dieťa je na vrchole, čo znamená, že pohľady sa môžu prekrývať. Tento typ kontajneru je vhodný pri vytváraní tieňu pre Android 5.0 a nižší, kde nie je implementovaná plná podpora *Material Designu*[3]

RecyclerView

RecylerView je pokročilejší grafický prvok, ktorý umožnuje načítať väčšie množstvo dát. Obmedzuje počeť pohľadov ktoré je potrebné načítať a recykluje neviditeľné položky. Diagram RecyclerView je znázornený na obrázku 4.4 . Z diagramu je vidieť, že na zobrazenie pohľadov jednotlivých prvkov sa používa *LayoutManager*. Pre samotné zobrazenie dát je potrebné implementovať Adaptér podľa názvu návrhového vzoru Adaptér[9], ktorý zobrazí dáta.



Obr. 4.4: Diagram RecylerView

[3]https://material.io/

Kapitola 5

Implementácia

Následujúca kapitola podrobne rozoberie implementáciu mobilnej aplikácie na objednávanie drobných služieb. Pre vývoj mobilnej aplikácie sa používa vývojové štúdio s názvom Android Studio. Implementácia vychádza zo špecifikácie požiadavkov 2 ako aj z návrhu UI/UX 3.1. V nasledujúcich podkapitolách bude podrobne vysvetlená implemetácia *Firebase Realtime database* s ukážkami kódu a podporená diagramami tried. Podľa špecifikácií budú rozpísané jednotlivé prípady ako boli návrhnuté a taktiež budú podporené diagramami tried. V neposlednom rade sa v skratke predstavia použité knižnice a programovací jazyk *kotlin*.

5.1 Implementácia jadra aplikácie

Jadro aplikácie je implementované pomocou knižnice *Dagger2*, ktorá bude popísaná v kapitole 5.5.1. Vďaka tejto knižnici sa vytvára strom závislosti a napríklad pri implementácii *jedináčka (singleton)* dátového modelu pre prihlasovanie je zjednodušená tvorba jedináčka a odkazovanie sa ňan. Prihlasovací ViewModel pri zostavovaní aplikácie nepotrebuje referenciu na tento dátový model. Na následujúcich riadkoch kódu bude ukazané ako sa používa táto knižnica. Pre vytvorenie dátového modelu typu jedináčik sa v hlavičke triedy pomocou anotácie knižnice *Dagger2* napíše *@Singleton* a pri konštruktore *@Inject*.

```
@Singleton
class SignInDataModel @Inject constructor() {
...
}
```

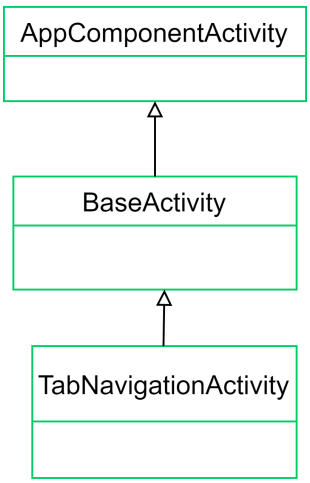
V bázovom ViewModely sa pomocou anotácie *@Inject* vloží referencia na prihlasovací dátový model. Táto premenná nemôže byť privátna.

```
abstract class BaseViewModel: ViewModel() {

...

@Inject
internal lateinit var dataModel: SignInDataModel
...
}
```

Prihlasovací ViewModel, ktorý dedí od bázového ViewModelu, pomocou rozhrania *AppComponent*, spája referenciu na dátový model. Bez tohto spojenia by nám aplikácia padala pri behu aplikácie.

```
class SignInViewModel: BaseViewModel() {

// region Internal Attributes

...

@Inject
internal lateinit var signinDataModel: SignInDataModel

...
// endregion Internal Attributes

// region Public Methods

init {
App.appComponent.inject(this)
}

...

// endregion Public Methods

}
```

