FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Android aplikace DayWork.cz: implementace aplikace pro pracovníky
Student:	Michal Sousedík
Vedoucí:	Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Portál DayWork.cz slouží pro párování nabídek a poptávek brigád. Práce je součástí projektu vytvoření Android aplikace pro pracovníky a Android aplikace pro firmy. Cílem této práce je především vývoj aplikace pro pracovníky a též vývoj obecné funkčnosti zpráv.

1) Implementujte aplikaci pro pracovníky na základě analýzy a architektury vytvořené v paralelně běžící bakalářské práci, zaměřte se zejména na následující funkčnosti:
- profil (dovyplnění profilu po registraci, úprava, zobrazení)
- seznam brigád (filtry, accepted, unaccepted, pending)
- detail brigády (akce - napsat zprávu, požádat o brigádu, potvrdit účast, zobrazení mapy).

2) Proveďte návrh, implementaci a integraci modulu zpráv, a to části v Android aplikaci i podpory na serveru. Zaměřte se zejména na synchronizaci napříč zařízeními.
3) Aplikaci a moduly řádně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 29. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Android aplikace DayWork.cz: implementace aplikace pro pracovníky

Michal Sousedík

Katedra softwarového inženýrství

Vedoucí práce: Ing. Robert Pergl, Ph.D.

11. května 2018

Poděkování

Rád bych zde poděkoval svému vedoucímu bakalářské práce Ing. Robertu Perglovi, Ph.D. za rady a pomoc při realizaci tohoto projektu. Dále bych rád poděkoval Michalu Mročkovi, za jeho pomoc a konzultace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 11. května 2018 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Sousedík, Michal. *Android aplikace DayWork.cz: implementace aplikace pro pracovníky*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato bakalářská práce se zabývá tvorbou dvou modulů mobilní aplikace určené pro operační systém Android, sloužící k hledání brigád Daywork.cz. Zejména pak implementací modulu, který budou využívat zájemci hledající možnosti dočasného přivýdělku. Jedná se o zobrazení dostupných brigád, úpravy pracovníkova profilu či filtru zobrazovaných brigád.

Práce taktéž obsahuje návrh a implementaci modulu sloužícího ke komunikaci a synchronizaci zpráv mezi pracovníkem a firmou. Výsledkem práce je funkční řešení obou modulů, které jsou zapracovány do aplikace vytvořené v paralelně řešené bakalářské práci.

Klíčová slova aplikace, brigáda, pracovník, modul, synchronizace, Daywork, chat

Abstract

This bachelor thesis deals with development of two different modules in mobile application for operating system Android, which helps people with their quest

to find a perfect part time job. First part is especially about implementation of a module which will be used by users who are looking for a job. I will mainly deal with the displaying of a various part time jobs offer, editing current user profile and his filter. Thesis contains design and implementation of a complex chat module, which mediates communication between the worker and company. The result is a fully functional solution of both modules, which will be added into application created in parallel running bachelor thesis.

Keywords application, part time job, worker, mudule, synchronization, Daywork, chat

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Úvod

Který středoškolský nebo vysokoškolský student si v dnešní době nepotřebuje při studiu něco málo přivydělat? Není žádnou novinkou, že webové portály se snaží těmto jedincům pomoci v jejich honbě za penězi. Najít dobře placenou brigádu, která by splňovala požadavky uchazeče a zároveň umožnila rychlou komunikaci se zadavatelem ovšem není žádná legrace. Touto problematikou se již nějakou dobu zabývá webový portál Daywork.cz, který byl vytvořen studenty naší fakulty v předchozích letech. Nově se však tvůrci tohoto projektu rozhodli přidat k portálu i mobilní aplikaci.

Toto téma mě oslovilo zejména kvůli možnosti spolupracovat na reálném týmovém projektu. Již dlouhou dobu jsem se chystal osvojit si vývoj mobilních aplikaci pro Android platformu a zde se mi naskytla příležitost toho dosáhnout.

Mobilní aplikace se skládá ze dvou separátních částí. První je určena čistě pro potencionálního brigádníka a druhá pro firmu zadávající brigádu. V mé práci se zabývám implementací aplikace pro pracovníky na základě architektury a designu vytvořených v paralelně řešených bakalářských pracích. Zejména pak umožním pracovníkovi upravit jeho profil a filtry podle kterých se mu párují nabídky jednotlivých brigád. Zobrazím přehled těchto brigád rozdělený do kategorií podle toho, které vyhovují jeho schopnostem, které se mu líbí, o které má zájem anebo které jsou mu již akceptovány zadavatelskou firmou.

Jak tomu už, tak někdy bývá, každému se občas stane, že je limitován slabým internetovým připojením například v metru nebo u babičky, s tím ovšem mé řešení počítá a přehled dříve načtených dat zobrazí i za této situace.

Rychlá zpětná vazba při hledání nové brigády je v dnešní době esenciální, a proto součástí mé práce je také návrh a implementace chatovacího modulu,

který zprostředkuje komunikaci mezi pracovníkem a firmou prostřednictvím komplexního chatu. V této části se budu nejvíce věnovat synchronizaci zpráv mezi mobilními zařízeními a webovým portálem.

Správnost celého řešení na závěr otestuji řadou jednotkových a instrumentálních testů.

Kapitola 1

Cíl a metodologie

1.1 Týmová práce

Vývojem této mobilní aplikaci se zabývají celkem tři bakalářské práce. Rozdělení oblastí vývoje je následující:

- Mroček, Michal – vývoj celkového jádra aplikace;
- Novotný, Kryštof – tvorba uživatelského rozhraní, vývoj části pro firmy;
- Sousedík, Michal – návrh a implementace chatovacího modulu, vývoj části pro brigádníky.

1.2 Cíl práce

Hlavním cílem této bakalářské práce je vytvoření části mobilní aplikace pro platformu Android, která zpřístupní funkcionality pracovníkům při hledání jejich brigády na portálu Daywork.cz. Celé řešení bude postaveno na architektuře a designu vytvořených v paralelně řešených bakalářských pracích. Pracovníkovo uživatelské rozhraní se skládá z následujících částí:

- **Profil** – Zobrazím potencionálnímu pracovníkovi jeho uživatelský profil, kde si bude schopný změnit své osobní a kontaktní údaje. Současně mu umožním CRUD operace u variace kvalifikačních údajů. Jedná se o pracovníkovy pracovní zkušenosti, jazykové schopnosti a počítačové dovednosti. Nejvíce se zaměřím na přidávání a zobrazení fotografií, pomocí kterých si uživatel zlepší své šance při hledání brigády.

- **Filtr** – Pracovníkovy preference týkající se místa a typu brigády se můžou časem změnit. Je potřebné uživateli umožnit změnu dosavadního filtru, podle kterého se mu párují vhodné brigády.
- **Seznam brigád** – Připravím kategorizovaný přehled pracovníkových brigád rozdělených podle toho, v jakém stavu se zrovna nachází. Stav je založen na aktuálním vztahu pracovníka a zadavatelské firmy k dané brigádě. Můžeme hovořit o celkové nabídce brigád, které jsou pro pracovníka vhodné, nebo také o seznamu brigád, které označil za své oblíbené, o které má zájem, které čekají na jeho schválení, anebo které jsou mu již potvrzeny zadavatelskou firmou.
- **Detail brigády** – Zobrazím uživateli podrobný výčet informací o kterékoliv brigádě z jeho seznamu. O zvolenou brigádu zde může pracovník projevit zájem nebo potvrdit či odmítnout svou účast. V případě zájmu může zahájit konverzaci se zadavatelskou firmou nebo se na mapě podívat, kde brigáda probíhá.

Dalším cílem je vytvoření chatovacího modulu, který značně usnadní komunikaci mezi pracovníkem a firmou. Zde se zaměřím zejména na synchronizaci napříč zařízeními, zobrazením relevantních notifikací a způsobem načítání starších zpráv v konverzaci.

Závěrem vytvořím řadu jednotkových a instrumentálních testů, které ověří správnost výše zmíněného řešení.

1.3 Metodologie

Tato práce je složená z několika kapitol. V kapitole Rešerše 2 popíši společnost, která tuto práci zadala, nastíním architekturu celého projektu a analyzuji technologie, které se mi budou hodit při vývoji aplikace. Další kapitola Analýza 3 bude o identifikaci požadavků pro tvorbu chatovacího modulu. V kapitole Návrh 4 vytvořím abstraktní pohled na řešení problémů. Začnu od těch základních jako je způsob práce s daty a později přejdu ke konkrétním návrhům modulů. V implementační části 5 ukáži zajímavosti, se kterými jsem se při implementaci setkal. Na závěr ukáži způsoby, kterými byla aplikace testována 6.

Pro tvorbu diagramů použiji UML 2.0, které vytvořím v nástroji Enterprise Architect. Pro vývoj mobilní aplikace využiji Android Studio 3.0.1. a pro verzování systém Git.

Kapitola 2

Rešerše

2.1 Představení zadavatelské společnosti

Daywork.cz je chytrý portál, který dokáže spárovat nabídky brigád s pracovníky dle jejich nastavených údajů v profilu. Daywork je primárně orientovaný na studenty středních a vysokých škol, kteří si chtějí přivydělat a nasbírat zkušenosti z pracovišť jako jsou kanceláře, hotely, restaurace, bary, obchody, veletrhy či různé eventy.

Projekt byl spuštěn 10. října 2017 a od té doby má celou řadu uživatelů a spokojených klientů z řad studentstva i firem.

Portál by vyvinutý studenty Fakulty informačních technologií na ČVUT a nově přibude k webu také mobilní klient.

2.2 Použité jádro aplikace

Celková architektura mobilní aplikace zejména programovací jazyk a architektonický styl, byly zvolené na základě analýzy, která je součástí jiné bakalářské práce [1].

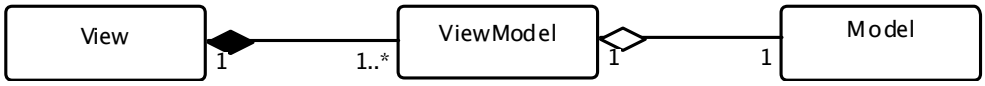
2.2.1 Programovací jazyk

Programovacích jazyků pro vývoj aplikace na Android je několik. Použít multiplatformní framework pro náš tým nemá velký smysl, neboť vyvíjíme čistě pro Android. Tudíž jsme se rozhodli vydat cestou nativního vývoje [1].

U nativního vývoje nezbývá nežli vybrat mezi Javou a Kotlinem a jelikož máme v týmu větší zkušenosti s Javou, rozhodli jsme se tento jazyk použít pro vývoj naší aplikace.

2.2.2 Architektonický styl

Během vývoje aplikací lze použít řadu architektonických stylů. Pro naši aplikaci byl vybrán MVVM.



Obrázek 2.1: Class diagram - MVVM architektura [1]

Tento vzor je podobný jinak známému MVC vzoru. Dokonce komponenty M (Model) a V (View) mají v obou vzorech podobnou strukturu. V čem se však zásadně liší je C (Controller) a VM (View Model).

Model Reprezentuje business logiku. Není nikterak svázaný s view ani viewmodelem, což ho dělá znovupoužitelným.

View Zobrazuje pozorované proměnné poskytované viewmodelem. Je navržen tak, aby bylo možné spojit více view k jednomu viewmodelu.

View Model Je odpovědný za obalení modelu a přípravu pozorovatelných dat pro potřeby view. Současně umožňuje view předávat události modelu [2].

Na základě tohoto vzoru je realizována struktura, kterou bude tato bakalářská práce dodržovat a dále rozšiřovat. Struktura se skládá ještě ze dvou komponent:

Repozitář Cílem repozitáře je zprostředkovat viewmodelu funkce, které komunikují buď s lokální databází nebo se serverovým API.

Pracovní vlákno Pracovní vlákno manipuluje s daty, přičemž pracuje s lokální databází nebo přímo se serverem. Je voláno repozitářem a běží asynchronně na pozadí [1].

2.2.3 Push zprávy

Jak již bylo dříve zmíněno, mobilní aplikace je klientem již fungujícího serveru. Tudíž při vývoji musíme brát v potaz také možné změny provedené uživatelem na webovém portálu, které se musí zákonitě projevit i na mobilním zařízení. Vezměme si například posílání zpráv mezi uživateli. Pokud by příjemce neobdržel žádné upozornění, zpráva by se mu zobrazila až ve chvíli,

kdy si otevře aplikaci a jde se podívat na své přijaté zprávy. Tento způsob nesplňuje požadavek na rychlou zpětnou vazbu. Z toho důvodu byl do jádra aplikace zakomponován modul pro příjem a zpracování zpráv ze serveru. Existují dva druhy zpráv:

1. **Hlasitá zpráva** – po doručení a zpracování této zprávy dojde k upozornění uživatele,
2. **Tichá zpráva** – zpracování probíhá vždy na pozadí.