5.2 Implementácia základnej aktivity



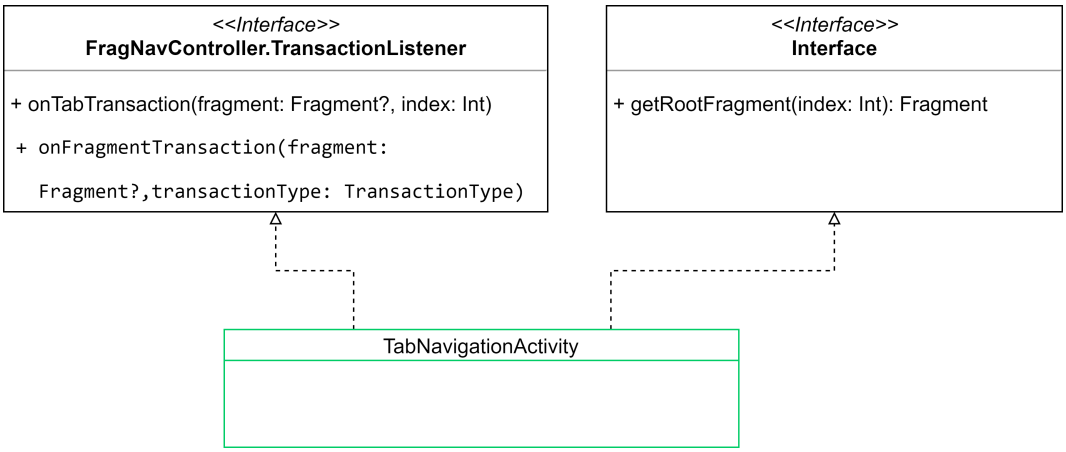
Obr. 5.1: Diagram tried pre TabNavigationActivity

Aktivita, ktorá sa spustí pri štarte aplikácie sa volá *TabNavigationActivity*. Z návrhu aplikácie z obrázka 3.1, je vidieť, že sa bude imlementovať spodná záložková navigácia.

Pre zobrazenie navigačnej ponuky v kontajneri sa používa vstavané riešenie, ktoré je súčasťou podpornej knižnice pre dizajn.

Z diagramu tried 5.1, je možné si všimnúť, že táto aktivita dedí z bázovej triedy (aktivity) a tá dedí z triedy z názvom *AppComponentActivity*, ktorá v sebe obshauje metódy zo životného cyklusu aktiviy, ktorý je znázornený na obrázku 4.2.

Pre zobrazenie jednotlivých fragmentov vo zvyšku aktivity sa používa knižnica *FragNavController*, ktorá uľahčuje prácu s prepínaním záložiek, zobrazovanie fragmentov a mnoho ďalšieho. Pre implementáciu tejto knižnice je potrebné implementovať rozhrania, ktoré sú vidieť na diagrame 5.2.



Obr. 5.2: Diagram tried implementácie rozhrania

5.3 Implementácia prihlásenia

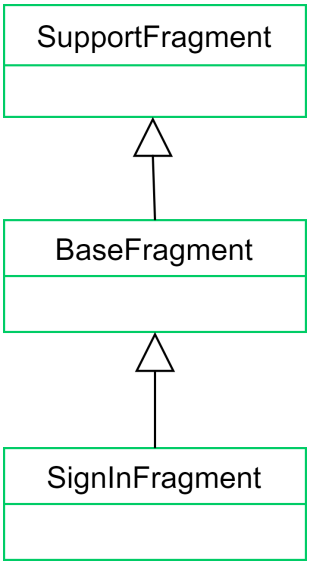
Pri implementácii prihlásenia sa vychádza z návrhu UX, ktorý je zobrazený na obrázku 3.1.

Prihlásenie je vo fragmente, čo znamená, že sa riadi životným cyklom základnej aktivity. Z diagramu tried 5.3 je zrejmé, že fragment vychádza z bázového fragmentu, ktorý dedí z fragmentu, ktorý je z podpornej knižnice, kvôli kompatibilite so staršími operačnými systémami Android.

Prihlasovací fragment má vlastný ViewModel, ktorý obsahuje metódy pre pre prihlásenie sa pomocou Facebooku, Gmailu alebo na otvorenie aktivity s prihlásením sa pomocou mena a hesla alebo registráciou.