Pro správné použití výše zmíněné funkcionality je zapotřebí:

- Na hostitelském serveru:
  1. Vytvořit jedinečný identifikátor a strukturu odesílané zprávy.
  2. Odeslat požadavek na Firebase cloud messaging, který bude obsahovat zprávu pro příjemce a identifikátor příjemcova zařízení.
- V mobilní aplikaci:
  1. Upravit metodu `deserialize(PushKey key, JsonObject root)` ve třídě `PushDeserilize`, tak aby třída věděla na jaký objekt deserializovat příchozí zprávu z Fcm.
  2. Přesměrovat přicházející zprávu na chtěný handler, který podle přidělené funkcionality zareaguje.

2.2.4 Použité komponenty

Během vývoje použiji některé abstraktní třídy implementované v rámci jádra aplikace.

BaseActivity Třída, od které dědí ostatní aktivity v aplikaci. Umožňuje získat view model, poskytuje `PreferenceHelper` a zajišťuje správné chování aplikace při změně uživatele.

BaseToolbarActivity Zjednodušuje zděděným aktivitám práci s komponentou `ToolBar`.

2.3 Datové listy

Při prvním pohledu na design [3] aplikace lze okamžitě identifikovat nejčastější komponentu, se kterou se budu potýkat. Jedná se o datový list o dynamickém množství položek. V této kapitole se budu věnovat nejpoužívanějším způsobům řešení zobrazování dynamického množství dat.

2.3.1 ListView

Komponenta `ListView` je určená ke shromažďování položek a jejich následnému vertikálnímu zobrazení v posuvném listu, kde se jednotlivé položky nacházejí jedna pod druhou [4]. Pořadí zobrazených položek odpovídá pořadí položek v přiložené kolekci. Samotná komponenta nezná detaily o svém obsahu. Takovým detailem je kupříkladu typ zobrazované položky, což dokazuje že slouží pouze k zobrazení, nikoliv k ukládání dat. `ListView` si o svůj obsah požádá adaptér v případě potřeby, například pokud se uživatel na svém mobilním zařízení posouvá nahoru nebo dolů za hranice načtených dat.

2.3.2 RecyclerView

S příchodem knihovny android.support.v7.widget se objevil `RecyclerView`. Tato komponenta je vylepšeným následovníkem `ListView`, se kterým sdílí řadu funkcionalit. Kromě zlepšení výkonu `RecyclerView` přináší lepší kontrolu nad zobrazovanými daty nebo lepší manipulaci se zobrazenými položkami za běhu programu kvůli akcím uživatele nebo změnám dat v databázi. Tato komponenta bude užitečná, pokud máme velké množství stejného typu dat, které potřebujeme zobrazit a s velkou pravděpodobností víme, že se nám nevejdou na displej zařízení. Uživatel musí listovat nahoru a dolů, aby viděl potřebná data. V tuto chvíli komponenta recykluje již neviditelné položky a znovu je používá v momentě, kdy se mají stát znovu viditelnými [5].

2.4 Adaptéry

Adaptér lze považovat za most spojující UI komponenty a zobrazovaná data. V mém případě bude naplňovat data do datového listu. Těchto adaptérů je celá řada. V následující části se podívám na některé z nich a uvedu jejich případy užití.

- **Rozšíření BaseAdapter** – Pokud chceme naprostou kontrolu nad událostmi v adaptéru, rozšíření této abstraktní třídy je perfektním způsobem, jak toho dosáhnout. `BaseAdapter` je úplným základem a je tedy potřeba veškerou funkcionalitu vytvořit ručně.
  - **ArrayAdapter** – ideální jako adaptér pro UI komponenty, které vyžadují kolekci menšího množství položek jako počáteční data, například `ListView` nebo `Spinner`;
  - **CursorAdapter** – získává svá data přímo z SQLite databáze [6];
  - **SimpleCursorAdapter** – mapuje sloupce z kurzoru v databázi na komponenty `TextView` nebo `ImageView`, které jsou definovány v XML souboru;

  - **SpinnerAdapter** – spojuje UI komponentu `Spinner` a jeho data, umožňuje definovat dva odlišné typy pohledů: jeden, který zobrazuje data v samotné komponenty a druhý, který zobrazuje data v rozbalovacím seznamu, když uživatel klikne na `Spinner`.

- **RecyclerView.Adapter** – `RecyclerView` představil úplně nový druh adaptéru. Hlavní funkcionalita zůstává stejná jako v `BaseAdapteru`, avšak jsou zde jisté rozdíly. Zatímco v `ListView` byl `ViewHolder` volitelný, v `RecyclerView` je povinný [7]. `ViewHolder` je třída, která si drží instance zobrazovaných dat a tím pomáhá zlepšit výkonnost adaptéru. Tento adaptér taktéž umožňuje UI komponentu notifikovat nejenom o změně svých dat, ale dokáže i sdělit jakoukoliv CRUD operaci. Tím umožňuje řadu animací při přidávaní, odebírání nebo upravování položek v listu a neplýtvá časem na vytváření celého pohledu znovu jenom kvůli změně jedné položky.

- **PagedList.Adapter** – V případě, že zobrazovaných dat je velké množství, vyplatí se adaptéru dodávat data pouze v případě, že jsou potřeba. To umožňuje `PagedList.Adapter`, který dokáže porovnávat data, která již dostal, s nově příchozími, a ve chvíli potřeby si žádat o nové [8]. Adaptér získává z lokální databáze stránkovaná data v podobě `PageListu` a porovnává je s daty, která již dříve získal, aby mohl určit, které položky se změnily, které byly přidány nebo které byly odebrány. Tím zabraňuje velkému množství animací na displeji zařízení a zlepšuje výkon celé aplikace.

2.5 Manipulace s fotografiemi

Veškeré části aplikace vyjma filtru spojuje jedna věc, a to zobrazení fotky na základě přidělené adresy url. Manipulace s fotografiemi bez použití externí knihovny není nic snadného. V rámci této kapitoly se zaměřím na analýzu nejpoužívanějších knihoven načítajících fotografie.

Picasso Android Picasso je knihovna sloužící k načtení a zpracování obrázků. Byla vyvinuta společností Square Inc. Mezi vývojáři je velmi oblíbená zejména díky své jednoduchosti. Mnohdy stačí pouze jeden řádek kódu pro použití požadované funkcionality [9].

UImageLoader UIL knihovna byla vyvinutá Sergeyem Tarasevichem a je považována za kvalitní a vysoce customizovatelný nástroj pro načítaní obrázků, jejich následné ukládání a zobrazení [10]. Vývojář má možnost zasáhnout do veškerých částí procesu, které manipulují s obrázky. Od vláken spouštějících procesy stahovaní, přes dekóder až po dočasné ukládání obrázku.

Fresco Projekt vytvořený Facebookem. Stejně jako výše zmíněné, se i tato knihovna stará o načítání a zobrazování obrázků. Obrázky dokáže načíst z internetu, z lokálního uložiště nebo lokálních zdrojů. Přidanou hodnotou je zobrazení dočasného placeholderu dokud není požadovaný obrázek připraven k použití. Má dvouúrovňovou cash; jednu v paměti a druhou v interním uložišti [11]. Umožňuje transformaci obrázků nejen kolem jeho centrálního bodu, ale kolem kteréhokoliv místa na jeho ploše.

Glide Glide se od ostatních knihoven liší hlavně svým rychlým a spolehlivým načítáním obrázků. Zaměřuje se na plynulý pohyb při scrollování a poskytuje jednoduché API s výkonným a rozšiřitelným dekóderem vstupních zdrojů. Glide pracuje efektivně také s videozáznamy nebo animovanými GIFy. Ve výchozím nastavení knihovna používá zásobník založený na `HttpUrlConnection`, současně však obsahuje knihovny, které se připojují k `Google Volley` nebo ke knihovně `OkHttp` [12].

	Picasso	UImageLoader	Fresco	Glide
Podpora Volley a OkHttp	ANO	ANO	ANO	ANO
Podpora formátu Gif	NE	NE	ANO	ANO
Disk + memory cash	ANO	ANO	ANO	ANO
Transformace obrázků	ANO	NE	ANO	ANO
Načítací placeholder	ANO	NE	ANO	ANO
Podpora thumbnail	NE	NE	NE	ANO

Tabulka 2.1: Výsledek analýzy rozdílů mezi knihovnami, které umožňují manipulaci s fotografiemi

2.6 Zobrazení informačních hlášek

Často nastává situace, že si chce uživatel uložit svá data na server. V případě, že však nastane chyba zapříčiněná buď špatným internetovým připojením nebo chybou na serveru, je uživatel informován o vzniklém problému prostřednictvím pop-up zprávy. Rozdíly mezi komponentami `Snackbar` a `Toast` jsou zachyceny v tabulce 2.2.

Snackbar `Snackbar` je Android design komponenta představená jako součást Material Design. Poskytuje zpětnou vazbu o operaci tím, že zobrazí krátkou zprávu ve spodní části obrazovky. `Snackbar` může obsahovat akci, která může ovlivnit budoucí chování aplikace. Například po odstranění emailu oznámíme uživateli, že email byl úspěšně odstraněn a dáme mu možnost tuto akci zvrátit tlačítkem UNDO.

Toast Toast je jiná komponenta z řady Android design, která existovala ještě před `Snackbarem`. Poskytuje jednoduchou zpětnou vazbu o operaci pomocí malé pop-up zprávy, která se může objevit v kterékoliv části obrazovky mobilního zařízení [13].

	Toast	SnackBar
Přidán v API levelu	1	23
Závislý na aktivitě	NE	ANO
Uživatelská událost	NE	ANO
Odstranitelný gestem	NE	ANO
Použití	informační zprávy	zprávy vyžadující pozornost uživatele

Tabulka 2.2: Výsledek rozdílů mezi komponentami Toast a SnackBar

2.7 Tvorba dialogů

Doplnit menší množství informací například do stávajícího profilu nebo si od uživatele vyžádat povolení k použití jeho soukromých dat jsou události, které jsou součástí většiny novodobých aplikací. Způsoby prezentace tohoto požadavku se zabývám v následující části.

AlertDialog Tento dialog se používá především pokud je třeba se uživatele zeptat na uzavřenou otázku s možností předem připravených hodnot (například ano nebo ne) [14]. Současně umožňuje zobrazit až tři tlačítka a případně seznam řetězců s tím, že uživatel vybere jeden klepnutím. Lze ho ovšem customizovat podle přání vývojáře, a tak do něj přidat například UI komponentu `EditText` pro úpravu textu.

DialogFragment Speciální fragment, který zobrazuje okno dialogu a nachází se nad oknem aktivity, ze které byl vytvořen. Typicky se používá pro zobrazení potvrzovacích dialogů nebo dialogů, které vyzývají uživatele k vyplnění údajů bez potřeby přechodu do jiné aktivity. Používat pouze komponentu `Dialog` pro vytvoření obsahu překrývajícího obsah jiný, je od verze Androidu 3.0 považováno za špatnou praxi [15]. Vychází se z toho, že aktivity umí s fragmenty pracovat velmi dobře, a tak se nemusí starat ještě o další typ objektů. Tak vznikla třída `DialogFragment`, která obohacuje třídu `Fragment` o schopnost zobrazit se jako dialog, zatímco neztrácí svou schopnost zobrazit se jako view.

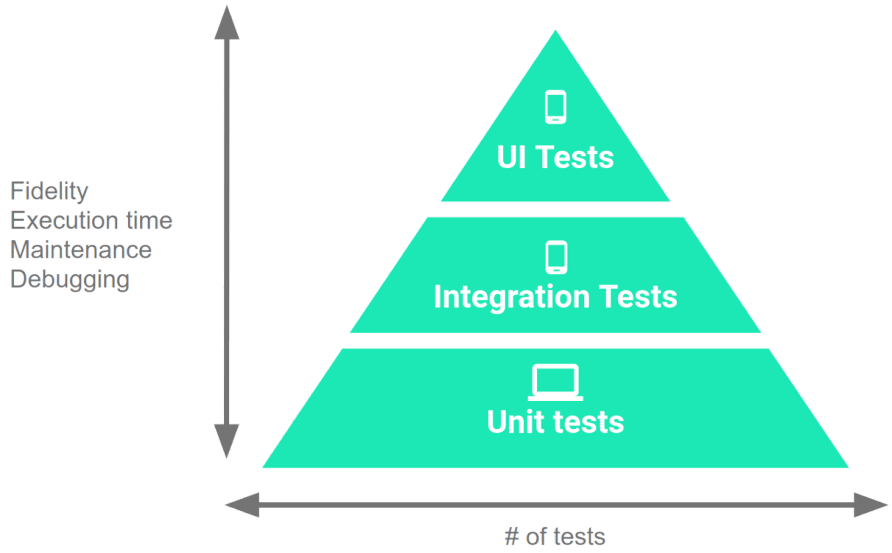
2.8 Testování

V této podkapitole se podívám na možnosti testovaní aplikací pro platformu Android.

2.8.1 Testovací pyramida

Testovací pyramida, zobrazená na obrázku 2.2, ilustruje jakým způsobem by měla aplikace pokrývat tři kategorie testů, které se anotují:

1. **@SmallTest** – jsou jednotkové testy, které běží odděleně od produkčního systému. Typicky mockují každou větší komponentu a běží velmi rychle na mobilním zařízení [16]. Slouží k ověření fungování a korektnosti implementace menších částí systému.
2. **@MediumTest** – jsou integrační testy, které spojují několik komponent a běží na emulátoru nebo na skutečném mobilním zařízení. Tyto testy umožňují kontrolovat životní cyklus android komponent.
3. **@LargeTest** –jsou UI testy, které v rámci svého běhu prochází celé workflow aplikace.



Obrázek 2.2: Testovací pyramida [16]

2.8.2 Espresso

Espresso je UI testovací framework, který umožňuje vytvoření automatizovaných UI testů aplikace. Snaží se důvěryhodně simulovat interakci uživatele

při jeho užívání aplikace. Tak se může vývojář snadněji vyhnout neočekávaným výsledkům. Framework slouží pro otestování uživatelské interakce pouze v rámci jedné aplikace. Pro testování napříč aplikacemi je lepší použít framework UIAnimator.

Kapitola 3

Analýza

Mobilní aplikace je rozdělena do několika modulů, které nejsou na sobě vzájemně závislé. Mým cílem je vytvořit modul, který umožní komunikaci mezi dvěma uživateli a modul, který poskytne funkcionalitu na základě funkčních a nefunkčních požadavků analyzovaných v jiné bakalářské práci [1].

3.1 Chatovací modul

Předtím nežli začnu implementovat chatovací modul, musím analyzovat jeho aktuální stav a navrhnout nové řešení. Po seznámení s aktuálním stavem problému, rozeberu požadavky na základě specifikace zadání.

3.1.1 Aktuální stav problému

Uživatelé webového portálu aktuálně používají pro komunikaci zasílání zpráv. Důvodů k zaslání zprávy jinému uživateli je hned několik. Pokud je uživatelem firma poskytující brigády, důvodem může být například pozvání brigádníka k osobnímu pohovoru, detailní oznámení o změně stavu brigády nebo získání potřebných informací o brigádníkovi. Brigádník používá tento způsob komunikace hlavně kvůli doplňujícím otázkám ohledně brigády, o kterou by měl případně zájem.

Po zaslání zprávy dojde příjemci email a notifikace o příchozí zprávě (pokud má tuto službu povolenou). Příchozí zprávu si příjemce může zobrazit buďto v emailu nebo po otevření webového portálu. To značně zpomaluje celý proces komunikace, neboť uživatel nemůže být na webovém portálu celý den a nemůže tedy odpovídat okamžitě po obdržení zprávy.

3.1.2 Rozbor požadavků

Sběr požadavků probíhal na základě analýzy webového portálu spolu s počátečními představami zadavatelské společnosti. Další požadavky byly přidány v průběhu návrhu chatovacího modulu a po získání finální architektury od člena vývojářského týmu. Následující seznam představuje aktuální podobu funkčních a nefunkčních požadavků.

3.1.2.1 Funkční

F1: Přehled veškerých konverzací uživatele Každý uživatel si může prohlédnout přehled svých dosavadních konverzací s jinými uživateli. Výčet bude seřazený podle data přijetí poslední zprávy v konverzaci.

F2: Zprávy v konverzaci s jiným uživatelem Aplikace zobrazí jednotlivé zprávy, které byly odeslány nebo přijaty v rámci jedné konverzace. Zprávy budou seřazeny podle času vytvoření, aby uživatel měl k dispozici chronologický detail celé konverzace.

F3: Odeslat zprávu uživateli Uživatel bude schopný odeslat zdánlivě libovolně dlouhou zprávu jinému uživateli v rámci jedné konverzace. Nebude však schopný odeslat jednu zprávu více příjemcům. Zpráva bude pouze textová, fotografie ani jiný typ souboru nebude pro odeslání podporován.

F4: Seznam nepřečtených konverzací Aplikace uživateli zřetelně oddělí nepřečtené konverzace od těch přečtených.

3.1.2.2 Nefunkční

N1: Spolehlivost Modul zobrazí vždy relevantní a aktuální data. Zaručí, že odeslaná data budou doručená příjemci.

N2: Uživatelsky příjemné prostředí Animace poskytované aplikací zprostředkují plynulý zážitek při prohlížení starších zpráv v konverzaci nebo obdržení a odeslání nové zprávy.

N3: Řešení pro případ absence internetu Ve chvíli, kdy uživatel nemá přístup k internetu, aplikace zobrazí dříve načtená data jak u konverzací, tak u zpráv konkrétní konverzace.

N4: Zabezpečená komunikace se serverem Chatovací modul bude se serverem komunikovat pouze po předchozí autentizaci. To se týká především odesílání nových zpráv z aplikace.

3.1.3 Uživatelské role

Uživatelské role jsou předem určené webovým portálem a v budoucnu budou sloužit k rozdílným způsobům třídění konverzací. Zatímco pracovník si bude schopný zobrazit všechny konverzace s danou firmou, aplikace zobrazí firmě přehled konverzací s pracovníkem nebo o konkrétní brigádě. Implementace této funkcionality je předmětem budoucí práce na projektu, nikoliv součástí bakalářské práce. Jedná se o následující role.

Uživatel role, která definuje entitu, která používá aplikaci;

Pracovník uživatel aplikace, který hledá brigádu;

Firma uživatel aplikace, který vytváří pracovní pozice v rámci brigády.

3.1.4 Návrh případů užití

Na základě uživatelských rolí a požadavků zadavatele jsem sestavil případy užití, které z pohledu uživatele popisují fungování aplikace. U složitějších funkcí bude rozebrán jak základní, tak i případný alternativní scénář.

U1: Zobrazit zprávy v konverzaci
Primární aktér: Uživatel
Popis: Po rozkliknutí konverzace, kterou si chce uživatel detailněji prohlédnout, se zobrazí seznam zpráv, seřazený podle data vytvoření sestupně, který je součástí zvolené konverzace. Po zobrazení zpráv, aplikace zkontroluje připojení k internetu. Pokud je výsledek kontroly negativní, notifikuje uživatele, že zobrazená data nejsou aktuální.

U2: Zobrazit přehled konverzací
Primární aktér: Uživatel
Popis: Pro uživatele je důležité vidět celkový přehled všech jeho konverzací, kde je zřetelně vidět, které položky již má přečtené a které má ještě přečíst. Po zobrazení konverzací, aplikace zkontroluje připojení k internetu. Pokud je výsledek kontroly negativní, notifikuje uživatele, že zobrazená data nejsou aktuální.

U3: Aktualizovat přehled konverzací
Primární aktér: Uživatel
Popis: Pokud má uživatel pocit, že aplikace nereaguje na příchozí zprávy a přehled není aktuální, může manuálně získat veškeré konverzace znovu ze serveru.

U4: Zobrazit konverzace s pracovníkem
Primární aktér: Firma
Popis: Firma má možnost si zobrazit veškeré konverzace s určitým brigádníkem.

U5: Odeslat zprávu
Primární aktér: Uživatel
Popis: Každý uživatel aplikace může poslat zprávu opačné uživatelské roli. Detailní scénář tohoto případu užití je zobrazen v tabulce 3.1. Alternativní scénaře se nacházejí v tabulkách 3.2 a 3.3.

Krok	Role	Akce
1	Uživatel	Otevře konverzaci, ve které chce vytvořit novou zprávu.
2	Uživatel	Napíše zprávu do kolonky k tomu určené.
3	Uživatel	Klikne na tlačítko pro odeslání.
4	Aplikace	Zkontroluje, zdali zpráva není prázdná.
5	Aplikace	Odešle zprávu na server.
6	Aplikace	Zobrazí uživateli jeho odeslanou zprávu.

Tabulka 3.1: Hlavní scénář odesílání zprávy

Krok	Role	Akce
3a1	Aplikace	Nezapne tlačítko pro odesílání zpráv.
3a2	Uživatel	Pokračuje 2. krokem hlavního scénáře 3.1.

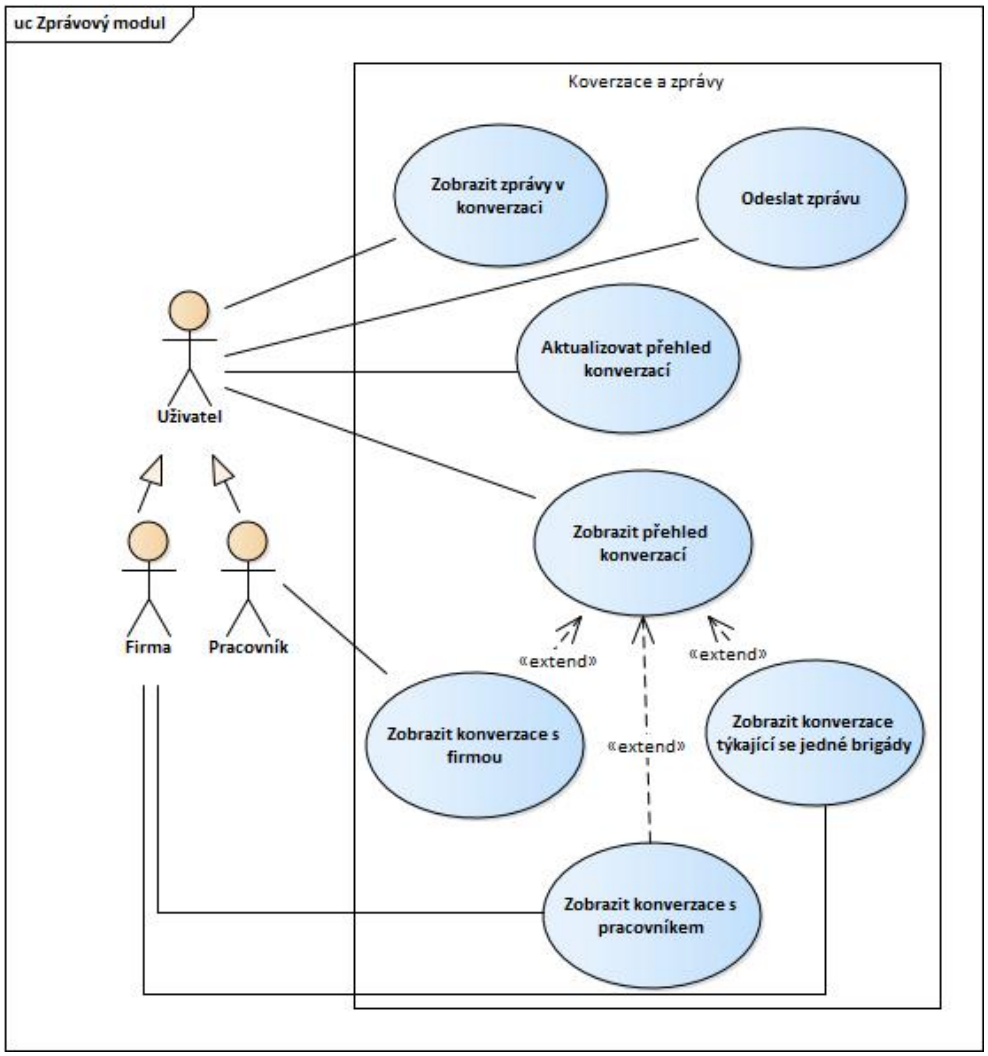
Tabulka 3.2: Alternativní scénář – zpráva je prázdná

Krok	Role	Akce
6a1	Aplikace	Notifikuje uživatele o chybě, která nastala při komunikaci se serverem.
6a2	Uživatel	Pokud uživatel chce zkusit odeslat zprávu znovu, pokračuje krokem 3 hlavního scénáře. 3.1.