Pre prihlásenie sa pomocou účtov od Facebooku a od Gmailu po úspešnom prihlásení cez tieto účty sa musí zavolať metóda na prihlásenie sa do mobilnej aplikácie pomocou overenie. Na nasledujúcich riadkoch bude ukážka implementácie:

```
fun signInWithCredential(token: String,
  socialNetwork: Int): Observable<FirebaseUser> =

    Observable.create { emitter ->
      val credential = when (socialNetwork) {
        SignInViewModel.Constants.SIGN_IN_FACEBOOK ->
          FacebookAuthProvider.getCredential(token)
```



Obr. 5.3: Diagram tried pre dedičnosť zobrazenie dedičnosti fragmentu

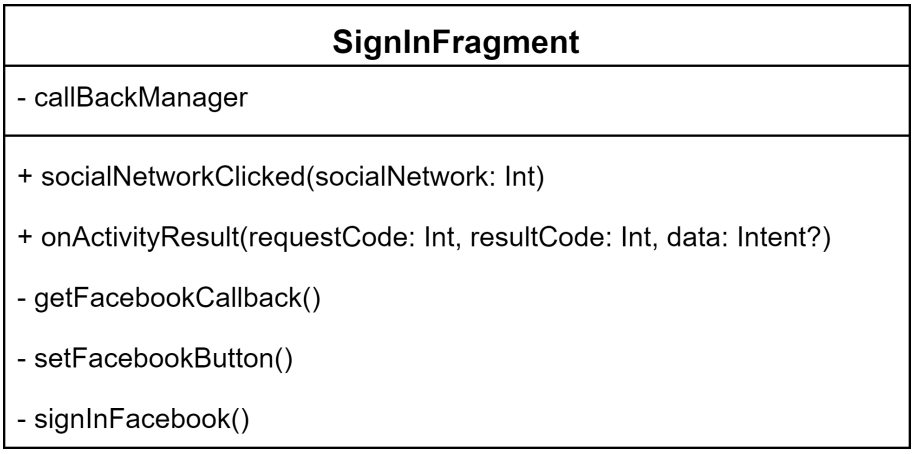
```
    else -> GoogleAuthProvider.getCredential(token, null)
  }

  auth.signInWithCredential(credential) \\Fireabse
    .addOnSuccessListener {
      Timber.d("signInWithCredential(): OnSuccessListener")
      emitter.onNext(it.user)
      emitter.onComplete()
    }
    .addOnFailureListener {
      Timber.e("signInWithCredential():
       OnFailureListener: error: %s", it)
      emitter.onError(it)
   }
}
```

5.3.1 Implementácia prihlásenia sa pomocou Facebooku

V kapitole 2.3.1 prihlásenie pomocou Facebooku je špecifikovaný tok dát pre prihlásenie sa. Pre implementáciu kódu sa použilo SDK od spoločnosti Facebook[1]. Na diagrame tried 5.4, sú zobrazené atribúty a metódy, ktoré sa použili pri implementácii prihlásenia pomocou Facebooku.

Pri implementácii sa musí upraviť aj konfiguračný súbor Manifest, do ktorého sa vložia informácie o Facebook Api Key pre danú aplikáciu v tomto prípade je to pre mobilnú aplikáciu na objednávanie drobných služieb.

[1] https://developers.facebook.com/docs/android/



Obr. 5.4: Atribúty a metódy pre implementáciu prihlásenia pomocou Facebooku

5.3.2 Implementácia prihlásenia sa pomocou Gmailu

V špecifikácii požiadavkov v kapitole 2.3.2 bol pomocou sekvenčného diagramu vysvetlený tok dát pre prihlásenie pomocou emailu od Googlu. Pre tento účel muselo byť implementované prihlásenie pomocou *Google OAuth 2.0* a následne využité api zavolanie prihlásenia do cloudového riešenie Google Firebase.

5.4 Implementácia bezpečnostných pravidiel

V podkapitole 3.6, je predstavené cloudové riešenie od Google Firebase. V tejto podkapitole bude vysvetlené ako sa implementujú bezpečnostné pravidlá. Tieto pravidlá nastavujú, kto môže zapisovať do databázy a kto z nej čítať. Na nasledujúcom bloku JSON kódu sú zobrazené bezpečnostné pravidlá, ktoré sú aplikované pre mobilnú aplikáciu.

```
{
  "rules": {
    "conversations":{
          "$con":{
            ".read": "root.child('userConversations')
                .child(auth.uid).child($con).child('id').val() === $con",
                ".write": "root.child('userConversations')
                .child(auth.uid).child($con).child('id').val() == $con"
            }
          },
          "userConversations":{
                "$uid":{
                        ".read": "$uid === auth.uid",
                        ".write": "auth != null"
                }
          },
          "offers":{
            ".read": "true",
              "$offer":{
```

```
                ".write": "data.child('uid').val()
                === auth.uid || (!data.exists() && newData.child('uid').val()
                === auth.uid)"
            }
        },
        "users":{
          "public":{
            ".read": "true",
                "$uid":{
                  ".write": "data.child('uid').val() === auth.uid ||
                      (!data.exists() && newData.child('uid').
                      val() === auth.uid)"
                }
          }
        }
  }
}
```