Tabulka 3.3: Alternativní scénář – server vrátí chybovou hlášku

U6: Zobrazit konverzace s firmou
Primární aktér: Pracovník
Popis: Pracovník má možnost si zobrazit veškeré konverzace s určitou firmou.

U7: Zobrazit konverzace o brigádě
Primární aktér: Firma
Popis: Firma má možnost si zobrazit veškeré konverzace s pracovníky, týkající se jedné brigády.



Obrázek 3.1: Případy užití chatovacího modulu

3.2 Výběr technologií

V této podkapitole se zaměřím zejména na výběr vhodných technologií pro vývoj jak modulu pro pracovníky, tak chatovacího modulu. Tímto navazuji na kapitolu 2, kde jsem analyzoval možné přístupy k řešení jednotlivých problémů.

3.2.1 Tvorba dynamických datových listů

Ve všech případech, kde je zapotřebí zvolit datový list, jsem použil komponentu `RecyclerView`. Učinil jsem tak hlavně díky jeho pokročilé funkci recyklovat své vlastní položky, a tak ušetřit aplikaci část jeho výkonu. Komponenta `ListView` by vyhovovala pouze za předpokladu, že by seznam nebyl větší výšky, než je velikost mobilního zařízení. U všech případů užití je počet položek v kolekci dynamický, až na filtr, kde je položek neměnné množství, ale i tak se na výšku mobilního zařízení nevejde. V následující podkapitole se zaměřím na výběr adaptérů naplňujících `RecyclerView`.

3.2.2 PagedList.Adapter

Tento adaptér se hodí pro obsáhlé a často se měnící seznamy. Konkrétně se jedná o seznam konverzací uživatele a seznam brigád. Kvůli velkému množství odpovídajících dat na serveru, je nutné si o tato data žádat po částech.

3.2.3 RecyclerView.Adapter

Ve zbývajících případech tvorby datového listu vím, že množství položek nebude příliš velké nebo bude zapotřebí si stránkování upravit podle svého.

Seznam zpráv V případě seznamu zpráv v konverzaci se může zdát, že spíše splňuje předpoklady pro `PagedList.Adapter` (jedná se mnohdy o velké množství dat), ale není tomu tak. Při načítání starších zpráv by byl sice užitečný, ale pro kvalitní animaci při postupném přidávání jednotlivých zpráv, se nehodí. Jeho algoritmus, který porovnává příchozí kolekci je pro tento případ až zbytečně důkladný. Odeslané zprávy v konverzaci není možné měnit ani mazat, tudíž nemá smysl kontrolovat, zdali se náhodou nezměnily. Z tohoto důvodu jsem vytvořil vlastní algoritmus pro porovnávání příchozích zpráv a použil jsem čistý `RecyclerView.Adapter`.

Profil – kvalifikace Každý pracovník má své kvality a schopnosti, které ve svém profilu uvádí, aby si mohl vybrat z většího výběru brigád. Tyto schopnosti, ať už se jedná o jazyky, kterými uživatel mluví, o dřívější pracovní zkušenosti nebo schopnosti pracovat s počítačem, si pracovník edituje podle libosti. Je zapotřebí mu umožnit CRUD operace nad těmito seznamy. Pro tuto funkcionalitu se `RecyclerView.Adapter` velmi hodí, neboť umožňuje plynulé animace při editaci jeho zdrojových dat. Těchto dat není nikdy velké množství, a proto není třeba stránkování.

Filtr Pracovník má své preference ohledně zobrazovaných brigád. Určí si pozice a lokality, ve kterých chce pracovat. Podle připraveného designu bude pracovníkovi zobrazen seznam všech pozic a lokalit pod sebou, ze kterých si

bude moct vybrat jen ty, které ho zajímají a které upřednostňuje. Počet řádků v seznamu je fixní a pohybuje se okolo 10, tudíž stránkování je v tomto případě zbytečné, ale velikost seznamu nejspíš bude větší než velikost obrazovky.

3.2.4 Manipulace s fotografiemi

Po analýze jsem se rozhodl, že všechny knihovny zpracovávající fotografie můžu použít. Rozdíly mezi nimi nejsou nikterak velké. Přesto jsem si vybral Glide. Zejména mě oslovil svou možností zobrazit dočasný obrázek, pokud chtěná fotografie ještě není připravena. Také možnost automaticky upravit kvalitu a velikost obrázků, pokud je jejich počet v jednom pohledu příliš vysoký, je velmi užitečná například při zobrazení velkého množství konverzací, kde každá položka má obrázek reprezentující brigádu, o které se v konverzaci hovoří. Knihovna má také uživatelsky příjemné API.

3.2.5 Zobrazení informačních hlášek

Neaktuální data V aplikaci nastávají okamžiky, kdy uživatel nemá přístup k internetu. V tu chvíli se stávají zobrazená data neaktuálními, neboť uživatel si nemůže být jistý, že na serveru nedošlo ke změnám, které mohly zobrazená data ovlivnit. Z tohoto důvodu je uživateli zobrazen `SnackBar` s informací, že zobrazená data nejsou aktuální a kdy proběhla jejich poslední aktualizace. `SnackBar` jsem vybral především z toho důvodu, že si jeho přítomnosti uživatel všimne s vetší pravděpodobností než komponenty `Toast`.

Chyba při načítání dat ze serveru Neúspěšná aktualizace dat nebo hláška ze serveru jsou momenty, kdy přichází v užitek `Toast`. V tyto okamžiky chci upozornit uživatele, že událost proběhla buďto správně nebo s chybou. Informace se zobrazí na krátkou chvíli a neovlivní aktuální práci uživatele.

3.2.6 Tvorba dialogů

K přidávání odkazů na youtube video a k odeslání motivačního dopisu jsem použil `AlertDialog`. Dá se vytvořit snadno pomocí třídy `Dialog.builder` a rychle upravit pomocí XML souboru.

Kapitola 4

Návrh

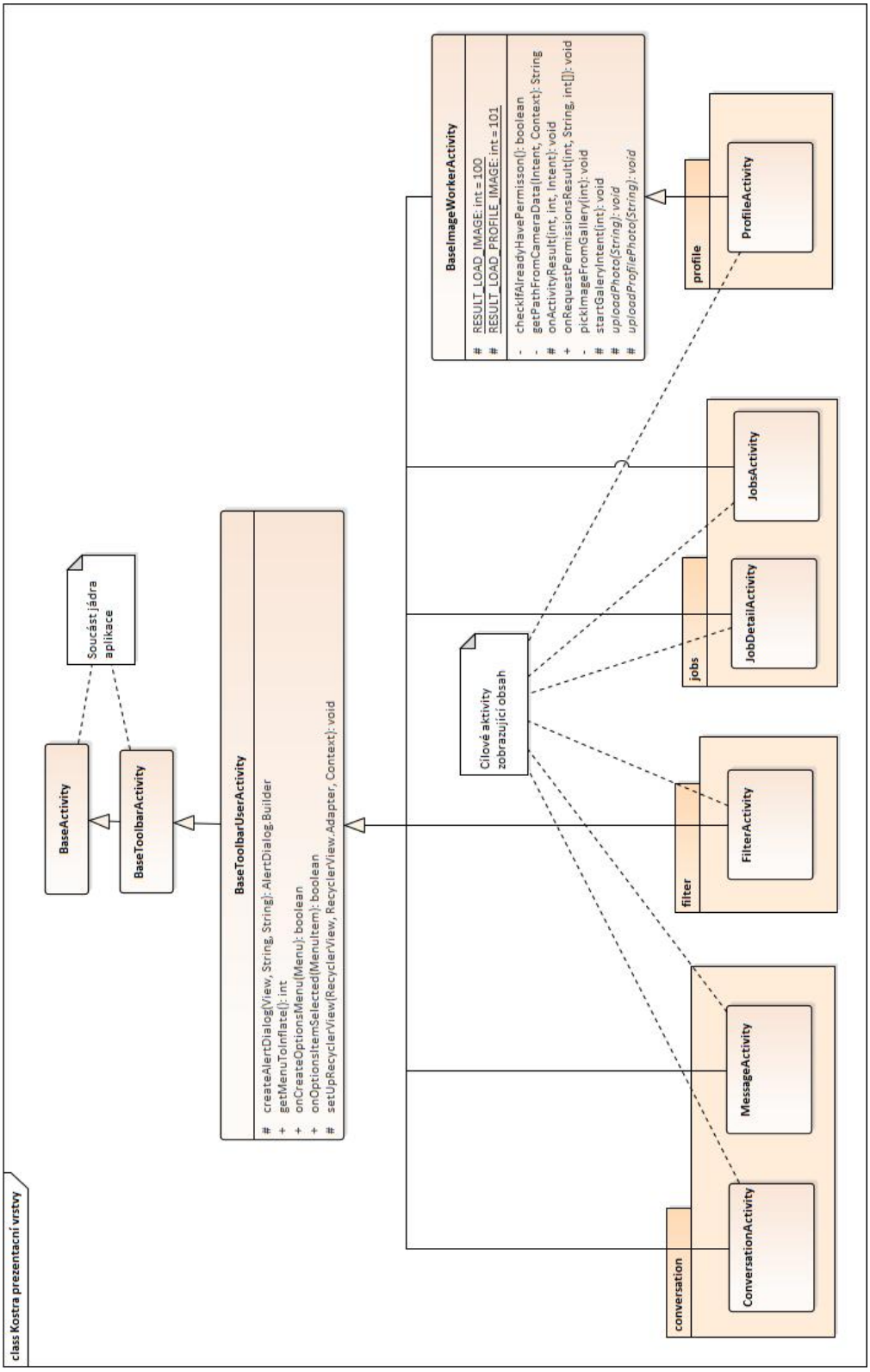
V rámci této kapitoly navrhnu moduly pro chatování a pracovníky. Nejdříve se zaměřím na řešení funkcionalit, které jsou společné pro oba moduly. Postupně se dostanu ke konkrétním problémům, které se pokusím co nejvíce atomizovat a navrhnout jejich řešení.

4.1 Kostra prezentační vrstvy

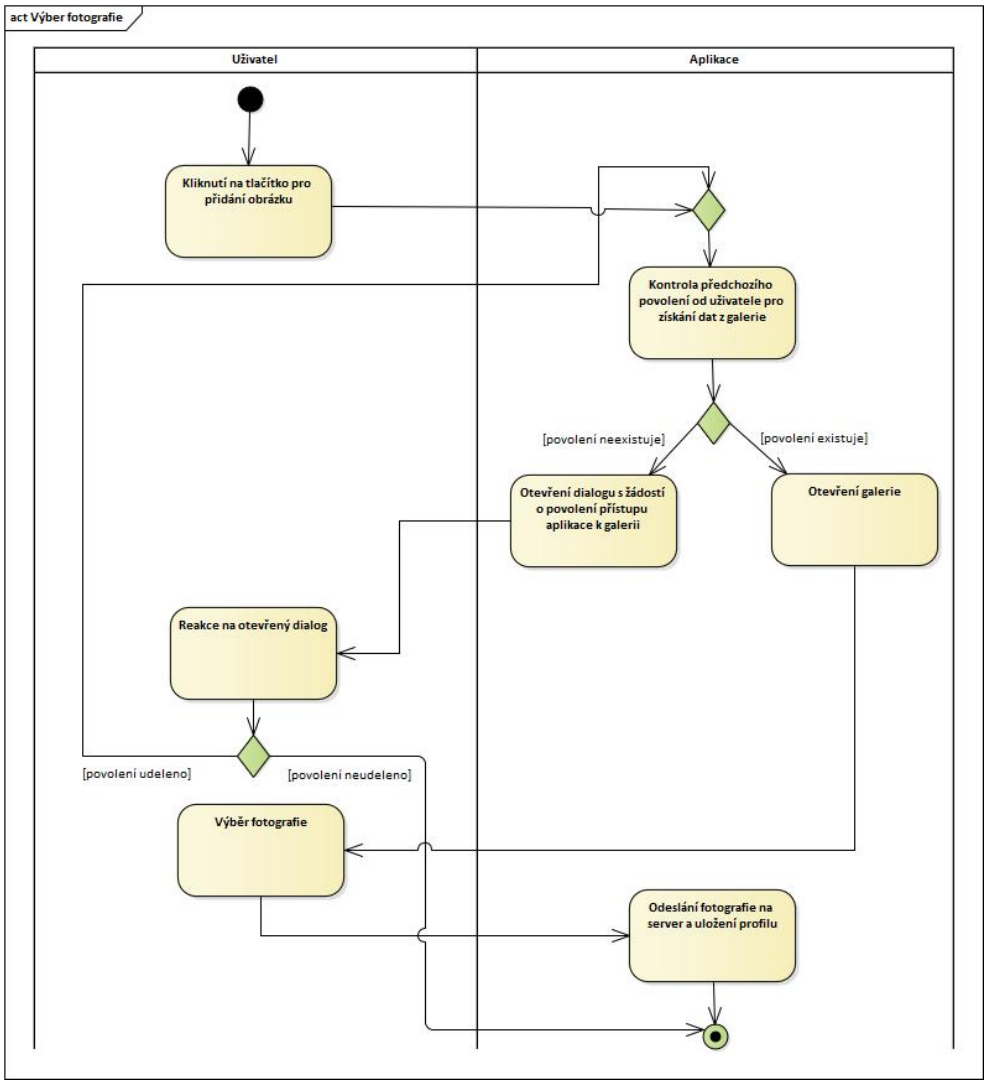
Pro účel tvorby prezentační vrstvy je třeba vytvořit několik abstraktních tříd, které budou sloužit k usnadnění vývoje aktivit, fragmentů a adaptérů napříč aplikací. U každé abstraktní třídy uvedu její hlavní funkce.

- **BaseToolbarUserActivity** — Tato abstraktní třída umožňuje jednoduchou manipulaci s komponentou `Toolbar`, která má v různých pohledech různou strukturu. Definuje, který soubor obsahující strukturu vzhledu `Toolbaru` má být použitý a jaké chování budou mít jeho položky. Třída současně obsahuje prostředky pro vytváření dialogů a usnadňuje nastavení komponenty `RecyclerView` při jejím vytváření.

- **BaseImageWorkerActivity** — Jedná se o potomka abstraktní třídy `BaseToolbarUserActivity`. Zděděné funkcionality rozšiřuje o celou logiku vybírání obrázků z galerie mobilního zařízení. Tento proces popisuje obrázek 4.2. Při požadavku o výběr obrázku z galerie, je zapotřebí ověřit, zdali má aplikace oprávnění. Aplikace otevře galerii, pokud tato oprávnění již dříve obdržela. V opačném případě vytvoří akceptační dialog pro získání těchto oprávnění od uživatele. Po vybrání fotografie si aplikace získá místo, kde se fotografie nachází a tuto cestu předá dalším třídám, které odešlou obrázek na server, kde se mají obrázky uložit.



Obrázek 4.1: Diagram tříd – prezentační vrstva



Obrázek 4.2: Diagram aktivit – výběr obrázku z galerie

- **BaseFragment** — Obdoba `BaseActivity` pro fragmenty. Stará se o poskytování viewmodelů svým potomkům.
- **BaseRecyclerViewAdapter** — Na rozdíl od výše zmíněných abstraktních tříd, se `BaseRecyclerViewAdapter` nestará o zobrazování dat, ale o jejich dodávání. Všechny adaptéry naplňující `RecyclerView` nestránkovanými daty, by měly tento abstrakt dědit. Implementuje metody, které `RecyclerView.Adapter` má za abstraktní a dále po svých potomcích požaduje pouze vzhled položky, která má byt vložena do listu a jakým způsobem má být položka naplněna. Dále umožňuje vkládání a odebírání

elementů po jednotkách.

4.2 Načítání dat

Aplikace se bez dat získaných z lokální databáze nebo ze serveru neobejde. Jakmile je zapotřebí zobrazit data uživateli, repozitář požádá connector, třídu odpovědnou za spouštění a organizaci pracovních vláken, o vytvoření vlákna, které tato data získá ze serveru. Vlákno získaná data uloží do lokální databáze. V případě chyby, která se může objevit kvůli špatnému internetovému připojení nebo chybě na serveru, je uživatel o této události notifikován.

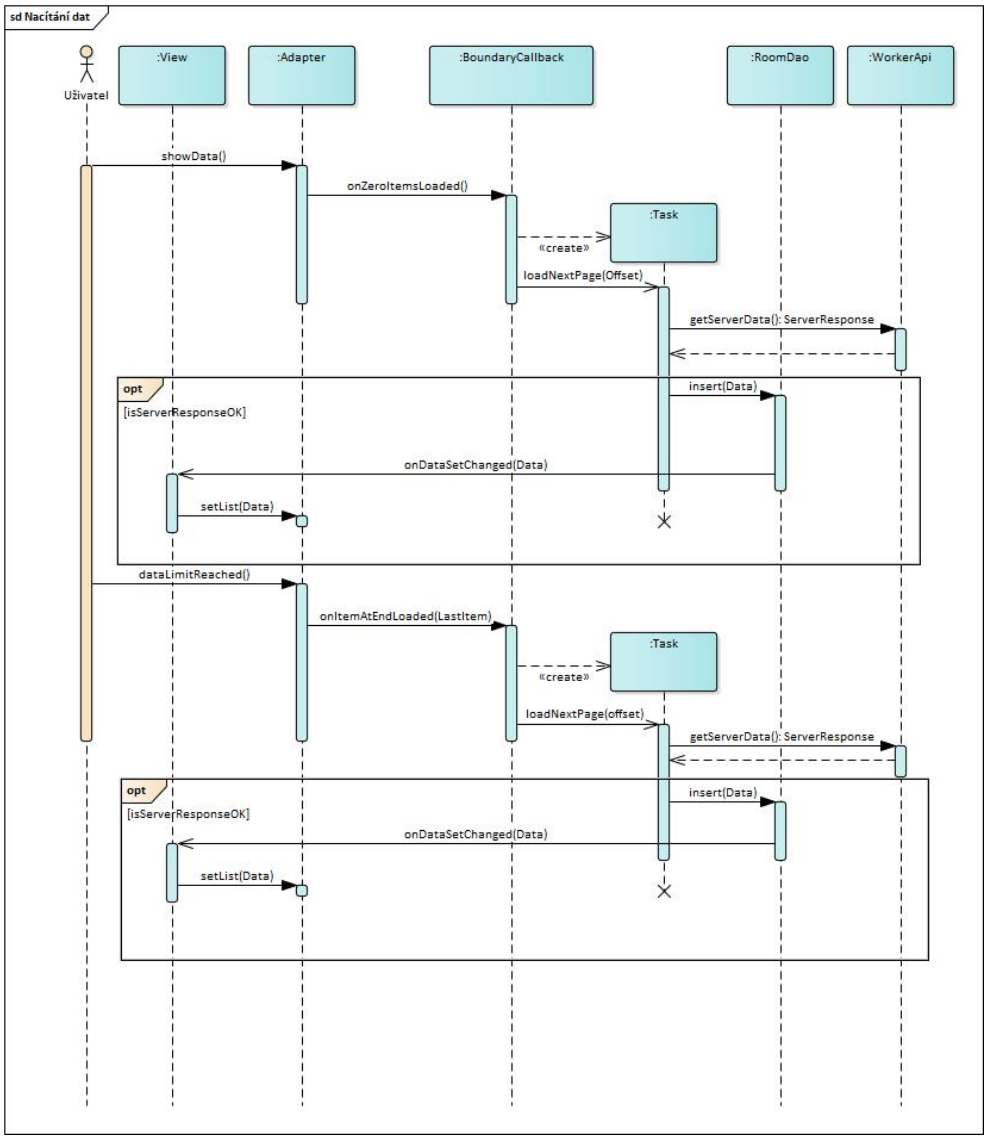
Při inicializaci si repozitář získá referenci na data v databázi zabalená do třídy `LiveData` a dále je prostřednictvím viewmodelu poskytuje prezentační vrstvě. Aktivita nebo fragment sledují tato data a na jejich změny reagují přímo úpravou obsahu UI komponent, které jsou viditelné uživateli. Tak je zaručeno, že uživatel má zobrazena vždy aktuální data.

4.2.1 Po stránkách

Výše uvedený postup platí pouze pokud data ze serveru získáváme v celku. Pokud je dat velké množství, vyplatí se o ně žádat po částech. Stránkovanými daty budu vždy naplňovat `RecyclerView` s využitím třídy `PagedList.Adapter` jako jeho adaptéru. Komunikace mezi prezentační vrstvou a viewmodelem zůstává stejná. Zásadně se změní podoba repozitáře, který musí být modifikovaný tak, aby umožnil získávání dat po stránkách. Celý proces je součástí obrázku 4.3.

Přibyly třídy `BoundaryCallback` a `Pagedlist`. `Pagedlist` je kolekce, která podporuje stránkování. `BoundaryCallback` se stará o načítání dat ze serveru. Pokud je počet požadovaných dat v lokální databázi nulový, zašle třída požadavek na server, ve kterém specifikuje, jaké množství dat chce vrátit. Pokud je počet nenulový a uživatel chce zobrazit více dat, než je aktuálně v lokální databázi, zašle třída požadavek na server, ve kterém specifikuje, jaké množství dat chce vrátit a jedinečný identifikátor poslední položky, která se v lokální databázi nachází. Tím informuje server, od které položky je zapotřebí data vracet.

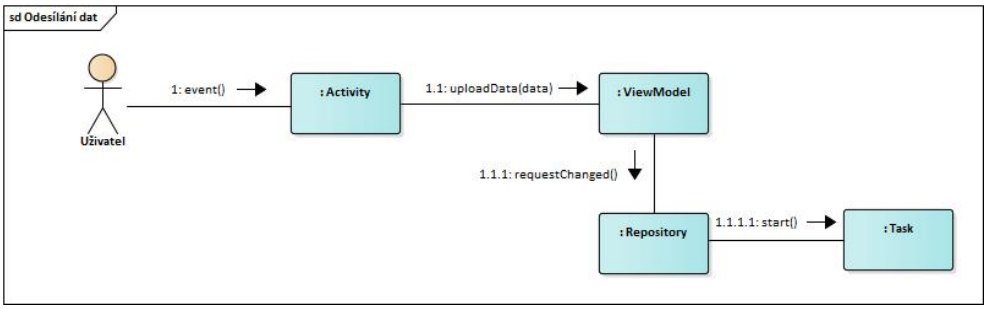
Jakmile adaptér dostane nová data, porovná je s daty, které má již uložená. Porovnání probíhá pomocí generické třídy `DiffCallback<T>`. `DiffCallback<T>` je třída, která obsahuje dvě metody. Jedna rozhoduje, jestli dvě položky jsou stejné a druhá, jestli mají dvě položky stejný obsah. Nová a upravená data dále adaptér prezentuje uživateli plynulou změnou položek v komponentě `RecyclerView`.



Obrázek 4.3: Sekvenční diagram – načítání stránkovaných dat

4.3 Odesílání dat na server

Při odesílání dat na server je struktura tříd stejná jako při načítání celistvých dat s tím rozdílem, že proces je jednosměrný. Jakmile aplikace odešle data, soustředí se na to, zdali server vrátí chybu nebo zprávu o úspěšném uložení dat. Tento stav se prezentuje uživateli pomocí komponenty `Toast`. Po úspěšném uložení dat na serveru se data uloží do lokální databáze. Průběh odesílání dat znázorňuje diagram 4.4.



Obrázek 4.4: Kolaborační diagram – odesílání dat na server

4.4 Zpracování push zprávy

O změnách dat na serveru je nutné informovat také mobilní aplikaci. O rozdělení těchto zpráv jsem se zmínil v rámci rešeršní kapitoly 2. Tiché a hlasité zprávy jsem rozšířil o pojmy invalidační a notifikační zpráva.

Invalidační zpráva Pokud na serveru proběhne operace typu, uživatel projevil zájem o novou brigádu apod., je výhodnější informovat aplikaci, že seznam brigád, o které má uživatel zájem již není aktuální, nežli mu poslat zprávu s informací, o kterou brigádu projevil uživatel zájem. Hlavní výhodou je zejména to, že pořadí dat v lokální databázi zůstane totožné s tím, v jakém jsou data získána ze serveru. Jakmile aplikace obdrží zprávu o tom, že některá data již nejsou aktuální, najde příslušný handler repozitář, který je odpovědný za dodání těchto dat a oznámí mu, že je potřeba o tato data server znovu požádat. Abstraktní třída `BaseInvalidateRepository` definuje funkcionalitu pro repozitáře, jejichž data se mohou stále invalidními.