Pre neprihlaseného používateľa, ktorý je definovaný pomocou prípadu užitia 2.3, je aplikované pravidlo, že každý môže čítať z uzlu öfferäle zapisovať môže len používateľ, ktorý je prihlásený a v ponuke bude jeho unikátny identifikátor.

Z bloku kódu sa vysvetlí jedno z ďalších pravidiel, ktoré definuje, kto môže čítať a kto zapisovať do konverzácií.

Podľa pravidla pre konverzácie do uzlu *"conversations"*, ktorý ma potomka s unikátnym identifikátorom konverzácie, v ktorom sa nachádzajú jednotlivé správy medzi používateľmi má právo čítať a zapisovať používateľ, ktorý v uzle *üserConversations"* má svoj unikátny identifikátor a v potomkovi sa nachádza potomok s unikátnym identifikátorom konverzácie a v tom potomkovi je identifikátor konverzácie.

5.5 Implementačné detaily

Pri implementácii sa používali knižnice, ktoré zjednodušili prácu pri vytvorení jednotlivých vzhľadov ako aj pri implementácii obsluhy fragmentov a nahrádzanie *spätných volaní (callback)* pomocou reaktívneho programovania. V tejto podkapitole si v krátkosti predstavíme Kotlin, programovací jazyk, ktorý bol využitý pre implementáciu mobilnej aplikácie na objednávanie drobných služieb. Informácie čerpané k jazyku Kotlin sú z knihy [6]
a z oficiálnych stránok [1].

5.5.1 Kotlin

Kotlin je statický typovaný programovací jazyk, ktorý spája objektovo orientovaný prístup s funkcionálnym. Beží nad JVM, čo znamená, že je plne kompatibilný s programovacím jazykom Java s možnosťou kompilácie do Javascriptu.

Jeho veľkou výhodou oproti jazyku Java je, že má takzvanú ***null safe*** politiku, čo znamená, že primárne žiadna premenná nemôže byť **null**.

Použité knižnice

Gson je knižnica pomocou ktorej môžeme serializovaqť a deserializovať triedu do JSON stringu.

FragNavController - knižnica na jednoduchšiu správu a prácu s fragmentami , pre záložkovú navigáciu

RoundImage knižnica, ktorá sa využíva na vytvorenie okrúhlych obrázkov pri profile a pri správach

RxJava knižnica na tvorenie asynchrónnych požiadavkov a jednoduchšiu obsluhu vlákien, týchto požiadavkov

Support knižnice knižnica na oklieštenú podporu nových funkcií na staršom opereačnom systéme Android

Kotlin podpora pre jazyk Kotlin

Facebook knižnica pomocou ktorej sa implementovalo prihlasovanie pomocou Facebook

Dagger2 Knižnica slúžiaca na vytváranie vôzby závislosti. Zjednodušuje prácu pri vytváraní jednináčkov ako aj určuje aké komponety a ako budú na sebe závislé.

Firebase Knižnica, ktorá ponúka kompelxné riešenie pre správu serverovej časti ako aj vzdialenú konfiguráciu aplikácie.

Kapitola 6

Testovanie

Kapitola sa zaoberá testovaním implementácie. V kapitole bude ootestovaných pár prípadov zo špecifikácie požiadavkov 2.

6.1 Testovanie prihlásenia pomocou riešeni od spoločnosti Google a Facebook

Pri implementácii ako aj po následnej implementácii bolo tstované prihlásenie cez Gmail a Facebook.

6.1.1 Testovanie prihlásenia pomocou Facebook

Pri implementácií a následnom testovaní sa zistilo, že aplikácia sa prihlasuje cez api rozhranie mobilnej aplikácie od spoločnosti Facebook. Pokiaľ by sa chcel používateľ, ktorý nie prihlasený do aplikácie facebook, tak by musel najskôr musel odhlásiť aktuálneho používateľa a prihlásiť seba. Toto riešenie je natívne pre Android od spoločnosti Facebook.