Notifikační zpráva V případě, že se na serveru stala událost, o které je důležité notifikovat přímo uživatele, obdrží aplikace hlasitou zprávu. Tato zpráva obsahuje data, která je třeba prezentovat uživateli. Příkladem může být přijetí žádosti uživatele o účast na brigádě. V tomto případě se jedná současně i o invalidační zprávu, neboť aplikace musí repozitáře s brigádami, o které má uživatel zájem a které jsou nepotvrzené, informovat, že jejich data již nejsou validní.

4.5 Offline data

Pro splnění požadavku zobrazovat uživateli data i v situaci, že není připojený k internetu je použito získávání dat z lokální databáze. Takto uživatel, i když nezíská aktuální data ze serveru, pořád má ty, které od něj již jednou obdr-

žel. Důležité je uživateli oznámit, že data, která jsou mu zobrazena, již nejsou aktuální. `TimeObject` je abstraktní třída, která si drží čas, kdy byla vytvořena. Kterákoliv entitní třída, která je součástí aplikace a může být zobrazená uživateli i když nemá připojení k internetu dědí od `TimeObject`.

Jakmile pohled získá data, třídní metoda `handleDataUpToDate(Context, Timestamp)`, která je součástí třídy `DateState`, zavolá metodu zjišťující stav připojení k internetu. V případě, že internetové připojení není dostupné, metoda `onGetDataError(Context, Timestamp)` zobrazí `SnackBar` s informací, že zobrazená data nejsou aktuální, včetně času, kdy byly data naposledy aktualizována.

4.6 Filtr

Pracovní pozice i lokality jsou dvě různé třídy se stejnou strukturou. Struktura se skládá z id, názvu filtru a zdali je filtr uživatelem označený nebo ne. V aplikaci budou obě třídy součástí jedné databázové tabulky. Odlišovat je bude proměnná typu enum, podle které bude tabulka indexována. Indexace umožní rychlejší vyhledávání v tabulce při získávání dat. Repozitáře získávající data z databáze jsou totožné. Jediné, co je od sebe odlišuje je výše zmíněné enum. Z tohoto důvodu existuje abstraktní třída `FilterGetRepository`, která bude obsahovat sdílené metody pro oba druhy filtrů. Pro získání chtěného filtru bude stačit repozitář zdědit od `FilterGetRepository` a definovat, který typ filtru chceme z lokální databáze získat.

Dalším milníkem při návrhu filtrů, je vytvoření dvou separátních fragmentů, kde jeden bude zobrazovat pracovní pozice a druhý lokality. Mezi fragmenty může uživatel přepínat v horním dvou položkovém menu nebo pomocí horizontálního listování. Komponenta umožňující horizontální listování se nazývá `ViewPager`. Aby tato komponenta správně fungovala, je zapotřebí ji naplnit pomocí adaptéru zděděného od `FragmentPagerAdapter`.

Výsledné fragmenty budou filtry zobrazovat pomocí `RecyclerView`, kde jedna položka bude obsahovat `CheckBox`, identifikující, jestli je položka uživatelem vybraná a `TextView`, které položku textově definuje. Jako bonus může uživatel považovat sdružující položky, které nejsou součástí odpovědi ze serveru a které vypadají jako kterékoliv jiné položky, ale mají rozdílné vlastnosti. Jedna sdružující položka je navázána na více filtrových položek. Například v lokalitách je takovou sdružující položkou Centrum Prahy. Pokud ji uživatel zvolí, jsou automaticky zvoleny filtrové položky Praha 1, Praha 2 a Praha 3.

4.7 Seznam brigád

Situace se zde velmi podobá filtru. Cílem je zobrazit 3 fragmenty, které zobrazí výčet brigád v různých stavech. Brigády mají vždy stejnou strukturu. Obsahují veřejný klíč brigády, cestu k obrázku, kdy byla brigáda vytvořena, číslo její aktuální verze, kolik je hodinová sazba a kdy brigáda končí. Nabízí se tedy řešení, napodobit postup u filtru. Problémem je dotahování dat ze serveru u třídy `Pagedlist.Adapter`. Stránkování pomocí komponent `Pagedlist` třídy reaguje na jakékoliv změny dat v lokální databázi a zahájí kontrolní dotaz na server, zdali nejsou dostupná další data. To zapříčiní zbytečné serverové volání u dat, které aktuálně uživatel třeba ani nevidí. Proto má každý stav brigády svou vlastní tabulku v databázi. Těchto stavů je 5:

1. filtered,
2. pending,
3. unaccepted,
4. accepted,
5. favourite.

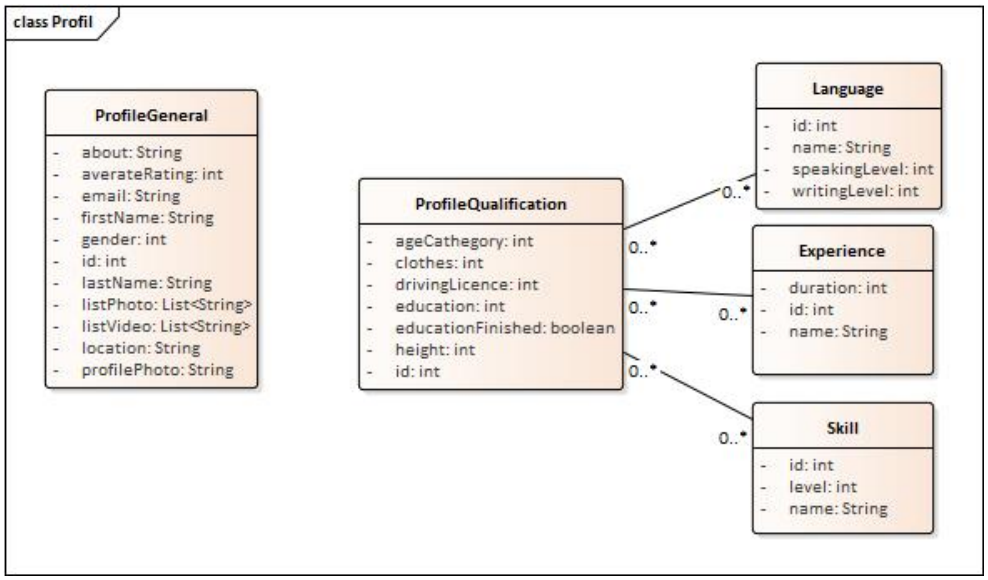
Jejich repozitáře a třídy `BoundaryCallback` jsou téměř totožné. Čím se liší jsou tabulky, ze kterých získávají brigády a o které brigády si na serveru žádají. `BaseJobRepository` je abstraktní třída, která obsahuje sdílené metody napříč repozitáři s brigádami. Inicializuje si `JobBoundaryCallback`, kterému předá, o jaký stav brigády se jedná. Tento stav třída obdrží ve svém konstruktoru. Pro přidání nového stavu stačí určit tabulku, ze které se mají čerpat brigády a stav brigády, který je zapotřebí ze serveru získat.

Po získání brigád si společný adaptér `JobsAdapter` přijatá data porovnává podle jejich jedinečného identifikátoru a obsah pomocí proměnné `version`. Pokud se ze serveru vrátí upravená brigáda, má vždy jiné číslo verze, než tomu bylo předtím.

Přechody mezi fragmenty obsahující výčty brigád v různých stavech jsou součástí komponenty `FrameLayout`. Spodní menu, umožňující přechody mezi fragmenty vyřeší komponenta `BottomNavigationView`.

4.8 Profil

Profil se skládá z několika navzájem propojených tříd 4.5.



Obrázek 4.5: Diagram tříd – profil

Mezi hlavní funkce aktivity patří:

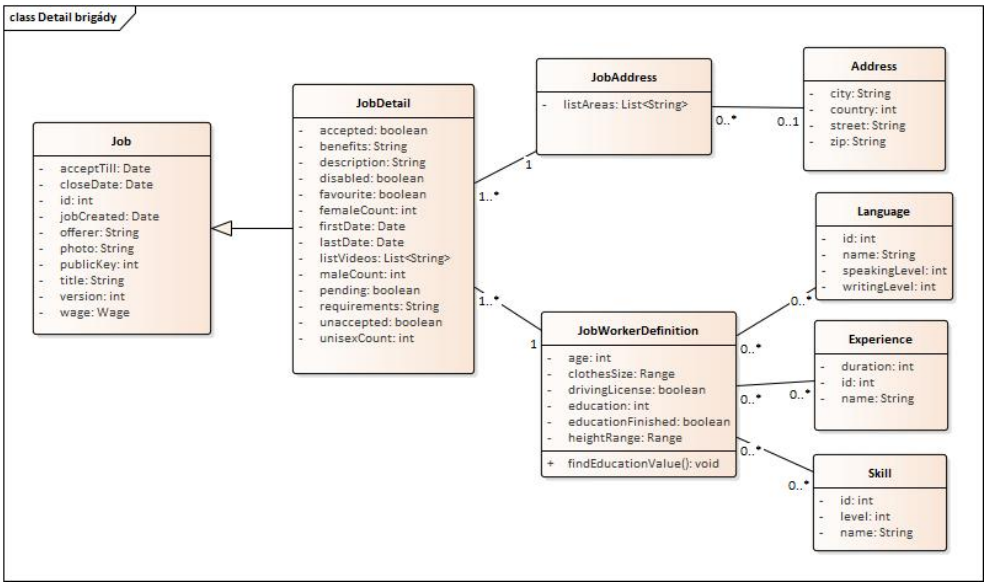
- **Nahrávání obrázku** – Aktivita zdědí tuto vlastnost od abstraktní aktivity `BaseImageWorkerActivity`, která se již o nahrávání obrázku postará sama. Stačí nastavit událost, která nahrávání obrázku spustí a viewmodel, který obrázek uloží na server.
- **Upravování dat a jejich validace** – Na základě připraveného designu je zapotřebí vytvořit komponentu, která bude vypadat jako obyčejný text, ale po kliknutí na ní bude uživatel schopný text upravit. Využiji možnosti nastylovat UI komponentu `EditText`, aby splnila výše uvedená kritéria. Po úpravě se vyplněný text zkontroluje a v případě, že nová data splňují kritéria, která jsou od nich požadována, tak se část profilu obsahující změněná data odešle a uloží na server.
- **Výběr z enumerace** – Kromě textu si uživatel může v případech jako věk, konfekční velikost apod. zvolit z nabídky předpřipravených možností, takzvaných enumerací, které jsou načteny do aplikace při přihlášení uživatele. Za tímto účelem použiji komponentu `Spinner`, která vypadá podobně jako `EditText`, avšak po rozkliknutí nabízí dialog s výčtem možností, mezi kterými si může uživatel jednu vybrat. Poté se zvolená položka objeví v textovém poli komponenty. Pro naplnění komponenty daty potřebuji adaptér `EnumerationSpinnerAdapter`, který umožní skrýt

některé hodnoty v enumeraci, vyhledat pozice položky v komponentě a získat její id.

- **CRUD operace u schopností** – Jazyky, IT dovednosti a pracovní zkušenosti uživatele tvoří tři separátní komponenty `RecyclerView`. Každá má svůj adaptér zděděný od `BaseRecyclerView`, který umožní přidávat, odebírat a upravovat schopnosti v komponentě. Položku schopností lze rozdělit na dvě části. V první vybírá uživatel z předvyplněných položek (pracovní pozice, jazyk, IT schopnost) a ve druhé ji hodnotí. Konkrétně udává úroveň jazyka, IT schopnosti nebo dobu setrvání na pozici, kterou v minulosti vykonával.

4.9 Detail brigády

Detail brigády lze považovat za neupravitelnou aktivitu Profil. Obě tyto aktivity mají společný vzhled a částečně i strukturu dat. Z tohoto důvodu použiji obdobné techniky při tvorbě detailu brigády.



Obrázek 4.6: Diagram tříd – detail brigády

Hlavním rozdílem mezi aktivitami je sada tlačítek, které každé vyvolá odlišnou událost. Tlačítka lze rozdělit do dvou kategorií. První kategorie obsahuje ta, která jsou součástí procesu zvolení brigády a druhá obsahuje tlačítka, která slouží uživateli pro získávání informací o brigádě. Událost vyvolaná tlačítkem první kategorie změní stav brigády. Po této změně, repozitáře se seznamy brigád, jejichž obsah byl změnou ovlivněn, musí invalidovat, aby aplikace získala

aktuální data ze serveru. Invalidování těchto repozitářů platí i v případě, že uživatel vykoná událost nad brigádou prostřednictví webového portálu. V tabulce 4.1 uvedu tlačítka první kategorie včetně repozitářů, jejichž data se stanou po události vyvolané na tlačítku neaktuálními.

Akce	Repozitář k invalidování
Mám zájem	pending
Nemám zájem	pending
Potvrdit	unaccepted, accepted
Odmítnout	unaccepted
Přidat do oblíbených	favourite
Odebrat z oblíbených	favourite

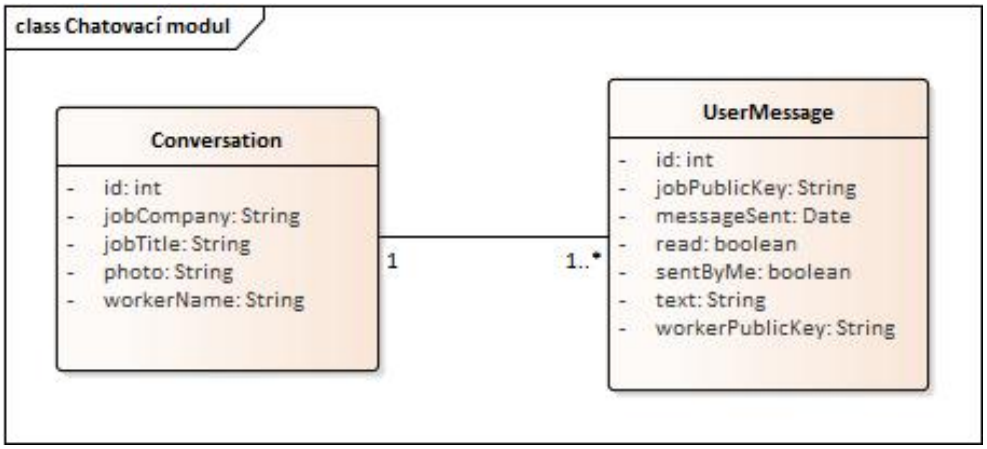
Tabulka 4.1: Detail brigády – tlačítka 1. kategorie

Tlačítka druhé kategorie jsou:

- **Mám dotaz** – Otevře konverzaci nad brigádou.
- **Zobrazit na mapě** – Zobrazí adresu brigády ve vestavěných mapách Androidu.

4.10 Chatovací modul

Na základě analýzy jsem sestavil podobu entitních tříd, které budou prezentovány uživateli 4.7.



Obrázek 4.7: Diagram tříd – chatovací modul

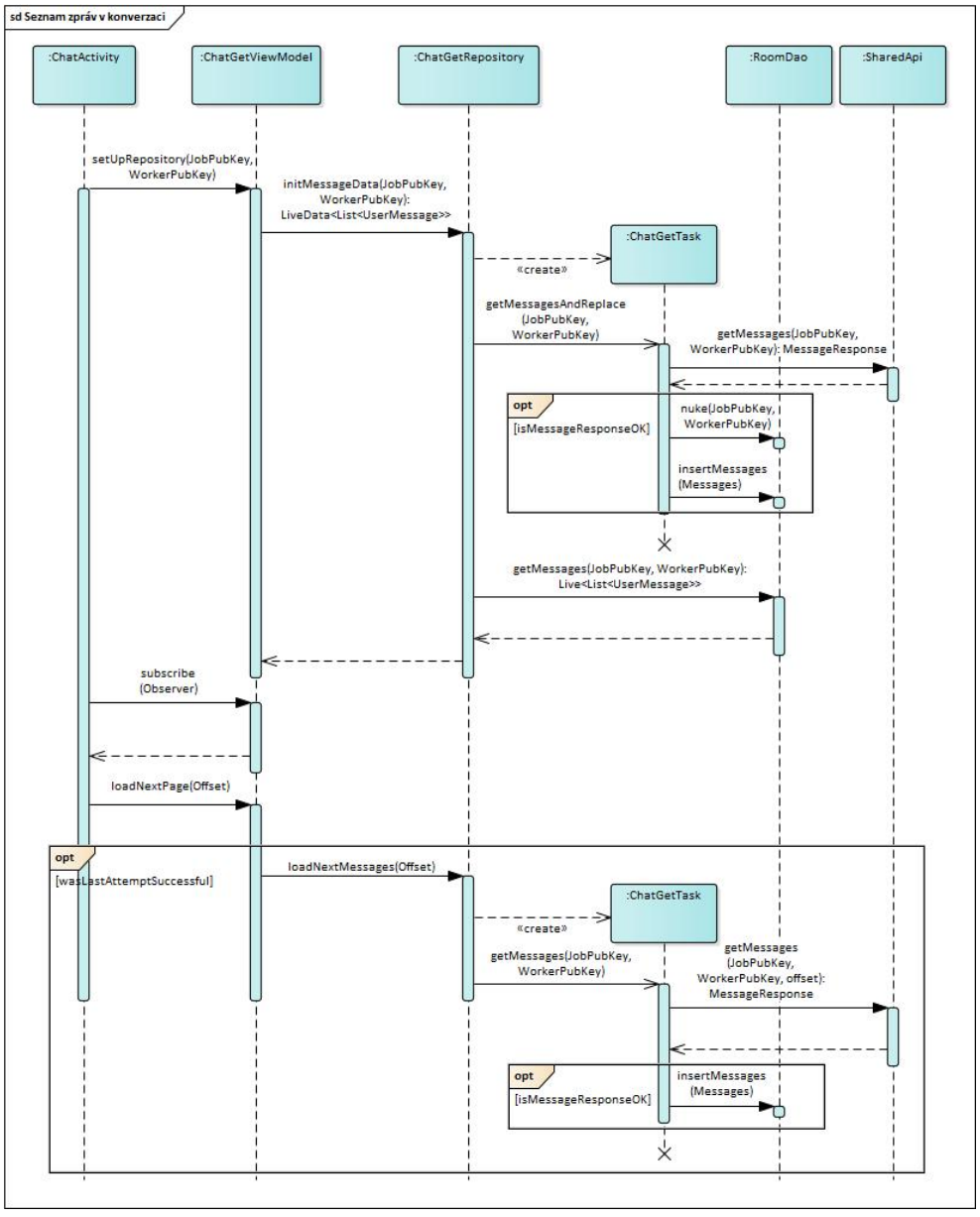
V této podkapitole se soustředím na návrh řešení případů užití chatovacího modulu. U zobrazení kolekcí uvedu způsob načítání dat a specifikuji parametry procesu, které se budou od ostatních lišit.

Seznam konverzací Každý uživatel může mít velké množství konverzací. Z toho důvodu se vyplatí získávat ze serveru stránkovaná data a následně je prezentovat uživateli pomocí třídy zděděné od `PagedList.Adapter`. Pro porovnání dvou konverzací se použije její primární klíč id a pro porovnání obsahu se použije text poslední zprávy v konverzaci.

Seznam zpráv v konverzaci Zpráv v konverzaci bude z největší pravděpodobností mnohem více nežli konverzací samotných. Stránkovaná data budou tedy nezbytná. Pro naprostou kontrolu nad stránkováním, animacemi a přidáváním nových zpráv použiji třídu, zděděnou od `BaseRecyclerViewAdapter`, která mi poskytne snadné přidávání nových položek. Činím tak hlavně z důvodu, že `PagedList.Adapter` neumožňuje přidávat položky po jednotkách a tím zbytečně zatěžuje výkon mobilního zařízení. Jelikož odeslané zprávy v konverzaci nejde měnit ani mazat, nemá smysl při každé přijaté zprávě porovnávat celé kolekce. Je zapotřebí porovnat pouze položky, které se mohly změnit. Nepoužitím třídy `PagedList.Adapter` ovšem ztrácím její vlastnost získávat data po stránkách, a proto si musím tuto funkcionalitu vytvořit sám. Proces načítání dat je znázorněn na obrázku 4.8.

Odeslání zprávy Odesílání zpráv je postaveno na principu charakterizovaném v podkapitole desílání dat na server 4.3. Výjimku tvoří odpověď ze serveru, která obsahuje nově vytvořenou zprávu včetně jejího id, které je generováno na serveru a není jinak v aplikaci dostupné. Tato získaná zpráva se následně uloží do lokální databáze. Aktivita chatu je notifikována o nově přidané zprávě.

Přijetí zprávy Abych mohl zobrazit aktuální zprávy v konverzaci na všech zařízeních, musím brát v úvahu situace, ve kterých lze zprávu vytvořit. Odesílatel může používat webový portál nebo mobilní zařízení. Vytvořil jsem tedy strukturu, které nezáleží na tom, odkud byla zpráva odeslána. Jediné, co potřebuje vědět je, kdo je příjemce a kdo odesílatel zprávy. Po úspěšném uložení, portál odešle notifikaci do aplikace příjemce o obdržení nové zprávy. Notifikace bude ve svém těle obsahovat taktéž všechna data o příchozí zprávě a aplikace tedy bude schopná ji přidat do lokální databáze. Současně je repozitář s konverzacemi notifikován, že jeho data již nejsou aktuální, neboť výčet konverzací má nyní jinou podobu, než tomu bylo doposud. Zcela určitě se změní poslední přijatá zpráva a možná také pořadí konverzací. Stejně tomu je i u odesílatele. Ten obdrží totožná data ze serveru, pouze nezobrazí uživateli notifikaci.



Obrázek 4.8: Sekvenční diagram – načítání zpráv

Kapitola 5

Implementace

V této kapitole přiblížím realizaci návrhu aplikace a problémy, se kterými jsem se při implementaci potýkal.

5.1 Zobrazování obrázků

Knihovna Glide poskytuje přehledné rozhraní, kvůli kterému lze zobecnit způsoby zobrazení obrázků v aplikaci. Prvním způsobem je tradiční zobrazení obrázku a druhým je zobrazení obrázku transformovaného do kruhovitého tvaru a zmenšeného na pětinu své původní velikosti. Veškeré obrázky se cashují, aby se neopakovalo opětovné stahování obrázků z internetu. Třída `DisplayImage` poskytuje dvě statické metody `displayCircleImage(String, ImageView)` 5.1 a `displayImage(String, ImageView)`, které tyto funkce poskytují. Vše, co potřebují k zobrazení obrázku, je jeho url adresa, kde se nachází a UI komponenta `ImageView`, která fotografii zobrazí.

```
Glide.with(App.getInstance())
    .load(url)
    .thumbnail(0.2f)
    .apply(new RequestOptions()
        .diskCacheStrategy(DiskCacheStrategy.ALL)
        .placeholder(R.drawable.person_placeholder))
    .apply(RequestOptions.circleCropTransform())
    .into(imageView);
```

Kód 5.1: Metoda zobrazující fotografii

5.2 Chatovací modul

Hlavní implementační překážkou při tvorbě chatovacího modulu je vytvoření automatického stránkování zpráv v konverzaci a algoritmu, který dokáže podle požadavků porovnávat dvě kolekce zpráv.

5.2.1 Stránkování zpráv

V okamžiku, kdy uživatel začne vertikálně listovat v `RecyclerView`, aplikace rozhodne, zdali je potřebné získat další data ze serveru.

Komponenta `RecyclerView` si drží informaci o poslední viditelné zprávě a o celkovém množství zpráv v adaptéru. Jakmile se začnou tyto proměnné rovnat, uživatel narazil na konec dostupných dat. V tuto chvíli si aplikace řekne serveru o zprávy, které se nacházejí za poslední, která byla zobrazena uživateli.

5.2.2 Porovnávání zpráv

Zprávy přicházející do adaptéru se musí porovnat s těmi, které jsou v něm již obsaženy. Zprávy jsou uchovávány v `LinkedList`, aby se jejich pořadí zaručeně nemohlo měnit a aby bylo snadné přistupovat k prvnímu a poslednímu prvku kolekce. Porovnání musí brát v úvahu následující vlastnosti zpráv, aby byl algoritmus co nejvíce efektivní:

- nové zprávy mohou být pouze na konci nebo na začátku,
- zprávu nelze smazat,
- zprávu nelze upravit.

Algoritmus rozdělím na tři disjunktní situace podle vztahu velikostí příchozí kolekce a původní kolekce zpráv.

1. **Příchozí kolekce se rovná původní** — Obě kolekce jsou shodné, neboť zprávy nelze upravovat.
2. **Příchozí kolekce je větší než původní** — Porovnám první položky v kolekcích, pokud jsou stejné, nové zprávy začínají na n-té pozici v příchozí kolekci, kde n je počet zpráv v původní kolekci 5.2.