6.1.2 Testovanie prihlásenia pomocou Gmail

Prihlásenie pomocou Gmailu, dáva na výber z akého účtu sa chce prihlásiť ale ak používateľ, nie je prihlasený na danom mobilnom zariadení, tak najskôr tohto používateľa prihlási do aplikácie Gmail a potom následne do aplikácie TouchDeal.

6.2 Testovanie pridávanie ponuky

testovanie pridávanie ponuky bolo testované počas vývoja ako aj následne bol ootestované používateľom, s ktorým prebhla aj úšpesná transakcia.

Kapitola 7

Záver

Cieľom bakalárskej práce bolo navhrnúť a implementovať aplikáciu na objednávanie drobných služieb. Pri návrhu som použil UML diagramy pre špecifikáciu požiadavkov. V tejto kapitole som pomocou diagramu prípadu použitia špecifikoval, aké prípady môžu nastať, keď je používateľ prihlásený a kedy nie. Z toho vyplynuli špecifikácie pomocou sekvenčných diagramov, pomocou ktorých som si pomaly tvoril architektúru mobilnej aplikácie. Po tomto návrhu som pomaly začať tvoriť obrazovky, ktoré sa budú zobrazovať podľa materiálneho dizajnu. Potom som tieto obrazovky s logickou časťou naimplementoval. V časti testovanie som otestoval aplikáciu na reálny prípad použitia aplikácie. Testovaná aplikácia bola funkčná a je zobrazená aj reálny prípad objednania reálnej služby.

V tejto bakalárskej práci sa mi bohužiaľ nepodarilo dodržať dizajn podľa materiálneho vzhľadu, ktorý je odporúčaný od spoločnosti Google. Bakalárska práca je v stave pre ďalší vývoj, kvôli dobrému architektonickému návrhu.

Bakalársku prácu by som chcel do budúcna upraviť a rozšíriť, hlavne sa zamerať na design a upraviť ho podľa odporúčaní materiálneho vzhľadu. Ďalej by som chcel ju rozšíriť o hodnotenie používateľov. Taktiež zlepšiť metódy prihlásenia a overenie používateľov, ktorými by boli overenie používateľa podľa kreditnej karty a zadania kódu, ktorý by dostal cez SMS.

Literatúra

[1] Kotlin Programming Language. [Online; navštívené 22.02.2018].
URL https://kotlinlang.org

[2] AndroidDevelopers: Develop Apps. [Online; navštívené 5.03.2018].
URL https://developer.android.com/

[3] Bajaník, F.: *Implementujte Android aplikaci pro stahování a poslech audio podcastů*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18060

[4] Bernard, B.: *Prezentační vzory z rodiny MVC*. May 2009, [Online; navštívené 5.5.2018].
URL https://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

[5] Firebase: Firebase. [Online; navštívené 20.01.2018].
URL https://firebase.google.com/

[6] Jemerov Dmitry, I. S.: *Kotlin in action*. NY: Manning Publications Co., 2017, ISBN 978-1617293290.

[7] Jim Arlow, I. N.: *UML 2 a unifikovaný proces vývoje aplikací*. Brno: Computer Press, 2007, ISBN 978-80-251-1503-9.

[8] Object Management Group (OMG): Unified Modeling Language Specification, Version 2.5.1. OMG Document Number formal/17-12-05 (https://www.omg.org/spec/UML/2.5.1/), December 2017.

[9] Pecinovský, R.: *Návrhové vzory*. Computer Press (CPress), 2007, ISBN 978-80-251-1582-4.

[10] Wikipedia contributors: Model–view–viewmodel — Wikipedia, The Free Encyclopedia. 2018, [Online; navštívené 4.5.2018].
URL https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93viewmodel&oldid=839019781

[11] Wikipedie: Model-view-controller — Wikipedie: Otevřená encyklopedie. 2016, [Online; navštívené 4.5.2018].
URL https://cs.wikipedia.org/w/index.php?title=Model-view-controller&oldid=13938376

Príloha A

Obsah CD

Priložené CD obsahuje

- **documentation** - adresár s textom bakalárskej práce v šablóne LaTeXvrátane obrázkov
- **source-code** - adresár so zdrojovými textami aplikácie
- **bachelor-thesis.pdf** textová časť v PDF
- **Readme.txt** - súbor s informáciami