   Pokud se první položky nerovnají, algoritmus najde pozici zprávy v příchozí kolekci, která se rovná první zprávě v původní kolekci. Pokud je rozdíl velikosti přicházející kolekce a pozice větší nežli velikost původní kolekce, je zřejmé, že jsou nové zprávy na konci přicházející kolekce,

které je nutné přidat. Na závěr se přidají zprávy ze začátku přicházející kolekce dokud algoritmus nenarazí na zprávu, která je již v původní kolekci obsažená 5.3.

```
if (messagesAreSame(incomingCollection.get(0),
                    originalMessages.get(0))) {
    for (int i = originalMessages.size();
             i < incomingCollection.size();
             i++) {
        addLast(incomingCollection.get(i));
    }
}
```

Kód 5.2: Algoritmus na porovnávání zpráv – nové zprávy na začátku přicházející kolekce

```
int pos = 1;
for (; pos < originalMessages.size(); pos++) {
    if(messagesAreSame(incomingCollection.get(pos),
                       originalMessages.get(0))){
        break;
    }
}
if(incomingCollection.size()-pos> originalMessages.size()){
    for(int i = originalMessages.size();
            i< incomingCollection.size();
            i++)
        addLast(incomingCollection.get(i));
    }
    pos--;
    for(; pos>=0; pos--){
        addFirst(incomingCollection.get(pos));
    }
}
```

Kód 5.3: Algoritmus na porovnávání zpráv – nové zprávy na začátku i konci přicházející kolekce

3. **Příchozí kolekce je menší než původní** — Situace, která nemůže nastat, neboť zprávu nelze smazat.

5.3 Sdružující filtry

Pojem sdružující filtr jsem představil v podkapitole Filtr 4.6. Jedná se o položku ve výčtu, která představuje souhrn více filtrů. Například univerzální

položka Vše. Jak už název napovídá, pokud je zvolená, musí být zvolené i všechny filtry ve výčtu zaškrtnuté a naopak. U lokačních filtrů je sdružujících filtrů vetší množství. Pro přehlednost představím řešení jednoho z nich. Konkrétně zpracuji Centrum Prahy, které se skládá z Prahy 1, Prahy 2 a Prahy 3. Uvedu metody, které jsou součástí `FilterAreaAdapter` a jsou použity pro implementaci výše nastíněného problému.

- `void setupCheckBoxes()` – Všechny sdružující filtry zjistí, zdali mají být zaškrtnuté zavoláním metody `isCenterChecked()` a na základě výsledku tak učiní.
- `boolean isCenterChecked()` – Vrátí informaci o tom, jestli jsou filtry Praha 1, Praha 2 a Praha 3 zaškrtnuté.
- `void checkCenter(boolean isCheckedCenter)` – Vytvoří nebo zruší zaškrtnutí na základě parametru v metodě filtrům Praha 1, Praha 2 a Praha 3.

Během vytváření výčtu filtrů pro komponentu `RecyclerView` je volána třídní metoda `setupCheckBoxes()`. Pokud pracovník zvolí kterýkoliv z filtrů mimo Centrum Prahy, zavolá se `setupCheckBoxes()`. Pokud zvolí Centrum Prahy, zavolá se metoda `checkCenter(boolean isCheckedCenter)` a následně se zkontroluje, jestli jiné sdružující filtry nebyly pozměněny `setupCheckBoxes()`.

Zbytek sdružujících filtrů je vyřešen stejným způsobem, pouze s jinými parametry hledání položek ve výčtu filtrů.

5.4 Adaptér enumerací

`EnumerationSpinnerAdapter` má na rozdíl od jiných adaptérů více funkcí. Nenaplňuje pouze komponentu `Spinner` daty, ale umožňuje blokaci položek ve výčtu, zvolení enumerace atd.

Použití Výčet prvků požadované enumerace se naplní do adaptéru metodou `setEnumeration(List<Enumeration>)`. Na základě získaných dat aplikace určí komponentě `Spinner`, kterou enumeraci zvolit. K tomu má k dispozici id enumerace a metodu `getPositionDwId(int)`, která vrací pozici, na které se nachází enumerace s daným id.

V případě, že uživatel zvolí ve `Spinneru` enumeraci, která má být uložena na server, musí adaptér vrátit id enumerace na zvolené pozici. K tomu slouží metoda `getItemDwId(int position)`.

Blokování enumerace Při výběru kvalifikačních schopností si pracovník nemůže zvolit dvakrát tu stejnou schopnost. Z tohoto důvodu si adaptér uchovává množinu enumerací `mHiddenEnumerations`, které jsou již zvolené. Jakmile uživatel vybere enumeraci, její id je přidáno do této množiny. Při tvorbě prezentovaného výčtu si adaptér u každé enumerace ověří, zdali už není jednou zvolená a pokud je, tak ji zakáže. Toho jsem docílil prostřednictví přepsání metody `isEnabled(position)` 5.4.

```
@Override
public boolean isEnabled(int position) {
    return !mHiddenEnumerations.contains(position);
}
```

Kód 5.4: Zjištění stavu položky v enumeraci

5.5 Validace

Validace editovatelného textu jsou prováděny samostatnou třídou `Validate`. Třída obsahuje statické metody, které vracejí informaci o tom, zdali je text v pořádku pro odeslání a uložení. Nejvíce je používána metoda pro zjištění prázdnosti vyplňovaného pole 5.5.

```
boolean validateEmpty(Context context, EditText et){
    if(isInputEmpty(et)) {
        setErrorToInput(context, et,
                        context.getString(R.string.error));
        return false;
    }
    return true;
}
```

Kód 5.5: Validace prázdnosti komponenty EditText

Metoda `setErrorToInput(Context, EditText, String)` se stará o zobrazení chybové zprávy uživateli v případě špatně vyplněného pole.

Kapitola 6

Testování

Důležitou součástí vývoje je také otestování implementovaného řešení problému. Testování probíhalo ve třech fázích. Nejdříve jsem vytvořil unit testy a instrumentální testy tak, aby splňovaly principy testovací pyramidy zmíněné v rešeršní kapitole 2.8, a nakonec jsem nechal aplikaci otestovat několika uživatelům.

6.1 Small testy

Během implementace unit testů jsem se soustředil na testování pomocných tříd, které obalují statické metody, a to třídy `Utils` a `Validate`. Dále jsem vytvořil testy na netriviální metody tříd, které ke svému běhu nepotřebují kontext aplikace jako jsou například databázové konvertory a entitní třídy. Pro tvorbu unit testů jsem použil JUnit 4.12 a pro mockování framework Mockito 2.10.

6.2 Medium testy

Těchto testů jsem využil při testování správného chování adaptéru napříč aplikací.

6.3 Large testy

Large testy jsem vytvořil za pomoci Espressa, se kterým jsem otestoval aktivity a komplexnější UI komponenty.

6.4 Uživatelské testy

Aplikaci jsem nechal otestovat 3 různými lidmi. Jednalo se o studenty vysokých škol, což je hlavní cílová skupina, na kterou aplikace míří. Každému uživateli jsem představil hlavní myšlenku celého projektu a nechal jsem ho založit si účet. Sám jsem vytvořil testovací firmu, která nabízela několik brigád. Uživatel si vyplnil profil a prošel si celý proces výběru brigády včetně komunikace s firmou. Každý uživatel mi následně sdělil své dojmy. Výsledky byly pozitivní. Všechny zapojené osoby zvládly bež pomoci celý proces. Líbil se jim jak chatovací modul, tak sdružující položky ve filtru. Dva uživatelé měli potíže s pochopením ukládání dat v profilu. Chybělo jim tam odesílací tlačítko.

Závěr

Cílem této práce bylo naimplementovat část mobilní aplikace, která slouží potencionálnímu brigádníkovi k hledání přivýdělku. Dále navrhnout a vytvořit chatovací modul, díky kterému se zjednoduší komunikace mezi pracovníkem a firmou.

V první části práce jsem v rámci rešerše zjistil, jakými způsoby je možné řešit problematiky, které se vztahují k jednotlivým cílům této práce. V rámci analýzy jsem z těchto možných způsobů řešení vybral ty, které použiji při vývoji a vytvořil výčet požadavků a případů užití chatovacího modulu.

Dále jsem vytvořil podrobný návrh jak chatovacího modulu, tak modulu pro pracovníky. Začal jsem od obecných pojmů jako získávání a odesílání dat a postupně jsem přecházel ke konkrétnějším funkcionalitám.

V implementační kapitole jsem popsal, jakým způsobem jsem vyřešil některé z komplexnějších problémů, které vyplývaly přímo z návrhu aplikace. Pro otestování realizované části jsem vytvořil 3 skupiny testů, které ověřují správnou funkčnost tříd, komunikaci mezi komponentami a uživatelské prostředí.

Stanovené cíle jsem splnil. Jako rozšíření jsem přidal zobrazení veškerých notifikací, které pracovník může získat z webového portálu. Od zobrazení adresy na vykreslené mapě Prahy jsem opustil po domluvě se zadavatelkou projektu. Místo toho jsem zvolil odkaz do vestavěných map. Do budoucna je potřeba dodělat synchronizaci kalendářů mezi aplikací a serverem. Užitečným rozšířením by byla možnost odeslat fotografii v chatu a zprostředkovat uživateli informaci o tom, zdali je jeho zpráva příjemcem přečtena.

Literatura

[1] Mroček, M.: *Android aplikace Daywork.cz: jádro*. Bakalářská práce, praha, 2018, České vysoké učení technické v Praze. Fakulta informačních technologií, Ing. Robert Pergl Ph.D., Nepublikovaná práce.

[2] Hakeem, H.: Android by example : MVVM +Data Binding -> Introduction (Part 1) [online]. *Medium*, [cit. 2018-04-12]. Dostupné z: `https://medium.com/@husayn.hakeem/android-by-example-mvvm-data-binding-introduction-part-1-6a7a5f388bf7`

[3] Novotný, K.: *Android aplikace Daywork.cz: design*. Bakalářská práce, praha, 2018, České vysoké učení technické v Praze. Fakulta informačních technologií, Ing. Robert Pergl Ph.D., Nepublikovaná práce.

[4] Developers Android: *ListView [online]*. 2016, [cit. 2018-04-12]. Dostupné z: `https://developer.android.com/reference/android/widget/ListView.html`

[5] Rittmeyer, W.: A First Glance at Android's RecyclerView [online]. *Grokking Android*, [cit. 2018-04-13]. Dostupné z: `https://www.grokkingandroid.com/first-glance-androids-recyclerview`

[6] Uknown: Populating a ListView with a CursorAdapter [online]. *Codepath*, [cit. 2018-04-13]. Dostupné z: `https://guides.codepath.com/android/Populating-a-ListView-with-a-CursorAdapter`

[7] Developer Android: *RecyclerView.Adapter [online]*. [cit. 2018-04-13]. Dostupné z: `https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html`

[8] *PagedListAdapter [online]*. [cit. 2018-04-13]. Dostupné z: `https://developer.android.com/reference/android/arch/paging/PagedListAdapter.html`

[9] Panigrahy, N.: How to Use Picasso Image Loader Library in Android [online]. *Stacktips*, [cit. 2018-04-13]. Dostupné z: http://stacktips.com/tutorials/android/how-to-use-picasso-library-in-android

[10] Tarasevich, S.: *Universal Image Loader [online]*. 2015, [cit. 2018-04-13]. Dostupné z: http://stacktips.com/tutorials/android/how-to-use-picasso-library-in-android

[11] N, C.: My Experience using Fresco [online]. *Medium*, [cit. 2018-04-13]. Dostupné z: https://medium.com/myntra-engineering/my-experience-using-fresco-b45cc36ac878

[12] Codepath: Displaying Images with the Glide Library [online]. *Codepath*, [cit. 2018-04-13]. Dostupné z: https://guides.codepath.com/android/Displaying-Images-with-the-Glide-Library

[13] Anon: SnackBar Vs Toast [online]. *Programmer Guru*, [cit. 2018-04-13]. Dostupné z: http://programmerguru.com/android-tutorial/snackbar-vs-toast/

[14] Murphy, M.: *The Busy Coder's Guide to Advanced Android Development*. CommonsWare, LLC, 2011, ISBN 9780981678054.

[15] Codepath: Using DialogFragment [online]. *Codepath*, [cit. 2018-04-13]. Dostupné z: https://guides.codepath.com/android/using-dialogfragment

[16] *Fundamentals of Testing [online]*. [cit. 2018-04-16]. Dostupné z: https://developer.android.com/training/testing/fundamentals

Příloha A

Seznam použitých zkratek

CRUD Create, Read, Update, Delete

MVVM Model–view–viewModel

Fcm Firebase cloud messaging

UI User Interface

XML eXtensible Markup Language

API Application Programming Interface

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `apk` ..... adresář s apk aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS