



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

**BOOKSHARING – MOBILNÍ APLIKACE PRO SDÍLENÍ
KNIH**

BOOKSHARING – MOBILE APPLICATION FOR BOOK SHARING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN BALÁŽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL KAPINUS

BRNO 2023

Zadání bakalářské práce



146926

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Baláž Martin**
Program: Informační technologie
Specializace: Informační technologie
Název: **BookSharing – mobilní aplikace pro sdílení knih**
Kategorie: Uživatelská rozhraní
Akademický rok: 2022/23

Zadání:

1. Prostudujte postupy návrhu uživatelských rozhraní moderních mobilních aplikací. Seznamte se s platformou Android a nastudujte možnosti a specifika tvorby mobilních aplikací pro tuto platformu.
2. Navrhněte aplikaci umožňující sdílení knih. Aplikace by měla umožnit nabídnout vlastní knihy k výpůjčce a zároveň vypůjčit knihy nabídnuté ostatními uživateli.
3. U výpůjček půjde nastavit parametry jako poplatek za výpůjčku, možná místa předání a délka výpůjčky. Aplikace by také měla obsahovat možnost hodnocení uživatelů na základě proběhlých výpůjček. Zaměřte se na použitelnost a uživatelskou zkušenosť.
4. Navrženou aplikaci implementujte pro platformu Android.
5. Proveďte uživatelské experimenty, demonstrujte a diskutujte vlastnosti vašeho řešení.
6. Vytvořte video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016
- Android Developers: <https://developer.android.com/index.html>

Při obhajobě semestrální části projektu je požadováno:

Body 1, 2, 3 a rozpracovaný bod 4.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Kapinus Michal, Ing.**

Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.

Datum zadání: 1.11.2022

Termín pro odevzdání: 10.5.2023

Datum schválení: 31.10.2022

Abstrakt

Cílem této bakalářské práce je vytvoření mobilní aplikace pro půjčování knih mezi uživateli. Uživatelé mohou přidávat knihy k vypůjčení a také si je vypůjčovat od ostatních uživatelů, s možností platby za půjčení. Aplikace je vyvinuta na operační systém Android s použitím programovacího jazyka Kotlin. Aplikace komunikuje se serverem, který je také vyvinut v jazyce Kotlin s využitím frameworku Ktor. Server zajišťuje ukládání, filtraci a zpracování dat uložených v databázi na serveru, která jsou nezbytná pro správný chod aplikace. Uživatelské rozhraní aplikace bylo společně s uživatelskou zkušeností testováno u šesti uživatelů a výsledky byly nadprůměrné.

Abstract

The goal of this bachelor thesis is to create a mobile application for lending books between users. Users can add books to lend and also borrow them from other users, with the option of paying for the loan. The application is developed for the Android operating system using the Kotlin programming language. The application communicates with a server that is also developed in Kotlin using the Ktor framework. The server handles the storage, filtering, and processing of data that are stored in the server database and that are necessary for the proper operation of the application. The application's user interface and user experience were tested with six users and the results were above average.

Klíčová slova

mobilní aplikace, Android, Kotlin, Ktor, půjčování, knihy, Firebase Cloud Messaging, databáze, API, uživatelská zkušenost, uživatelské rozhraní

Keywords

mobile application, Android, Kotlin, Ktor, lending, books, Firebase Cloud Messaging, database, API, user experience, user interface

Citace

BALÁŽ, Martin. *BookSharing – mobilní aplikace pro sdílení knih*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

BookSharing – mobilní aplikace pro sdílení knih

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Kapinuse. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Baláž
7. května 2023

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Michalu Kapinusovi za vedení a cenné rady, které mi byly velkou pomocí během psaní této práce a při vývoji aplikace. Dále bych chtěl poděkovat svým přátelům a rodině, kteří mi pomohli otestovat aplikaci a poskytli mi cennou zpětnou vazbu.

Obsah

1	Úvod	2
2	Vývoj mobilní aplikace na Android	3
2.1	Android	3
2.2	Uživatelské rozhraní	6
2.3	Uživatelská zkušenosť	8
2.4	Architektonické vzory	9
3	Návrh řešení	12
3.1	Průzkum podobných řešení	12
3.2	Návrh vlastního řešení	17
3.3	Návrh databáze	23
3.4	Návrh API	25
4	Implementace	28
4.1	Implementace aplikace BookSharing	28
4.2	Implementace serveru	30
4.3	Implementace klienta	36
4.4	Nedostatky implementace	43
5	Testování aplikace	45
5.1	Průběh testování s uživateli	45
5.2	Zpětná vazba uživatelů	46
5.3	Závěrečné hodnocení aplikace	47
6	Závěr	48
	Literatura	49
A	Obsah přiloženého paměťového média	51

Kapitola 1

Úvod

V dnešní době stále mnoho lidí čte fyzické knihy a mnozí si je buďto půjčují z knihoven nebo si je kupují. Nicméně, tito lidé nemají jednoduchý způsob, jak se podělit o své knihy s cizími lidmi a případně se i seznámit s lidmi, kteří mají podobné zájmy a vkus v knihách.

Cílem této bakalářské práce je vytvořit mobilní aplikaci pro zařízení s operačním systémem Android, která umožní uživatelům půjčovat si knihy navzájem a interagovat mezi sebou prostřednictvím oznamení. Aplikace také umožňuje placené výpůjčky, což uživatelům umožní vydělat si peníze a zabránit tak tomu, aby knihy ležely doma ladem, nebo byly prodány pod cenou. Aplikace pouze zprostředkovává kontaktní údaje, ale samotná komunikace mezi dvěma uživateli a předání knihy probíhá mimo aplikaci. Majitel knihy musí také zaevidovat vrácení knihy v aplikaci, protože aplikace nemá žádné zaměstnance, kteří by dohlíželi na výpůjčky. Po následném vrácení dané knihy se mohou obě strany ohodnotit a napsat recenzi, což přispívá důvěryhodnosti uživatelů. Uživatelé si pak na základě těchto hodnocení mohou udělat obrázek o uživateli, než se rozhodnou mu knihu půjčit. Důležité údaje, jako je e-mail pro kontaktování a telefonní číslo, jsou před uživateli z bezpečnostních důvodů skryty a jsou zobrazeny teprve poté, co si mezi sebou potvrdí nějakou výpůjčku.

V teoretické kapitole 2 jsou uvedeny nejdůležitější informace potřebné k vývoji aplikací pro Android. V kapitole 3 jsou popsány podobné aplikace, což zahrnuje aplikaci pro půjčování knih a aut. Dále se v této kapitole nachází návrh uživatelského rozhraní a další důležité návrhy. Implementační část v kapitole 4 obsahuje podrobné informace o praktické části bakalářské práce, včetně implementace klienta a serveru. Nakonec je popsáno testování s uživateli v kapitole 5, které poskytlo cennou zpětnou vazbu od uživatelů. Tyto zpětné vazby, spolu s nedostatky mé implementace, jsou zahrnuty do možného budoucího vývoje.

Kapitola 2

Vývoj mobilní aplikace na Android

Má bakalářská práce se vztahuje na mobilní zařízení s operačním systémem Android, a z toho důvodu se bude v následujících podkapitolách pojednávat o důležitých aspektech vývoje. Tyto podkapitoly obsahují například popis výhod a nevýhod jednotlivých programovacích jazyků a architektonických vzorů. Mimo jiné jsou zde podrobnější vysvětlení uživatelského rozhraní a uživatelské zkušenosti, které jsou pro vývoj nesmírně důležité.

2.1 Android

Android je nejrozšířenější operační systém pro mobilní zařízení na světě, který vyvinula firma Google. Používá se v mobilních telefonech, tablettech, hodinkách, televizích, ale i v náramcích a autech. Android běží na linuxovém jádře a je open-source, což znamená, že jeho zdrojový kód je volně dostupný a může tedy být instalován na zařízení různých výrobců. Na druhé straně, největší konkurent Androidu – iOS je uzavřený operační systém, kde Apple má kontrolu nad vývojem aplikací a jejich distribucí [10].

V říjnu 2008 byl představen světově první mobilní telefon s operačním systémem Android – HTC Dream, neboli T-mobile G1. Prodával se za cenu okolo 10 000 korun a do Česka se dostal na začátku roku 2009 [10].

V této podkapitole se dozvítíte o hlavních programovacích jazycích pro platformu Android a základních kamenech, na kterých Android stojí.

2.1.1 Programovací jazyky pro Android aplikace

Existuje mnoho programovacích jazyků, které lze použít pro vývoj mobilních aplikací, jako například Qt nebo Dart ve frameworku Flutter. Tyto jazyky umožňují vývoj aplikací pro oba operační systémy, Android a iOS. Nicméně, v této sekci se bude pojednávat pouze o dvou hlavních programovacích jazycích určené výhradně pro vývoj Android aplikací.

- **Java** – Java je objektově orientovaný programovací jazyk, který je navržený tak, aby měl co nejméně implementačních závislostí. Java byla vydaná v roce 1996 společností Sun Microsystems [11] .
- **Kotlin** – Kotlin je staticky typovaný programovací jazyk. Kotlin 1.0 byl vydaný na začátku roku 2016 společností JetBrains. Při komplikaci je přeložen do bytecode Javy, který běží v JVM (Java Virtual Machine). Kotlin není tak oblíbený ani rozšířený jako Java, nicméně je navržený právě pro vývoj mobilních aplikací [11] .

Google v roce 2017 rozhodl, že Kotlin bude hlavní programovací jazyk pro vývoj mobilních aplikací v integrovaném vývojovém prostředí (známé také pod anglickou zkratkou IDE, neboli Integrated Development Environment) Android studio.

Java vs. Kotlin

Není-li dále uvedeno jinak, tato podsekce vychází z informací získaných z [9]. Jednou z výhod Kotlinu je, že je navržený tak, aby toho programátoři zvládli víc za kratší dobu. Například u tříd se na rozdíl od Javy již automaticky generují metody pro získání a nastavení hodnoty (tj. getter a setter), obsahuje také funkci `copy`, která zkopíruje instanci a vytvoří novou, do které lze přesunout další hodnoty. V Javě sice k vyřešení úlohy napišeme víc kódu, ale máme více „know-how“ a hotových řešení, ze kterých můžeme čerpat.

V Javě je velkou nevýhodou `Nullable`. Když definujeme řetězcovou proměnnou `String name`, nelze s jistotou říct, zda v ní `name` má, nebo nemá hodnotu. Běžně se může stát, že při práci s proměnnou dostaneme výjimku `NullPointerException`. V Kotlinu máme možnost rozhodnout, zda proměnnou definujeme jako `Nullable` (`val name: String?`). Tady je možné s jistotou říci, že proměnná může obsahovat `null`. Pokud by proměnná byla definována bez otazníku, můžeme s jistotou říct, že proměnná nemůže být nikdy `null`. S tím souvisí tzv. Null safety, v Javě totiž musíme počítat s tím, že může být `null`, ale v Kotlinu jsme schopní to zajistit znakem „?“, což nám zajistí, že instrukce v případě `null` nepokračuje dál.

Pokud jde o práci s více vlákny, Kotlin má nativní podporu Kotlin Coroutines, neboli koprogramy¹. V Javě je nutné využít RxJava nebo callbacky (také známé jako zpětné volání), jejichž použití je složitější a méně přehledné. Coroutiny opět zjednoduší čitelnost psaného kódu, ale vyžadují ještě větší abstraktní představivost. Naopak v Javě je zase u RxJava nutné déle studovat kód a dokumentaci [11].

Navzdory všem rozdílům mezi těmito dvěma jazyky jsou Java a Kotlin 100% interoperabilní. Je možné volat kód Kotlinu z Javy a naopak. Je tedy možné mít třídy Kotlinu a Javy vedle sebe v rámci jednoho projektu a vše se bude stále kompilovat [1].

2.1.2 Základní stavební kameny Androidu

Každá Android aplikace musí mít alespoň jednu aktivitu, protože je to hlavní stavební kámen uživatelského rozhraní. Fragmenty jsou na druhé straně volitelným prvkem a aplikace může fungovat i bez nich. Tato sekce poskytuje detailnější popis obou těchto základních stavebních bloků.

Aktivita

Tato podsekce je převzata z oficiálního Android manuálu [6]. V rámci Android aplikací je aktivita určena na interakci s uživatelem. Aktivity se obvykle zobrazují přes celou obrazovku, ale mohou být také zobrazeny v plovoucím okně, nebo jako součást většího okna. Aktivity mohou být také použity v režimu více oken. Třída `Activity` se stará o vytvoření uživatelského rozhraní pomocí volání `setContentView()`. Aktivity jsou v systému spravovány pomocí zásobníku aktivit. Pokud je spuštěna nová aktivita, obvykle je umístěna na vrchol stávajícího zásobníku a stává se běžící aktivitou. Předchozí aktivita zůstává v zásobníku vždy pod ní a nezobrazí se opět na popředí, dokud nová aktivita neukončí svůj běh.

¹<https://cs.wikipedia.org/wiki/Koprogram>

Životní cyklus aktivity v systému Android je definován mezi prvním voláním metody `onCreate()` a posledním voláním metody `onDestroy()`. V metodě `onCreate()` se provádí inicializace celkového stavu aktivity a v metodě `onDestroy()` se uvolňují všechny zbývající zdroje. Například, pokud aktivita obsahuje vlákno, které běží na pozadí a stahuje data z internetu, může toto vlákno vytvořit v `onCreate()` a pak ho zastavit v `onDestroy()`.

Viditelný životní cyklus aktivity probíhá mezi voláním metod `onStart()` a `onStop()`. Během této doby může uživatel vidět aktivitu na obrazovce, ale nemusí s ní interagovat. Mezi těmito dvěma metodami lze udržovat zdroje, které jsou potřebné k zobrazení aktivity uživateli. Například je možné registrovat příjemce `android.content.BroadcastReceiver` v `onStart()` pro monitorování změn, které ovlivňují uživatelské rozhraní, a zrušit ho v `onStop()` poté, kdy uživatel přestane vidět to, co je zobrazeno. Metody `onStart()` a `onStop()` mohou být volány vícekrát, když se aktivita stává viditelnou a skrytou pro uživatele.

Životní cyklus aktivity v popředí se odehrává mezi voláním metod `onResume()` a `onPause()`. Během této doby je aktivita viditelná, aktivní a interaguje s uživatelem. Aktivita může často přecházet mezi stavy při zastavení zařízení, při doručení výsledků aktivity, atd. Proto by kód v těchto metodách mělo být poměrně lehké naprogramovat.

Fragment

Tato podsekce je převzata z oficiálního Android manuálu [7]. Fragmenty jsou prvkem uživatelského rozhraní v Androidu, který umožnuje znovupoužitelnost a flexibilitu. Fragmenty jsou závislé na hostiteli, kterým je aktivita. Přestože každý fragment má svůj vlastní životní cyklus, je tento cyklus ovlivněn hostitelem. Ideálním řešením je použít aktivitu na nejvyšší úrovni a vložit do nich fragmenty, které tvoří zbytek uživatelského rozhraní. Toto řešení umožňuje použití například navigačního panelu na úrovni aktivity, který je k dispozici v rámci všech fragmentů. Díky této flexibilitě a možnosti znovupoužití kódu jsou fragmenty velmi užitečným prvkem při vývoji aplikací pro Android.

Hlavní série metod životního cyklu, které jsou volány pro dosažení stavu `resumed`, tedy stavu, kdy lze začít interagovat s uživatelem, jsou:

- **onAttach** – Volá se, jakmile je fragment přidružen ke svému hostiteli.
- **onCreate** – Volá se pro počáteční vytvoření fragmentu.
- **onCreateView** – Vytvoří a vrátí hierarchii `View`, která je spojena s fragmentem.
- **onActivityCreated** – Říká fragmentu, že jeho hostitel dokončil své vlastní `onCreate()`.
- **onViewStateRestored** – Říká fragmentu, že všechny uložené stavy jeho hierarchie `View` byly obnoveny.
- **onStart** – Způsobí, že se fragment stane viditelným pro uživatele, poté co jeho hostitel byl spuštěn.
- **onResume** – Umožňuje fragmentu interagovat s uživatelem poté, co jeho hostitel byl obnoven.

Jakmile fragment již není používán, prochází reverzní sérií zpětných volání, neboli callbacků:

- **onPause** – Fragment již s uživatelem neinteraguje, buď protože je jeho hostitel po-zastaven, nebo protože operace fragmentu ho modifikují v hostiteli.
- **onStop** – Fragment již není viditelný pro uživatele, buď protože je jeho hostitel zastaven, nebo protože operace fragmentu ho modifikují v hostiteli.
- **onDestroyView** – Umožňuje fragmentu uvolnit zdroje spojené s jeho **View**.
- **onDestroy** – Volá se pro finální čištění stavu fragmentu.
- **onDetach** – Volá se okamžitě před tím, než se fragment již dále nepřidružuje k hos-titeli.

2.2 Uživatelské rozhraní

Uživatelské rozhraní (známé pod anglickou zkratkou UI – User Interface), pojednává pře-devší o grafické stránce aplikace. Patří sem tedy tlačítka, posuvníky, textová pole a další prvky s kterými uživatelé interagují. Grafičtí návrháři se musí však kromě interaktivních prvků starat například o barevná schémata, tvary prvků, druhy písma a jejich velikosti a spoustu dalších věcí [13].

Aby grafickým návrhářům byla umožněna rychlejší a snadnější tvorba uživatelského rozhraní a zároveň se sladil vzhled napříč různými vývojáři a designery, byl vytvořen Material design², což je návrhový jazyk od společnosti Google z roku 2014, jehož dalším účelem je také poskytnutí konzistentního a intuitivního uživatelského zážitku napříč různými platfor-mami a zařízeními.

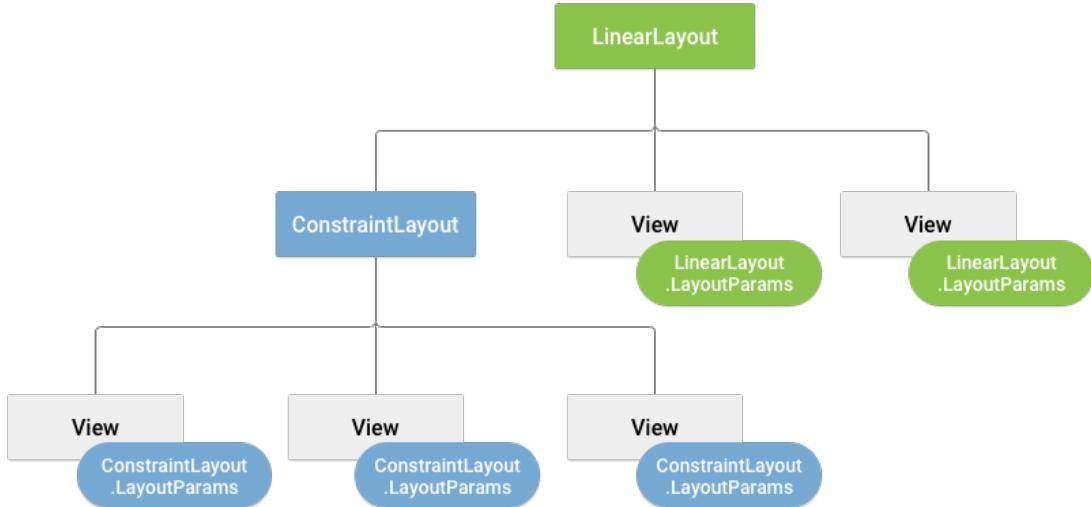
Ve vývoji Android aplikací je tvorba uživatelského rozhraní realizována pomocí jazyka XML, nebo pomocí Jetpack Compose knihovny pro jazyk Kotlin a právě o tom tato pod-kapitola pojednává.

The Extensible Markup Language

Tato sekce vychází z informací získaných z [8]. The Extensible Markup Language, neboli XML je obecný značkovací jazyk, který byl vyvinut a standardizován konsorcem W3C³. UI je v Androidu představováno hierarchickou strukturou, která se skládá z objektů **View** a **ViewGroup**. Objekty **View** mohou být například různé texty, obrázky a interaktivní prvky. Objekty **ViewGroup** představují kontejnery, známé jako Layouty, do kterých lze vkládat další objekty **View** a **ViewGroup**. Mezi nejpoužívanější typy Layoutů patří **ConstraintLayout**, kde si vývojáři mohou jednoduše prvky pozicovat v grafickém rozhraní, nebo například **LinearLayout**, kde jdou prvky horizontálně nebo vertikálně za sebou. Při použití XML je nutné zachovat hierarchickou strukturu, která musí být tvořena pouze jedním počátečním prvkem. Příklad takové hierarchické struktury lze vidět na obrázku 2.1.

²<https://m3.material.io/>

³<https://www.w3.org/Consortium/>



Obrázek 2.1: Obrázek zobrazuje hierarchickou strukturu uživatelského rozhraní, která začíná vždy jedním nejvyšším objektem `ViewGroup`, což jsou kontejnery, neboli Layouty, například `LinearLayout`, `ConstraintLayout` atd. `ViewGroup` se dále může větvit na další `ViewGroup`, nebo `View`, do kterého patří například tlačítka, texty a další. Obrázek byl převzat z [8].

Každému prvku v XML struktuře je možné přiřadit nějaké atributy, které patří konkrétní instanci objektu. Hlavním atributem je ID, které lze přiřadit k jakémukoli prvku. Kromě ID však může být nastavena například výška, šířka, barva, odsazení, velikost a typ písma, pozice v rodičovském prvku, viditelnost, klikatelnost apod. V kódu aplikace lze tyto atributy také měnit, nebo je možné reagovat na konkrétní události, např. stisknutí tlačítka, zaškrtnutí zaškrťávacího políčka, změna textu, atd.

Jetpack Compose

Text v této sekci vychází z informací získaných z [14]. Jetpack Compose je moderní framework, s jehož pomocí lze tvořit uživatelsky přívětivé grafické rozhraní pro mobilní aplikace. Nabízí širokou škálu možností na úpravu komponent. Compose je pouze jednou ze součástí sady knihoven Jetpack. Samotný Jetpack byl vyvinut společností JetBrains a umožňuje tvorbu uživatelského rozhraní pouze v jazyce Kotlin, což může být nevýhodou pro vývojáře, kteří preferují programování v jazyce Java.

Compose vylepšil starší návrh používající XML v následujících ohledech:

- Ve starším návrhu bylo potřeba uživatelské rozhraní definovat v XML souboru. Jeho kód byl oproti Compose poměrně dlouhý. U složitějších aplikací se pak v něm hůře orientovalo. Compose nabízí jednodušší způsob tvorby návrhu UI pomocí deklarativního kódu.
- Při tvorbě UI pomocí XML byla změna stavu aplikace, například textu v komponentě, nutná upravovat v kódu. Tím vznikl problém, když programátor zapomněl změnit text nějaké komponenty. Compose vyřešil stejný problém použitím architektury nazvané single source of truth (SSOT – jediný zdroj pravdy).
- Compose dále umožňuje jednodušší tvorbu vlastních komponent. Dříve bylo potřeba zdědit třídu `View` a implementovat hned několik metod. Tento způsob tvorby kompo-

nent byl poměrně náročný. Oproti tomu v Jeptack Compose frameworku není potřeba žádnou třídu dělit. Komponenty jsou zde implementovány jako funkce, na rozdíl od staršího návrhu, kde jednotlivé komponenty reprezentují třídy.

2.3 Uživatelská zkušenost

Není-li dále uvedeno jinak, tato podkapitola byla inspirována z [12]. Uživatelská zkušenost, neboli také User experience známé pod zkratkou UX, se týká každé interakce, kterou uživatel má s produktem, nebo službou. Návrh UX zohledňuje každý prvek, který tvoří tuto zkušenosť, například jaké pocity vyvolává v uživateli, nebo jak snadno uživatel dosáhne svých požadovaných úkolů. Cílem návrhu UX je vytvářet snadné, efektivní, relevantní a celkově příjemné zážitky pro uživatele.

Návrh UX se skládá ze 3 důležitých faktorů a jednoho bonusu (na konci každé odrážky bude následovat příklad v rámci aplikace na objednávání jídla):

- **Nutnost splnit uživatelovy potřeby:** Ať už uživatel přišel udělat cokoli na nějaký produkt, nebo službu, měl by toho bez problému dosáhnout. Pokud si uživatel chce objednat jídlo v aplikaci na to vytvořenou, musí toho dosáhnout.
- **Jednoduché na používání a jednoduché na naučení:** Uživatel musí pochopit co ve službě dělat a jak ji používat, i když ji nikdy předtím nepoužil. Uživatel by v aplikaci měl být schopen bez námahy najít jídlo, které hledá, zadat důležité informace jako adresu, jméno atd. a objednat jídlo.
- **Nutnost dát uživateli kontrolu a svobodu:** Uživatel by měl mít možnost rozmyslet si nějakou volbu a zrušit jeho předchozí akce. Pokud by si uživatel v aplikaci objednal jídlo, ale rozmyslel si to, měl by mít možnost objednávku ještě zrušit.
- **Překvapení a potěšení:** Cokoli co udělá uživateli radost a donutí ho cítit se výjimečně, aby chtěl použít produkt příště znova. Například při používání aplikace na objednání jídla, dostane uživatel při příštém nákupu zdarma tašku.

Důvod proč je dobrá UX velice důležitá je ten, že uživatelé se s vyšší pravděpodobností budou vracet na danou službu znovu a znovu a budou ji doporučovat přátelům a svým blízkým.

Základní pilíře UX

Tato sekce vychází z informací získaných z [2, 5]. Kvadrantový model UX popisuje čtyři hlavní oblasti, které tvoří ucelenou uživatelskou zkušenosť.

- **Experience Strategy (ExS)** – Zaměřuje se na definování strategie pro uživatelskou zkušenosť v souladu s obchodními cíli a potřebami uživatelů. Tato disciplína zahrnuje výzkum trhu, analýzu konkurence, stanovení cílových skupin a vytvoření strategie pro poskytování přidané hodnoty.
- **Interaction Design (IxD)** – Zaměřuje se na návrh interakcí mezi uživatelem a produktem, včetně ovládacích prvků, navigace a výběru obsahu. Cílem této disciplíny je udělat daný produkt co nejintuitivnější. Tato disciplína zahrnuje vytváření wireframeů, prototypů a testování uživatelských rozhraní.

- **User Research (UR)** – Zaměřuje se na pochopení potřeb a chování uživatelů, včetně analýzy uživatelských dat, provádění uživatelských testů a vytváření uživatelských person. Tato disciplína pomáhá designérům vytvářet produkty, které odpovídají potřebám uživatelů.
- **Information Architecture (IA)** – Zaměřuje se na organizaci a strukturu obsahu v produktu, aby byl pro uživatele snadno srozumitelný a přístupný. Tato disciplína zahrnuje tvorbu hierarchií obsahu, klasifikaci a organizaci dat. Účinná informační architektura pomáhá uživatelům identifikovat, kde se nacházejí a kam v daném systému kliknout, aby našli informace, které hledají.

2.4 Architektonické vzory

Není-li dále uvedeno jinak, tato podkapitola byla převzata z [4]. Architektura je jedním z klíčových prvků moderního vývoje pro platformu Android, který určuje celkovou složitost kódů a náklady na řízení projektu. Architektura aplikace výrazně ovlivňuje složitost, škálovatelnost a robustnost projektu a umožňuje snadné testování. Definováním hranic mezi jednotlivými vrstvami můžete jasně definovat jejich odpovědnosti a oddělit každou odpovědnost modulárním způsobem pomocí specifických rolí.

V minulosti se trendy v architektuře pro platformu Android měnily v závislosti na dostupných řešeních a nejlepších postupech. Před sedmi až osmi lety byly projekty pro Android vytvářeny s využitím architektonických vzorů MVC (Model-View-Controller) a MVP (Model-View-Presenter), ale nyní jsou většinou postaveny na architektonických vzorech MVVM (Model-View-ViewModel) a MVI (Model-View-Intent), protože byly představeny užitečné nástroje pro odběr informací jako RxKotlin, Kotlin Flows a LiveData.

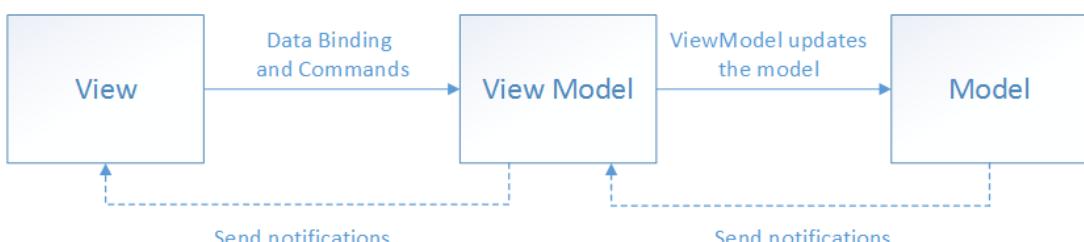
V této podkapitole se dozvíte o nejpopulárnějších architektonických vzorech posledních let: MVVM a MVI.

2.4.1 Model-View-ViewModel

V současnosti je MVVM jedním z nejpopulárnějších architektonických vzorů v moderním vývoji pro Android. A to od té doby, kdy Google oficiálně oznámil architektonické komponenty, jako jsou ViewModel, LiveData a Data Binding.

Historicky byl MVVM návrh využíván již více než desetiletí WPF⁴ vývojáři od doby, kdy byl představen vzor MVVM společností Microsoft.

Vzor MVVM se skládá z View, ViewModelu a Modelu, jak je vidět na obrázku 2.2.



Obrázek 2.2: Obrázek zobrazuje architektonický vzor Model-View-ViewModel a vztahy mezi jednotlivými komponenty. Obrázek byl převzat z [4].

⁴WPF – Windows Presentation Foundation

Každá komponenta má při vývoji Android aplikací na starosti různé úkoly, které jsou popsány níže:

- **View:** Je zodpovědný za konstrukci uživatelských rozhraní (dále jen UI), které uživatel vidí na obrazovce. View se skládá z komponent Androidu, které zahrnují prvky UI, jako jsou `TextView`, `Button`, `ImageView` a další. Prvky UI předávají uživatelské akce do ViewModelu a konfigurují obrazovky UI pozorováním dat nebo stavů UI z ViewModelu. Ideálně by měl View obsahovat pouze logiku UI, která reprezentuje obrazovku a interakce uživatele, jako jsou posluchače, a neměl by obsahovat business logiku⁵ (o tu by se měl starat Model).
- **ViewModel:** Je nezávislá komponenta, která nemá žádné závislosti na View a uchovává data nebo stavы UI z Modelu, aby je mohl předat prvkům UI (do View). Obvykle existují 1:N vztahy mezi ViewModelem a View, což znamená, že jeden ViewModel může být použit pro více View, které sdílejí stejná data nebo stavы UI. ViewModel informuje View o změnách pomocí tzv. observatelných objektů (observers), kteří pozorují změny a informují View, aby aktualizoval své UI.
- **Model:** Zapouzdřuje datový model/doménu aplikace, který obvykle zahrnuje business logiku, složitou výpočetní práci a validační logiku. Třídy Modelu jsou obvykle používány společně s vzdálenými službami a lokálními databázemi v repozitářích, které zapouzdřují přístup k datům jako kolekce spustitelných doménových funkcí. Repozitáře zajistují jediný zdroj pravdy⁶ z více zdrojů dat a nezměnitelnost pro celková data aplikace.

2.4.2 Model -View - Intent

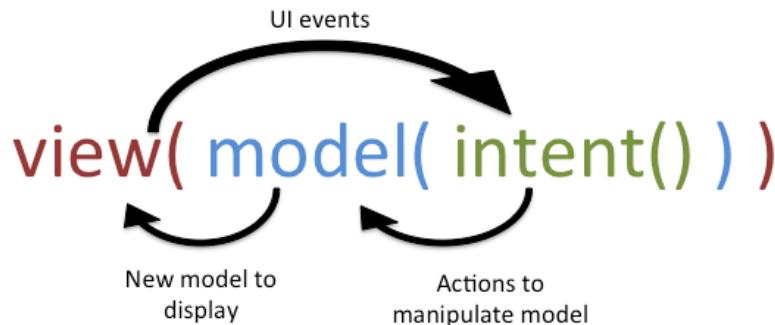
Tato sekce také vychází z informací získaných z [3]. MVI je také populární architektonický vzor v moderním vývoji pro Android od doby, kdy Jetpack Compose přinesl deklarativní programování do našich životů.

MVI se snaží poskytnout neměnné a jednosměrné stavы, které představují výsledek akcí uživatele a konfigurují obrazovky UI, a to pomocí principu jediného zdroje pravdy. MVI funguje na vrcholu jiných vzorů, jako jsou MVP nebo MVVM, s mechanismy řízení stavu, což znamená, že architektura MVI může přinést koncepty Presenteru nebo ViewModelu v závislosti na architektonickém návrhu.

Vzor MVI se skládá z View, Intentu a Modelu, jak lze vidět na obrázku 2.3.

⁵Business logika je souborem pravidel, vztahů a entit popisující fungování nějakého skutečného systému.

⁶Jediný zdroj pravdy (Single source of truth) je zdroj dat, který slouží jako jediný a spolehlivý zdroj informací pro aplikaci.



Obrázek 2.3: Na obrázku je vidět architektonický vzor MVI, jež demonstruje jednosměrný kruhový tok. Ve chvíli, kdy uživatel vyvolá nějakou akci na UI, vyvolá se a spustí se Intent, který manipuluje s modelem, jenž následně „změní“ svůj stav a UI tento nový model zobrazí. Obrázek byl převzat z [4].

Na rozdíl od MVVM a MVP je definice každého komponentu MVI mírně odlišná:

- **Intent:** Intent je definice rozhraní a funkcí, které zpracovávají uživatelské akce (události v UI, jako např. kliknutí na tlačítko). Tyto funkce transformují události z UI na rozhraní Modelu a doručují výsledek do Modelu pro následnou manipulaci s daným výsledkem.
- **Model:** Definice Modelu v MVI se plně liší od MVP a MVVM. Model v MVI reprezentuje stav aplikace a zpracovává akce z UI, které získá z výstupu Intentu. Model v MVI pracuje s neměnným stavem, který vychází z business logiky aplikace a sleduje princip jediného zdroje pravdy a jednosměrného toku dat. Model je také zodpovědný za předávání stavů uživatelským rozhraním, které jsou odvozeny z Modelu a jsou neměnné, aby se zajistila konzistence aplikace.
- **View:** View v MVI má stejné odpovědnosti jako MVP a MVVM, což znamená, že představuje obrazovku a interakce uživatele, jako jsou posluchači, a neobsahuje business logiku. Jeden z největších rozdílů v implementaci oproti jiným vzorům je, že MVI zajíšťuje jednosměrný tok dat, takže View zobrazuje prvky UI v závislosti na stavech UI, které pocházejí z Modelu.

Kapitola 3

Návrh řešení

V této kapitole se zabývám jak průzkumem podobných řešení, tak i samotným návrhem. Má bakalářská práce (dále jen BP) je zaměřena na mobilní zařízení, a proto jsem zkoumal hlavně mobilní aplikace. Návrh grafického rozhraní jsem poté s inspirací vytvářel v programu Figma.

3.1 Průzkum podobných řešení

Existuje pouze jedna aplikace (alespoň taková, která se dá snadno sehnat), která byla vytvořena za stejným účelem jako má BP. Tato aplikace se jmenuje Book Share, ale není plně funkční a obsahuje spoustu chyb, špatné grafické uživatelské rozhraní (dále jen GUI – Graphics user interface) a také špatnou uživatelskou zkušenosť (dále jen UX – User experience). Proto jsem většinu inspirace hledal u mobilní aplikace HoppyGo¹, která slouží k půjčování aut mezi uživateli. Ta je na rozdíl od Book Share více uživatelsky přívětivá. Vyzkoušel jsem si i jiné aplikace, jako např. Airbnb, ale již jsem měl přehled o tom, jak bude má BP vypadat a tyto aplikace neobsahovaly už nic inovativního.

3.1.1 Book Share

Když uživatel poprvé zapne aplikaci, může se zaregistrovat, přihlásit se, nebo si prohlédnout knihy, které uživatelé nabízejí k výpůjčce, jako neregistrovaný uživatel. Aby se zvýšila bezpečnost a minimalizoval se počet stejných uživatelů, musí uživatel při registraci mimo jiné zadat také své telefonní číslo.

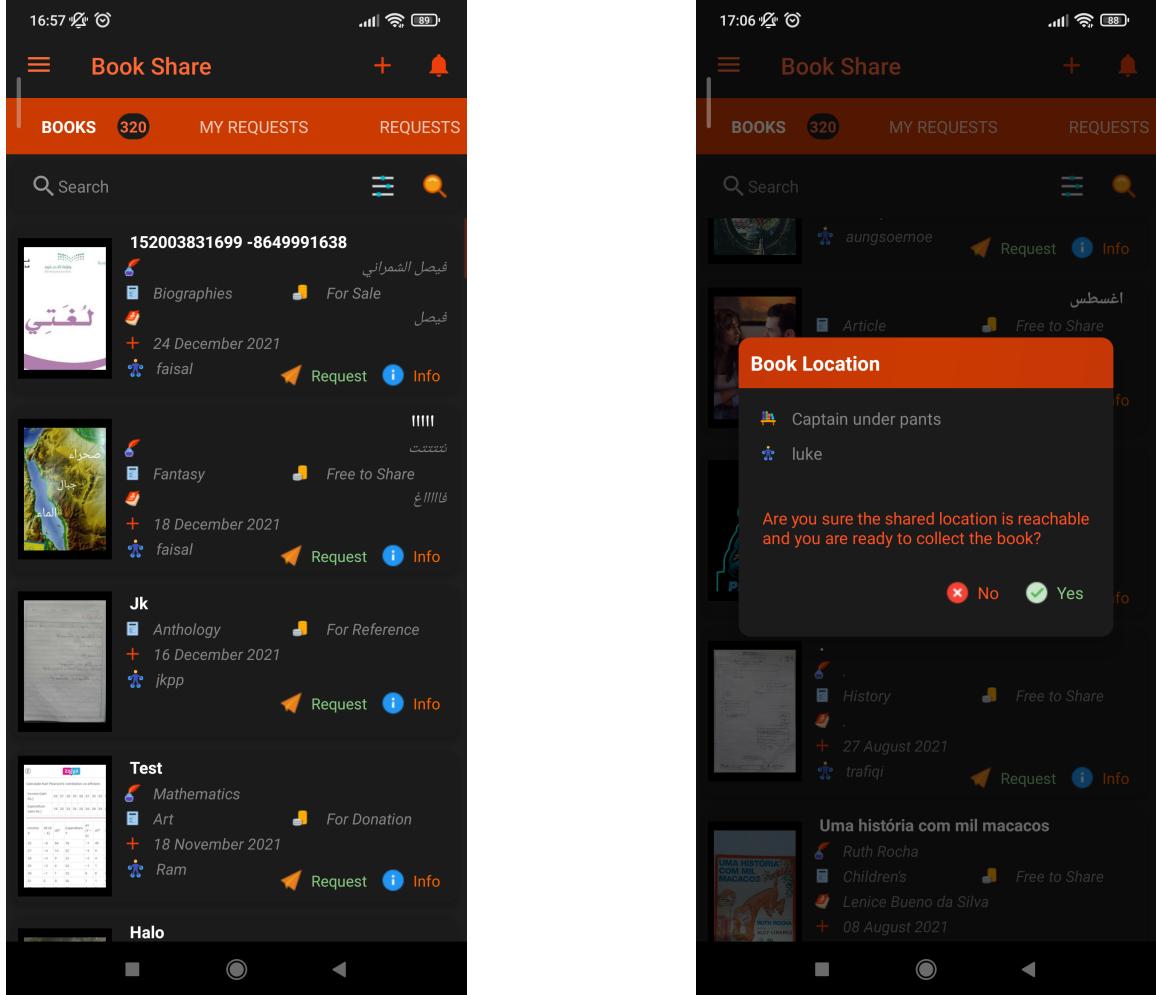
Na úvodní straně si uživatelé prohlížejí knihy, které jsou zde nabízeny k výpůjčce. Úvodní strana je vidět na obrázku 3.1a. Na přidání knihy k zapůjčení slouží „+“ nahoře vpravo vedle zvonečku. Zvoneček slouží k zobrazení příchozích oznámení. Uživatel si zde také může vyhledat knihu podle názvu. Filtrování není implementované a po stisknutí na tlačítko je uživatel informován, že tato funkcionalita vyde brzy.

Aplikace funguje na takovém principu, že pokud uživatel podá žádost o knihu, ukáže se mu tato žádost v okně „MY REQUESTS“ a druhému uživateli se tato žádost ukáže v okně „REQUESTS“, kde ji potvrdí, nebo odmítne.

Informace napsané u jednotlivých knih nedávají smysl, ikonky nejsou jednoznačně pochopitelné a navíc spoustu důležitých informací chybí, někdy i samotný název knihy. Po

¹<https://hoppygo.com/>

stisknutí tlačítka „Request“ u nějaké knihy vyskočí okno zobrazené na obrázku 3.1b, kde se aplikace ujistí, zda si uživatel chce opravdu tuto knihu vypůjčit.



(a) Úvodní strana

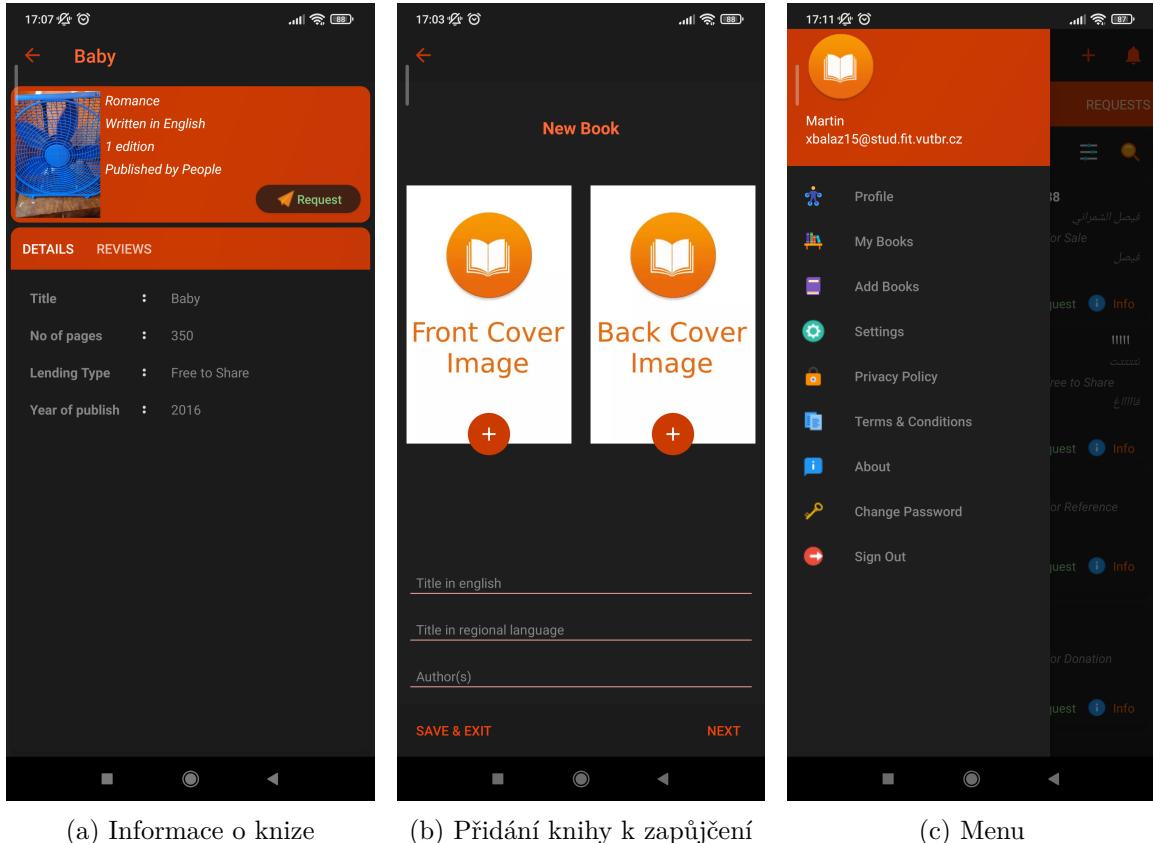
(b) Vyskakovací okno výpůjčky

Obrázek 3.1: První obrázek zobrazuje úvodní obrazovku, kde si uživatel může prohlížet a vyhledávat knihy, zobrazit jejich bližší informace, nebo požádat o vypůjčení. Když uživatel chce podat žádost o knihu, vyskočí dialogové okno zobrazené na druhém obrázku, ve kterém se ujišťuje zda si ji opravdu chce vypůjčit.

U každé knihy je možné se po stisknutí tlačítka „Info“ přesměrovat na okno s bližšími informacemi o knize (obrázek 3.2a). Na tomto obrázku si lze povšimnout obrázku modrého větráku na místě, kde má být obrázek knihy. Aplikace totiž nekontroluje při přidávání knihy, zda uživatel zadává lživé, či nějakým způsobem nevhodné informace a obrázky. Jednoduchý řešením tohoto problému by bylo přidat tlačítko na report, neboli nahlášení knihy. Administrátor pak na základě těchto reportů od uživatelů může tyto knihy odstranit. Takové řešení mám v plánu implementovat v mé řešení aplikace.

Uživatel může přidat knihu k zapůjčení buď prostřednictvím menu, nebo na úvodní straně. Přidávání knihy je zobrazeno na obrázku 3.2b, kde je možno vidět přidání obrázků přední i zadní strany knihy, názvu knihy a autora. Z uživatelského pohledu není možné

zjistit co se vyskytuje na další stránce, protože pokud uživatel stiskne tlačítko „NEXT“, vyskočí chybová hláška, že nebyl přidán obrázek knihy, i když ve skutečnosti byl. Menu z obrázku 3.2c obsahuje vcelku důležité věci, kterými se inspiruji ve vlastní aplikaci.



Obrázek 3.2: První obrázek zobrazuje informace o dané knize s nevhodným obrázkem. Uživatel si zde může požádat o vypůjčení knihy, nebo si zobrazit recenze. Na dalším obrázku je vidět proces přidávání knihy k zapůjčení a na posledním obrázku je zobrazeno menu.

Z průzkumu vyplývá, že ačkoli již existuje mobilní aplikace, která by mohla splňovat cíle mé BP, tak je však v nepoužitelném stavu. V momentálním stavu není možné přidat knihu, filtrovat knihy, informace a obrázky nikdo nekontroluje, a jak z průzkumu vyplynulo, tak uživatelé neberou tuto aplikaci vážně a nerespektují její pravidla. Aplikace tohoto typu se tedy neobejdou bez jisté kontroly přidávaných knih, např. pomocí tlačítka report. Přes všechny nedostatky Book Share má aplikace průměrné GUI a co se týče UX, není těžké najít věci, které uživatel potřebuje, ale je těžké pochopit některé informace, které knihy zobrazují.

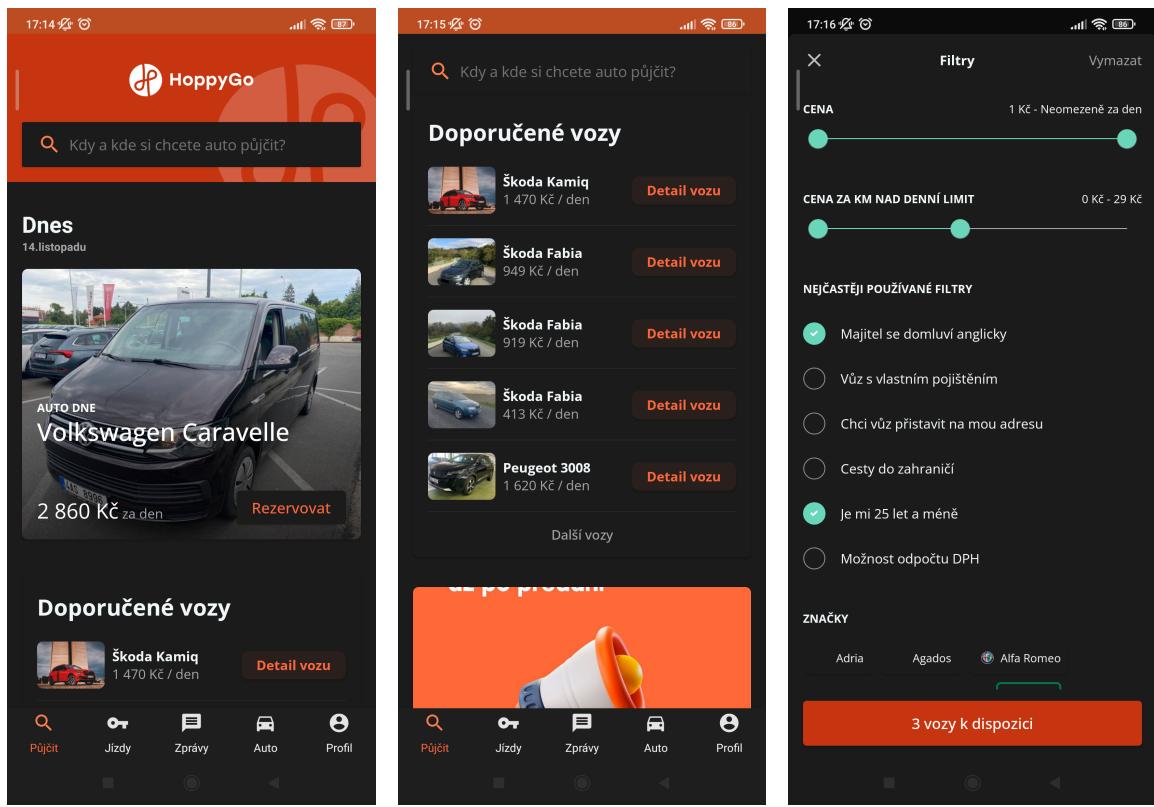
3.1.2 HoppyGo

Další zkoumanou aplikací je HoppyGo, která slouží na půjčování (pronajímání) vozidel mezi uživateli a to z toho důvodu, že princip je téměř stejný jako půjčování knih mezi uživateli a protože žádnoujinou aplikaci na téma mé BP jsem nenalezl.

Uživatel se může zaregistrovat, přihlásit se, nebo pokračovat na úvodní stranu. Uživatel dostane za registraci slevu 200 Kč na svou první výpůjčku auta, což značí výbornou UX.

Při registraci je však potřeba nahrát do systému vlastní řidičský a občanský průkaz. Tento krok jsem se ovšem z důvodu ochrany osobních údajů rozhodl nepodniknout. Aplikace je určena pro používání pouze v ČR, Polsku a na Slovensku, a proto je uživatel povinen vybrat si stát, ve kterém aplikaci bude používat.

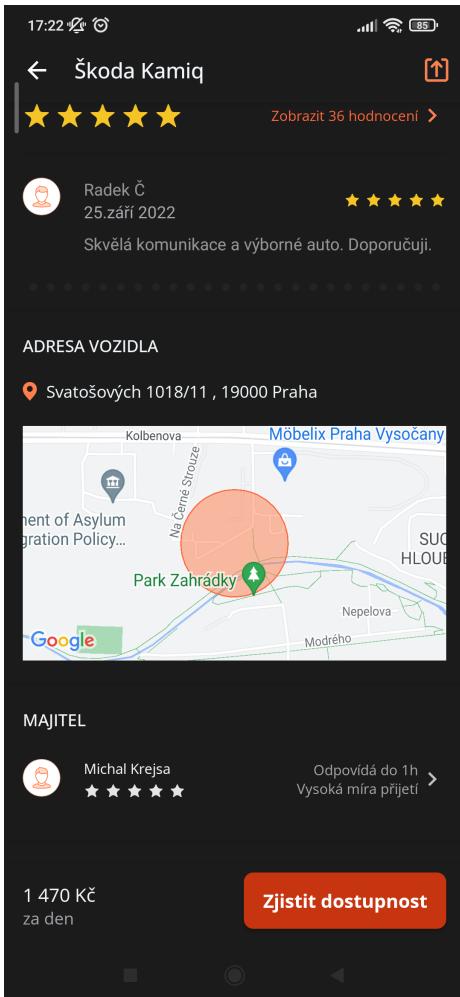
Úvodní strana (obrázek 3.3a a 3.3b) obsahuje vyhledávač, do kterého uživatel může zadat kdy a kde si chce auto vypůjčit. Hlavní náplní úvodní strany je auto dne a oblíbené vozy, který informují o ceně výpůjčky na den a obsahují tlačítko, které uživatele po stisknutí přesměruje na stranu, ve kterém se nachází informace o autě. Ve zbytku úvodní strany se nachází už pouze reklamy. Použití vyhledávače nasměruje uživatele na stranu, ve které se ukáží hledané výsledky. Zde se také nachází ikona filtru, která po stisknutí umožní uživateli specifikovat výběr. Na obrázku 3.3c je možno vidět filtr vozidel, který obsahuje cenu výpůjčky, nejčastěji používané filtry, značku auta, typ karoserie, převodovky a paliva, počet míst, různé možnosti výbavy a v poslední řadě rok výroby.



Obrázek 3.3: Na prvním a druhém obrázku je zobrazena úvodní strana, na kterém se nachází vyhledávač, auto dne, doporučené vozy a na konec reklamy. Na posledním obrázku je vidět část komplexního filtru.

Po vybrání vozidla a stisknutí na něj, se uživatel dostane na stranu informací o daném vozidle, jak je vidět na obrázcích 3.4a a 3.4b. Strana obsahuje obrázky vozidla a hned po nimi jsou hodnocení daného vozidla. Uživatelé mají možnost auto, které si jednou vypůjčili, orecenzovat a ohodnotit, lze však ohodnotit i samotného majitele vozidla. Na rozdíl od Book Share, kde se často ani nevyskytuje místo předání, je zde kromě toho i google mapa s okruhem místa výpůjčky. Dále jsou zde vypsány různé parametry vozidla, které jsou po-

psány výše v popisu filtrů. Každé vozidlo obsahuje ceník, ve kterém se mění cena výpůjčky na základě počtu dní výpůjčky, ale také se může zvýšit cena, pokud uživatel projede vozidlem více kilometrů, než je limit. Čím déle je auto vypůjčené, tím je levnější. Ceník je na konci stránky také rádově vysvětlen. Pokud se uživatel rozhodne vypůjčit si auto, vybere si v kalendáři rozsah výpůjčky, přičemž již zabraná místa jsou barevně označena.



(a) Informace o vybraném autě, část 1.



(b) Informace o vybraném autě, část 2.

Obrázek 3.4: Na obrázcích jsou zobrazeny informace o vybraném vozidle, jako například hodnocení majitele vozidla, ale i daného auta, adresa předání vozidla včetně google mapy, ceník pro daný počet dní výpůjčky a popis ceníku.

Z finálního hodnocení vyplývá, že aplikace, ačkoli je velice rozsáhlá, je zároveň velice jednoduchá na pochopení. Co se týče UX a GUI, uživatel rozhodně nebude zklamáný. Jediné mínu pro někoho být, že aplikace obsahuje spoustu reklam, ale tato aplikace je na rozdíl od Book Share řízena zaměstnanci, kteří se starají o bezpečnost uživatelů a chod aplikace. Reklamy slouží jako jeden z přínosů financí zaměstnancům aplikace, kteří si je bezpochyby zaslouží. Aplikace ale musí být mimořádně zabezpečená, protože vyžaduje citlivé informace od uživatelů, jako například fotografie řidičského a občanského průkazu. Hop-

pyGo nabízí také podporu a asistenční službu, která řeší různé problémy, např.: nevrácení vozidla majiteli.

3.2 Návrh vlastního řešení

Při vytváření návrhu mé aplikace, byla jistá inspirace aplikacemi z kapitoly 3.1. Od Book Share byla upravena a zpřehledněna úvodní strana a hlavní menu. Od HoppyGo byla zase inspirace se stránkou o informacích o dané knize a také s filtry na vyhledávání knih. Jako hlavní barevné schéma byl vybrán mix černošedé a oranžové barvy pro interaktivní komponenty. Při vytváření návrhu pro „Můj profil“ a „Oznámení“ byly využity mé vlastní nápady na uživatelské rozhraní a po otestování a vyslyšení názorů testovacích subjektů, byla odezva pozitivní až na pár drobností, které budou popsány níže.

Vstupní okno

Po spuštění aplikace BookSharing se objeví tzv. Vstupní okno, což je okno, ve kterém se uživatel může přihlásit, nebo zaregistrovat. Registrace uživatele vyžaduje od uživatele jeho e-mail, uživatelské jméno, datum narození, heslo a potvrzení souhlasu s podmínkami, zadání telefonního čísla je volitelné. Povinné údaje jsou označeny hvězdičkou. Součástí podmínek bude skutečnost, že e-mail použitý na přihlašování, bude také e-mail, který ostatní uživatelé uvidí, pokud si jej nezmění na straně „Můj profil“ v záložce „Kontakty“ jak je vidět na obrázku 3.8c. Přihlášení uživatele vyžaduje pouze e-mail a heslo, dále je zde také možnost zapamatování si uživatele do budoucna.

Úvodní strana

Po přihlášení se uživatel dostane na úvodní stranu (obrázek 3.5a). Na horní liště se nachází rozbalovací menu, které lze na obrázku vidět. Menu se nachází pouze na této straně, na ostatních je pouze šipka směrem doleva, která reprezentuje tlačítko zpět. To je z toho důvodu, že to vyhovovalo testovacím subjektům nejvíce, protože jsou na to již zvyklý z jiných aplikací. Menu obsahuje následující navigační položky: Můj profil, Oznámení, Přidat knihu, Moje knihy, Změna hesla, Odhlásit se. Dále horní lišta obsahuje ještě zvoneček, který uživatel přesměruje na stranu příchozích oznámení a také obsahuje fotku přihlášeného uživatele, která uživateli po stisknutí přesměruje na „Můj profil“.

Na úvodní straně se nachází vyhledávač, který bude vyhledávat knihy podle názvu, a vedle vyhledávače je tlačítko pro filtr. Dále se na stránce vyskytuje seznam jednotlivých knih seřazených sestupně od nejlepšího hodnocení po nejhorší, nejprve pro volné knihy a poté pro momentálně vypůjčené knihy. Každá položka seznamu obsahuje fotografii dané knihy, její název, autora, cenu (může být i zdarma), uživatelské jméno majitele knihy, jeho celkové hodnocení a informaci o dostupnosti. Uživatel má na této stránce na výběr dvě cesty, buď se rozhodne přidat novou knihu pomocí oranžového tlačítka „+“ v pravém dolním rohu, nebo si vybere nějakou knihu, kterou by si přál vypůjčit a stiskne ji.

Uživatel také může stisknout tlačítko pro filtrování obsahu na úvodní stránce, díky čemuž se objeví vyskakovací okno přes celou obrazovku (obrázek 3.5b). Na této stránce může uživatel filtrovat cenu pomocí tzv. slideru (česky posuvník), může vyfiltrovat jméno autora, nebo několik typů žánru knih, dále si může vyfiltrovat, zda si chce knihu vypůjčit osobně, nebo si ji chce nechat poslat na odběrné místo, záleží na jeho preferenci, může však i obojí. Může si vyfiltrovat zda kniha má být přístupna pouze pro dospělé, nebo pro všechny

věkové kategorie, přičemž pokud uživatel není plnoletý, nezobrazí se mu žádné knihy pro dospělé ať už s filtrem nebo bez filtru. Na posledním místě si lze vyfiltrovat dostupnost, jestli je kniha volná, nebo momentálně vypůjčená. Filtr obsahuje také tlačítko „vymazat“, které vymaze všechny nastavené filtry, křížek na odchod z okna, a oranžové tlačítko s fajfkou pro uplatnění filtrování.

The image consists of two screenshots of the BookSharing app. Screenshot (a) shows the main search results page with four book entries. Screenshot (b) shows a modal filter overlay with various filtering options.

Screenshot (a) - Úvodní strana (Home Screen):

- milk and honey** by *Rupi Kaur* - Zdarma, K dispozici ihned, rating 5 stars
- Institutes of the Christian religion** by *John Calvin* - Zdarma, K dispozici ihned, rating 5 stars
- The art of making memories** by *Meik Wiking* - 3 Kč / den, K dispozici ihned, rating 5 stars
- milk and honey** by *Rupi Kaur* - 4 Kč / den, K dispozici ihned, rating 5 stars

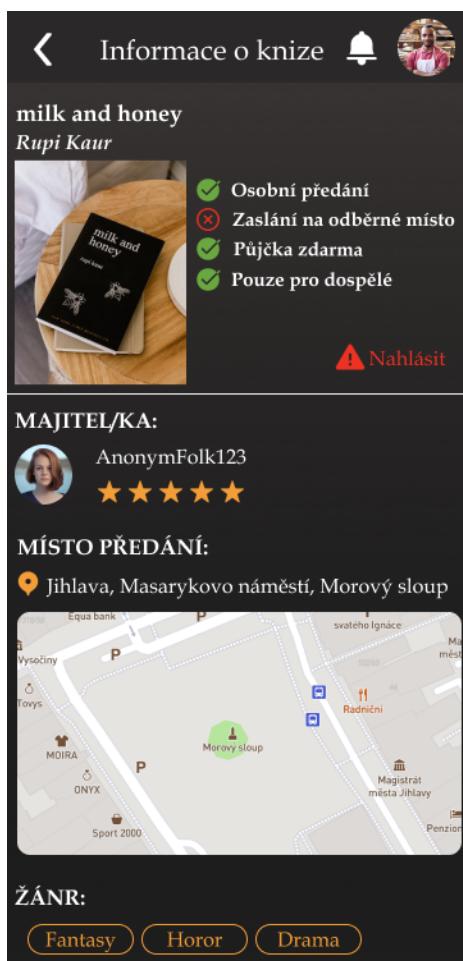
Screenshot (b) - Filtrování (Filter):

- Cena:** A slider from "Zdarma" to "Neomezeně za den".
- Jméno autora:** An input field.
- Žánry knih:**
 - Povídky, Fantasy
 - Romány, Sci-fi
 - Poezie, Horor
 - Drama, Detektivky
 - Komiksy
 - + Další žánry
- Možnosti zapůjčení:**
 - Osobní předání
 - Zaslání na odběrné místo
- Přístupnost:**
 - Pro dospělé
 - Pro všechny
- Dostupnost:**
 - Volná
 - Vypůjčená

Obrázek 3.5: Na prvním obrázku se nachází úvodní strana, na které se mimo vyhledávač a tlačítka filtrování nachází nabízené knihy, volné i vypůjčené, seřazené sestupně od nejlepšího hodnocení po nejhorší s příznakem volné, a poté s příznakem vypůjčené. Na druhém obrázku lze vidět různé typy filtrování, pomocí kterých si uživatel může zobrazit na úvodní straně pouze knihy s vybranými parametry.

Informace o knize

Na začátku této strany, která je vidět na obrázku 3.6a, si lze povšimnout vedle obrázku knihy, čtyř základních informací o knize, které uživatele nejvíce zajímají. Mezi ně patří: Osobní předání, Zaslání na odběrné místo, Půjčka zdarma a Pouze pro dospělé. Je zde také tlačítko „Nahlásit“, které po stisknutí otevře vyskakovací okno. Uživatel zde může nahlásit lživé, či nějakým způsobem nevhodné informace o dané knize, jako například obrázek větráku z obrázku 3.2a v aplikaci Book Share. Dále se na straně nachází majitel, po jehož stisknutí bude uživatel přesměrován na jeho profil. Pokud majitel vybral možnost osobního předání, bude zde napsáno místo předání, případně i vizuální zobrazení v podobě mapy. Mimo jiné se na této straně nachází maximální doba výpůjčky. Jakmile tato doba uplyne, přijde majiteli oznámení, ve kterém bude dotázán, zda mu byla kniha vrácena. Na úvodní straně bude také na základě této informace u vypůjčených knih napsána informace o tom, kdy přibližně by mohla být kniha volná. V poslední řadě je zde tlačítko „Požádat“ (obrázek 3.6b), po jehož stisknutí se objeví dialogové okno. Pokud žádost v okně potvrzdí, bude informován o tom, že dostane oznámení, jakmile majitel/ka jeho žádost potvrzdí. Pokud požádá o výpůjčku, když je kniha zrovna vypůjčená, přijde mu oznámení, až se uvolní.



(a) Informace o knize, část 1.

POPIS KNIHY:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Praesent auctor purus luctus enim egestas, ac scelerisque ante pulvinar. Donec ut rhoncus ex. Suspendisse ac rhoncus nisl, eu tempor urna. Curabitur vel bibendum lorem. Morbi convallis convallis diam sit amet lacinia. Aliquam in elementum tellus.

CENA VÝPŮJČKY:
Zdarma

STAV KNIHY:
Zachovalá, ohlé rohy

PŘÍSTUPNOST:
Pro všechny

MAX: DOBA VÝPŮJČKY:
90 dní

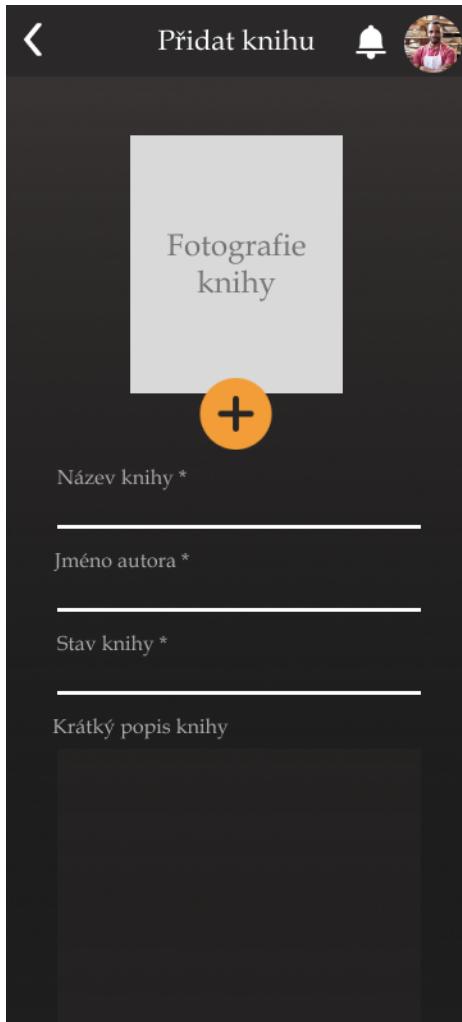
Požádat

(b) Informace o knize, část 2.

Obrázek 3.6: Na prvním a druhém obrázku jsou zobrazené informace o vybrané knize včetně jejího majitele s tlačítkem na požádání o výpůjčení.

Přidat knihu

V případě přidání knihy je potřeba zadat všechny požadované informace, s výjimkou popisu knihy. Na obrázcích 3.7a a 3.7b lze vidět formulář s požadovanými informacemi, nejsou zde však vidět všechny druhy žánrů. Uživatel musí mimo jiné zadat fotografii dané knihy, stav knihy, může si vybrat více než jeden druh žánru i možnosti zapůjčení. Pokud si zde vybere „Osobní předání“ a na profilu má vyplněné místo předání, bude zde automaticky předvyplněné, jinak jej bude muset zadat sám. Minimální hodnota délky zapůjčení je 7 dní a maximální hodnota činí 365 dní, neboli 1 rok.



(a) Přidat knihu, část 1.

The screenshot shows the second part of the 'Přidat knihu' (Add book) form. It includes a sidebar with categories: 'Cestopisy', 'Literatura faktu', 'Publicistika', and 'Biografie'. Below this is a section for lending options: 'Možnosti zapůjčení *' with a checked 'Osobní předání' option. A section for 'Místo předání *' follows. Further down are options for delivery: 'Zaslání na odběrné místo' (checked). Below that is a section for rental fees: 'Cena výpůjčky *' with 'Zdarma' (radio button) and 'Ceník' (radio button), and a field for 'Kč / den'. Another section for access rights: 'Přístupnost *' with 'Pro dospělé' (radio button) and 'Pro všechny' (radio button). Finally, there is a section for loan duration: 'Max. délka zapůjčení *' with a slider set between 1 and 365 days, and a 'den / dní' label. A large orange checkmark button is at the bottom right.

(b) Přidat knihu, část 2.

Obrázek 3.7: Na prvním a druhém obrázku je možno vidět přidání nové knihy, která bude zobrazena uživatelům v nabídce nabízených knih. Jsou zde vidět všechny důležité údaje o knize, které je potřeba vyplnit, přičemž je zde zobrazena pouze část žánrů.

Profil uživatele

Na obrázcích 3.8a, 3.8b a 3.8c lze vidět profil přihlášeného uživatele, přesněji „Můj profil“, který je rozdelen na 3 části. Fotografiu a uživatelské jméno uživatele lze upravit na všech částech. V první části (obrázek 3.8a) lze vidět všechny knihy, které uživatel nabízí k výpůjčce, jež jsou rozdělené do dvou kategorií na volné a vypůjčené. Volné knihy lze pomocí výběrové nabídky editovat, nebo odstranit (z databáze i aplikace). Pro vypůjčené knihy je tu možnost vrátit knihu do volných knih, pokud se uživatel rozhodne tak učinit, dostane na výběr ze dvou možností: Kniha byla úspěšně vrácena, Kniha nebyla vůbec vypůjčena. Pokud je uživateli kniha vrácena dříve, než po vypršení max. délky výpůjčky, vybere si tu první možnost a oběma uživatelům přijde oznámení, ve kterém se mohou navzájem ohodnotit a orecenzovat. Pokud kniha z nějakého důvodu nebyla vypůjčena, dostane oznámení na hodnocení pouze majitel knihy. V druhé části (obrázek 3.8b) jsou vidět udělené recenze od ostatních uživatelů a v poslední části (obrázek 3.8c) jsou uživatelovy kontaktní údaje, které lze doplnit a popřípadě editovat.

Profil jiného uživatele (tedy jiného než přihlášeného uživatele), vypadá stejně s tím rozdílem, že mu nelze nic editovat a záložka „Kontakty“ je mu nepřístupná, dokud mezi sebou nebudou mít potvrzenou nějakou výpůjčku. Přihlášený uživatel je na to upozorněn po pokusném stisknutí daného tlačítka.

Obrázek 3.8: Na prvním obrázku jsou zobrazené knihy, které přihlášený uživatel půjčuje, rozdělené na volné a vypůjčené. Na druhém obrázku jsou vidět hodnocení a recenze udělené přihlášenému uživateli a na třetím obrázku lze vidět kontakty uživatele, které si může upravit dle uvážení. Mimo jiné lze v této sekci změnit profilovou fotku a uživatelské jméno.

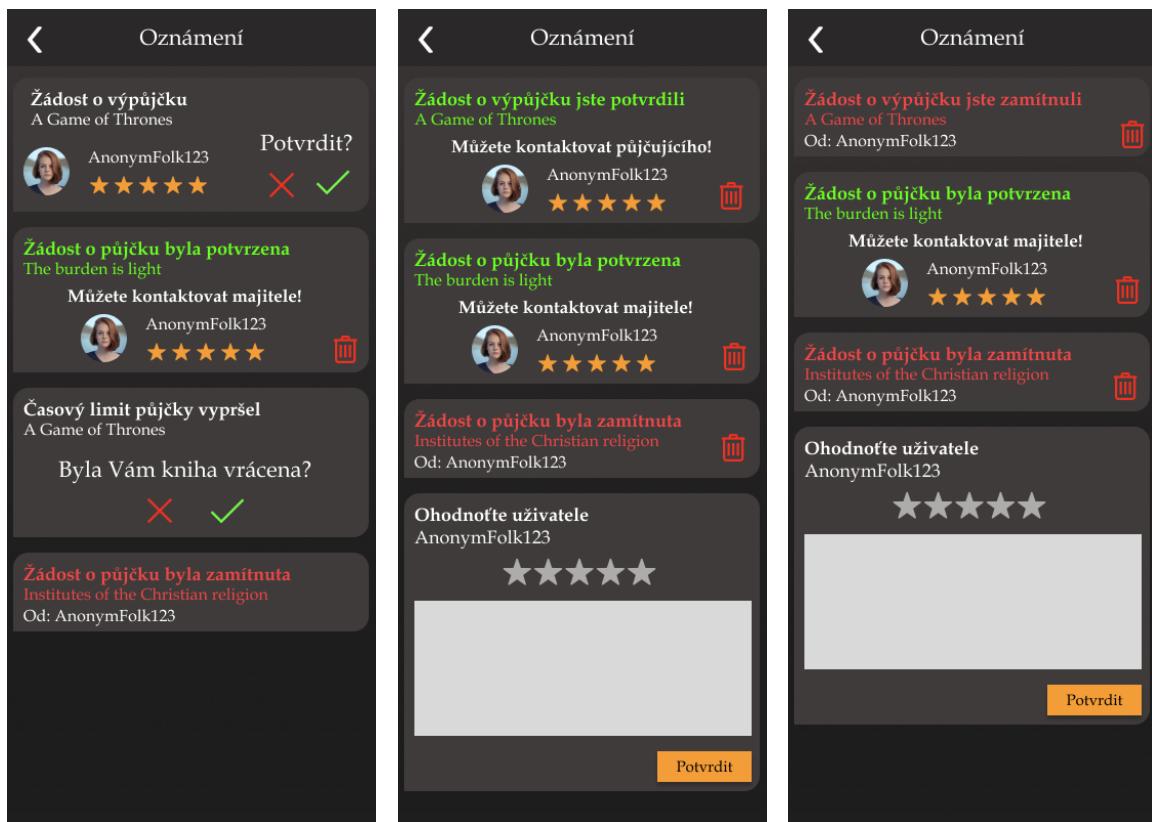
Oznámení

Je několik druhů oznámení, které může přihlášený uživatel obdržet:

- Interaktivní – potvrzení/zamítnutí žádosti o výpůjčku (obrázek 3.9a)
- Interaktivní – potvrzení/zamítnutí vrácení knihy (obrázek 3.9a)
- Interaktivní – ohodnocení uživatele (obrázek 3.9b a 3.9c)
- Oznamovací – žádost o půjčku byla potvrzena/zamítnuta (obrázek 3.9a, 3.9b, 3.9c)
- Oznamovací – žádost o výpůjčku jste potvrdili/zamítli (obrázek 3.9b, 3.9c)

Pokud se uživatel rozhodne potvrdit žádost o výpůjčku, dostanou obě strany oznámení o úspěšném potvrzení s odkazem na profil druhého uživatele na záložku „Kontakty“, je tedy na nich, kdo koho kontaktuje. V tento moment začne běžet čas předem nastavené maximální délky výpůjčky a pokud do té doby nebude kniha vrácena, přijde majiteli oznámení, ve kterém potvrdí, nebo zamítne vrácení knihy. Toto oznámení bude časově omezené, po uplynutí bude kniha odstraněna. Pokud jej potvrdí, přijde oběma uživatelům oznámení na vzájemné ohodnocení a kniha bude vrácena do volných knih. V případě, že jej odmítne, přijde toto oznámení pouze majiteli a zároveň se kniha odstraní z aplikace i databáze.

Udržuje se zde historie jednotlivých oznámení, uživatel má však možnost si je odstranit.



(a) Oznámení, část 1.

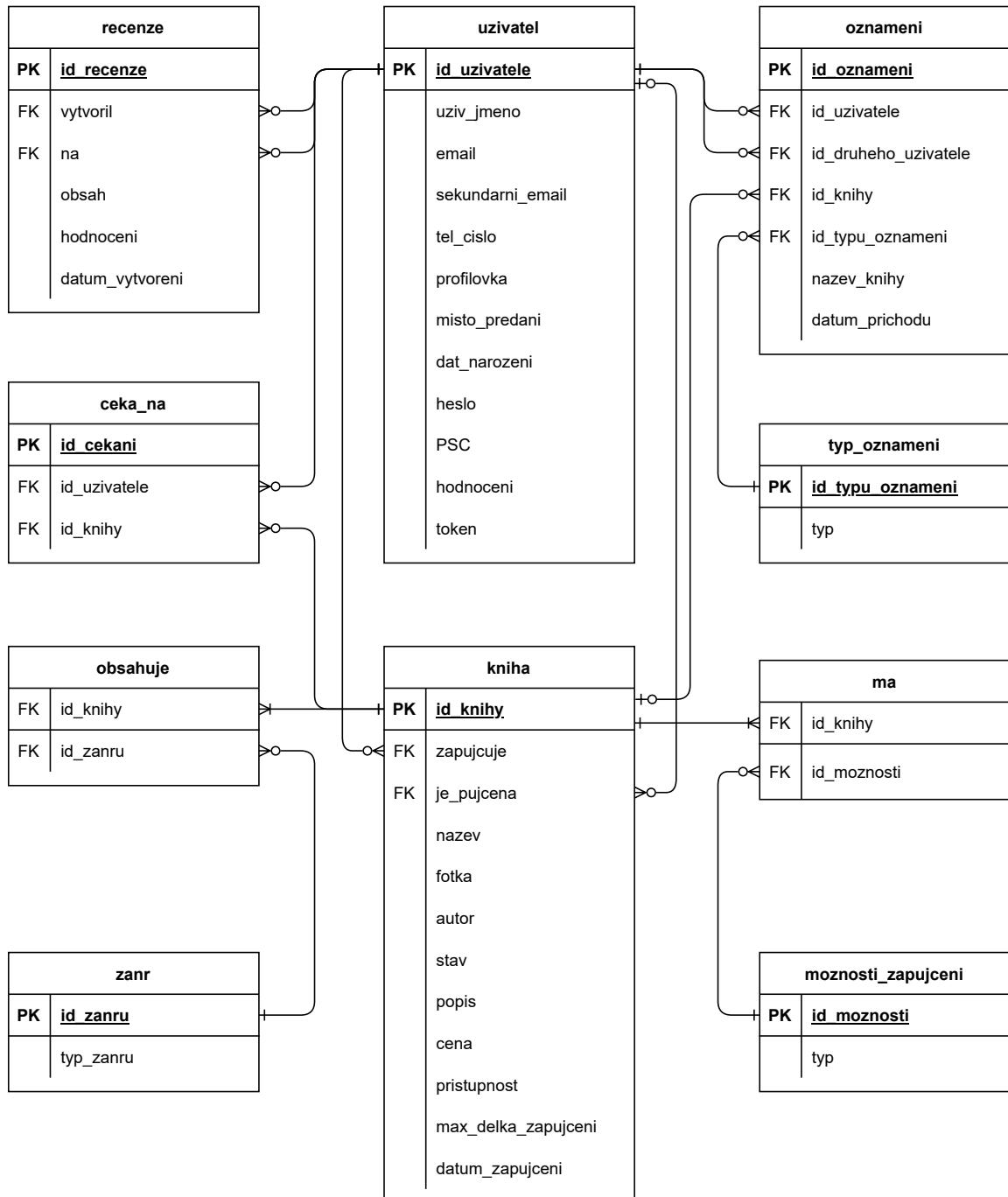
(b) Oznámení, část 2.

(c) Oznámení, část 3

Obrázek 3.9: Na prvním obrázku jsou vidět různá oznámení s interakcjemi. Na druhém a posledním obrázku jsou zobrazeny možnosti zobrazení potvrzených, nebo zamítnutých žádostí a také možnost hodnocení uživatele po úspěšném vypůjčení a vrácení knihy.

3.3 Návrh databáze

Jelikož aplikace pracuje s velkým množstvím dat, která musí být uchována a spravována, bylo nezbytné navrhnout vhodnou databázi. V následující podkapitole je představen ER diagram navržené databáze, který je zobrazen na obrázku 3.10.



Obrázek 3.10: Na obrázku lze vidět ER diagram navržené databáze vizualizující data a jejich propojení. Jsou zde zobrazeny primární klíče (PK) a cizí klíče (FK), které definují vztahy mezi tabulkami v databázi.

V této části je popsána každá entita v navržené databázi a její účel. Každá entita je zodpovědná za ukládání specifických informací, které jsou důležité pro správné fungování aplikace.

- **uzivatel:** Entita uchovává údaje jednotlivých registrovaných uživatelů aplikace, a to včetně registračního tokenu jejich zařízení.
- **recenze:** Tato entita slouží k ukládání recenzí na jednotlivé uživatele.
- **oznameni:** Tato entita ukládá všechna oznámení, která uživatelé dostávají. Součástí tento entity je také kniha, která s daným oznámením souvisí.
- **typ_oznameni:** V této entitě jsou uchovány všechny existující typy oznámení.
- **kniha:** Entita uchovává informace o knize, včetně informace, kdo knihu vytvořil a kdo ji má vypůjčenou.
- **ceka_na:** Tato entita slouží k uložení informace o tom, který uživatel čeká na kterou knihu, aby mohl daný uživatel dostat oznámení jakmile bude daná kniha volná.
- **obsahuje:** Toto je spojovací tabulka, která slouží k propojení knih a žánrů, které jsou ve vztahu M:N.
- **zahr:** Tato entita obsahuje všechny typy žánrů, kterých mohou knihy nabývat.
- **ma:** Toto je spojovací tabulka, která slouží k propojení knih a možností vypůjčení, které jsou ve vztahu M:N.
- **moznosti_zapujceni:** Tato entita obsahuje všechny možnosti zapůjčení, s kterými lze knihy zapůjčit.

3.4 Návrh API

API, neboli Application Programming Interface, je rozhraní, které se v případě komunikace mezi klienty a serverem, používá pro výměnu dat mezi těmito dvěma stranami. Pro vývoj mé aplikace je potřeba vytvořit server, který bude zajišťovat implementaci jednotlivých API endpointů, neboli v českém znění – koncových bodů. Každý endpoint bude pracovat s databází, jejíž návrh byl zmíněn výše. Návrh API rozhraní byl proveden pomocí webové aplikace SwaggerHub², která obsahuje vlastní textový editor s vlastní syntaxí pro navrhování API rozhraní. Aplikace také poskytuje přehlednou dokumentaci, která je generována právě z psaného kódu v textovém editoru. V následujících snímčích obrazovky 3.11 a 3.12 lze vidět danou dokumentaci, přesněji přehled všech endpointů s významem pro dvě skupiny: „uživatel“ a „kniha“.

uživatel Endpointy, které souvisí s uživatelem		
POST	/auth/register Registrace uživatele	✓ ↕
POST	/auth/login Přihlášení uživatele do aplikace	✓ ↕
POST	/user/{userId}/review Orecenzování uživatele v rámci oznámení	✓ ↕
GET	/user Získat informace o přihlášeném uživateli (můj profil)	✓ ↕
GET	/user/notifications Získat notifikace přihlášenému uživateli	✓ ↕
GET	/user/{userId} Získat informace o daném uživateli (profil)	✓ ↕
GET	/user/menuInfo Získat profilovou fotku (avatara), uživatelské jméno a e-mail přihlášeného uživatele do menu	✓ ↕
PUT	/user/changePassword Změnit heslo	✓ ↕
PUT	/user/contacts Změnit avatara, uživatelské jméno a mé kontaktní údaje	✓ ↕
DELETE	/user/deleteAccount Odstranit vlastní účet	✓ ↕
DELETE	/user/{notificationId} Odstranit notifikaci	✓ ↕

Obrázek 3.11: Dokumentace webové aplikace SwaggerHub, která zobrazuje všechny navržené endpointy pro skupinu „uživatel“, včetně významu každého endpointu a použitých HTTP metod.

²<https://app.swaggerhub.com/>

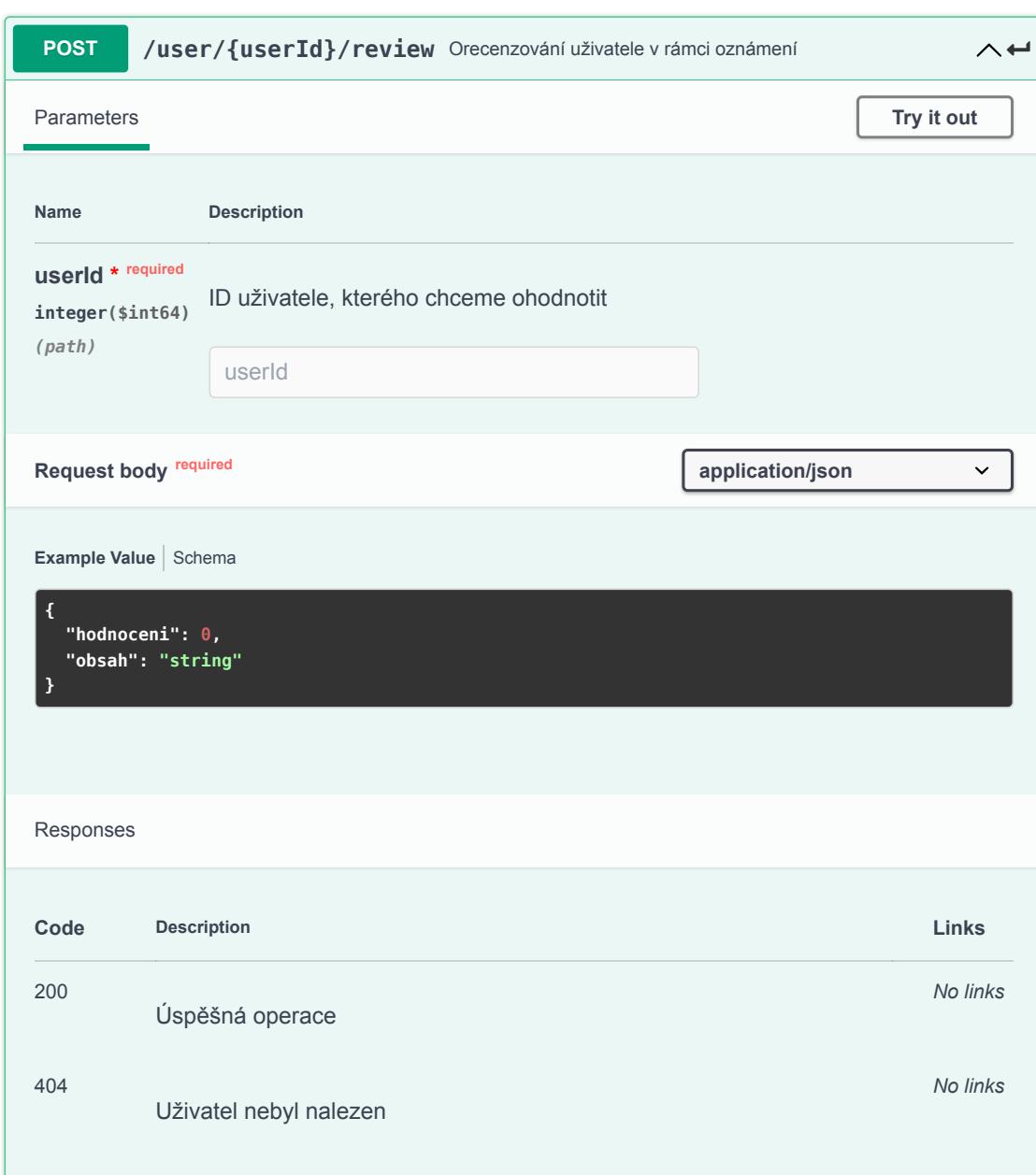
Poslední snímek obrazovky 3.13 ukazuje podrobnější informace o vybraném endpointu, který slouží pro vytváření recenzí uživatelů. Popis implementace jednotlivých endpointů na serveru je popsán v podkapitole 4.2.1

kniha Endpointy, které souvisí s knihami



POST	/book	Přidat novou knihu	
POST	/book/{bookId}/report	Nahlášení knihy kvůli nevhodnému obsahu	
POST	/book/{bookId}/borrow	Žádost o půjčku knihy, majitel přijde oznámení, ve kterém půjčku přijme/odmítne	
POST	/book/{bookId}/notifyMe	Žádost o oznámení, jakmile bude daná kniha volná	
POST	/book/{bookId}/answer	Potvrzení/zamítnutí žádosti o půjčku knihy	
POST	/book/{bookId}/returnLate	Potvrzení/zamítnutí vrácení knihy po vypršení max. délky výpůjčky	
POST	/book/{bookId}/returnSoon	Vrácení knihy do oběhu dříve s důvodem	
GET	/books	Získat knihy na hlavní straně (volné i již zapůjčené)	
GET	/books/filter	Získat knihy dle filtru	
GET	/books/{bookName}	Získat knihy dle zadанého názvu (vyhledávač)	
GET	/book/deliveryPoint	Získat vlastní místo předání při přidávání knihy (aby se zobrazilo v případě jeho zaškrtnutí v možnostech zapůjčení)	
GET	/book/{bookId}	Získat podrobnější informace o vybrané knize	
PUT	/book/{bookId}	Editovat vlastní knihu	
DELETE	/book/{bookId}	Odstanit vlastní knihu	
DELETE	/book/{bookId}/deleteRequest	Zrušení požadavku o půjčení knihy, nebo o oznámení	

Obrázek 3.12: Dokumentace webové aplikace SwaggerHub, která zobrazuje všechny navržené endpointy pro skupinu „kniha“, včetně významu každého endpointu a použitých HTTP metod.



POST /user/{userId}/review Orecenzování uživatele v rámci oznámení

Parameters

Name Description

userId * required
integer(\$int64)
(path) userId

Request body required application/json

Example Value | Schema

```
{
  "hodnoceni": 0,
  "obsah": "string"
}
```

Responses

Code	Description	Links
200	Úspěšná operace	No links
404	Uživatel nebyl nalezen	No links

Obrázek 3.13: Na snímku obrazovky lze vidět příklad zobrazení podrobností o vybraném endpointu. V tomto případě se jedná o endpoint pro vytvoření recenze na uživatele. Je zde povinný parametr `userId`, který se vyskytuje v cestě endpointu, dále je zde povinné tělo, které obsahuje obsah recenze a počet udělených hvězd v JSON formátu. Nakonec jsou zde zobrazeny možné druhy odpovědí, které mohou být odeslány klientovi po zpracování požadavku. Tyto odpovědi obsahují HTTP návratový kód a další detaily, ale mohou také obsahovat objekty převedené na JSON formát.

Kapitola 4

Implementace

Jelikož jsem při navrhování uživatelského rozhraní aplikace 3.2 postrádal zkušenosti s implementací mobilních aplikací, tak jsem se neostýchal použít mnohdy nereálné vymoženosti, jejichž změny jsou popsány v této kapitole. Kromě toho jsou zde popsány také podrobnosti o implementaci jak klienta, tak i serveru, včetně využitých technologií a knihoven.

4.1 Implementace aplikace BookSharing

Jako součást implementace mé aplikace jsem využil architekturu klient-server, což znamená, že existují dva hlavní aktéři: server a klient. Server může být umístěn na jakémkoli zařízení s připojením k internetu, zatímco klienti reprezentují mobilní zařízení uživatelů s operačním systémem Android. Implementace probíhala na mém vlastním lokálním počítači, což mi usnadnilo vývoj a ověřování funkčnosti aplikace.

Pro zajištění komunikace mezi serverem a databází jsem použil aplikaci XAMPP¹, která obsahuje webový server a databázi MySQL. Můj server komunikuje s daným databázovým serverem pomocí SQL dotazů, jako jsou SELECT, INSERT, DELETE a UPDATE. Databázi jsem vytvořil a posléze upravoval v programovém systému phpMyAdmin, což je nástroj napsaný v jazyce PHP, umožňující jednoduchou správu obsahu MySQL databáze prostřednictvím webového rozhraní. Při vytváření databáze jsem vycházel z definovaného návrhu 3.3.

Komunikace mezi klienty a serverem probíhá pomocí REST API², což je rozhraní, které je navrženo pro výměnu dat mezi klientem a serverem na základě REST architektury. Toto rozhraní používá standardní HTTP metody, jako jsou GET, POST, PUT, DELETE a komunikuje s klienty a servery pomocí HTTP protokolu. Při implementaci REST API na serveru, jsem vycházel z definovaného návrhu 3.4.

Kromě toho jsem pro odesílání notifikací na klientská zařízení využil Firebase Cloud Messaging. Konkrétně jsem použil Firebase Admin SDK, který umožňuje jednoduchou a bezpečnou komunikaci s Firebase Cloud Messaging. Díky tomu jsem mohl snadno ovládat, kterým klientským zařízením má být notifikace poslána a jaký obsah by měla notifikace mít. Všechny výše zmíněné technologie jsou zobrazeny na obrázku 4.1.

Na začátku mé implementace jsem postupoval podle návodů od Belal Khana, které mi poskytly základy nejen architektury MVVM pro klienta³, ale také autentizace uživatele na

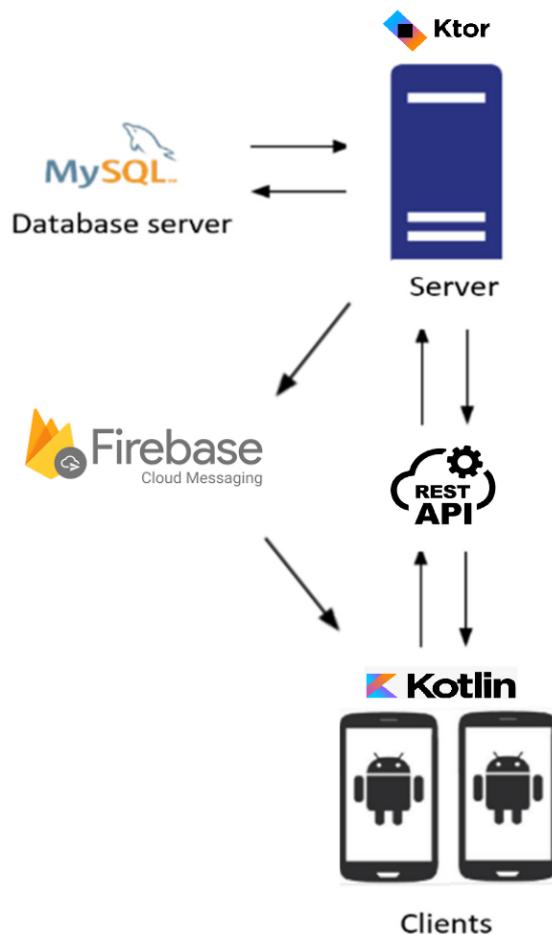
¹<https://www.apachefriends.org/>

²REST API – Representational State Transfer Application Programming Interface

³<https://github.com/probelalkhan/android-login-signup-tutorial>

serveru⁴ i klientovi. Inspirace od Khana mi umožnila rychleji pochopit a použít správné postupy v implementaci mé aplikace. Díky jeho návodům jsem byl schopen se naučit důležité techniky a použít je v mé aplikaci, což mi ušetřilo mnoho času a usnadnilo mi proces implementace. Při implementaci funkcionality notifikací na serveru⁵ i klientovi⁶ jsem se inspiroval kódem od Akashe Kamati, který poskytl užitečný návod pro implementaci podobného řešení. Kód jsem upravil a implementoval do mé aplikace s ohledem na specifické požadavky a potřeby.

Předtím, než jsem měl správně propojený server s klientem, jsem testoval implementované endpointy pomocí aplikace Postman. Tato aplikace umožňuje rychle a jednoduše otestovat jednotlivé API endpointy bez nutnosti implementovat kompletní klientskou aplikaci.



Obrázek 4.1: Obrázek zobrazuje architekturu klient-server, obohacenou o komunikaci serveru s MySQL databází, komunikaci mezi klientem a serverem skrz REST API a také o Firebase Cloud Messaging, který posílá notifikace klientským zařízením, na základě serverových požadavků.

⁴<https://github.com/probelalkhan/my-story-app-ktor-rest-api>

⁵<https://github.com/akash251/Push-Notification-Ktor-Server>

⁶<https://github.com/akash251/Push-Notification-In-Android>

4.2 Implementace serveru

Implementace serverové části probíhala ve vývojovém prostředí IntelliJ IDEA od společnosti JetBrains. Rozhodl jsem se implementovat server ve stejném jazyce jako klient a tím je Kotlin.

Ktor

Důležitou součástí při implementaci serveru bylo využití frameworku Ktor⁷, který slouží primárně pro zpracování HTTP požadavků od klienta a odesílání odpovědí zpět. Umožňuje také deserializaci/serializaci dat v JSON formátu a v neposlední řadě autentizaci uživatele pomocí JWT (JSON Web Token), což je standard pro vytváření a ověřování digitálních podpisů v rámci autentizace uživatele v aplikacích. Jedná se o formát tokenu, který obsahuje informace o uživateli a jeho oprávnění. Token se podepisuje klíčem serveru a je poté posílan v rámci HTTP požadavků jako způsob ověření identity uživatele na straně serveru.

Ktorm

Dále byla využita pro práci s databázemi v jazyce Kotlin open-source knihovna Ktorm⁸, která poskytuje snadné a intuitivní rozhraní pro vykonávání dotazů nad databází. Díky Ktormu je možné nadefinovat objekty entit, které reprezentují tabulky v databázi, a následně s nimi pracovat v kódu. Ktorm se zaměřuje na bezpečnost, přehlednost kódu a efektivitu. Navíc poskytuje podporu pro různé databázové systémy, jako jsou PostgreSQL, MySQL, SQLite a Oracle.

Úložný prostor

Server také funguje jako úložný prostor obrázků, mám tedy snadný přístup ke všem obrázkům na serveru a cesty k nim jsou ukládány do databáze. Profilové obrázky uživatelů jsou ukládány do složky pojmenované uživatelovým ID, a obrázky knih, které tito uživatelé přidali, jsou ukládány do vnořené složky *books*. Aby nedošlo k uložení obrázku s duplicitním jménem, je využíván čítač, který přidá k názvu obrázku následující číslo, které ještě neexistuje v dané složce. Cesta k obrázku knihy, přidaný uživatelem s ID 12 vypadá následovně:

images/12/books/book_image_4.jpg

Pro práci s obrázky mám vytvořené níže uvedené funkce:

- **encodeImageToBase64**: Tato funkce přijme cestu k obrázku, který je uložen v databázi (pokud existuje). Dále nalezne daný obrázek na serveru, zakóduje jej do Base64 a vrátí zakódovaný String, aby mohl být dále poslán klientovi v JSON odpovědi na nějaký požadavek. Klient dekóduje daný String a zobrazí obrázek uživateli.
- **decodeBookPhotoFromBase64AndStoreIt**: Tato funkce přijme obrázek knihy zakódovaný v Base64 a ID uživatele, který obrázek přidal. Funkce nejprve dekóduje String s obrázkem, vytvoří složku daného uživatele s jeho ID a vnořenou složku *books*, pokud ještě neexistují. Poté vytvoří unikátní název souboru a uloží ho do *books*. Nakonec vrátí String, který obsahuje cestu k obrázku a ten se uloží do databáze.

⁷<https://ktor.io/>

⁸<https://www.ktorm.org/>

- **decodeProfilePictureFromBase64AndStoreIt**: Tato funkce se liší od předchozí funkce v tom, že se ukládá profilový obrázek uživatele do složky pojmenované podle jeho ID.

Firebase Admin SDK

V poslední řadě jsem využil Firebase Admin SDK, což je knihovna pro vývojáře, kteří chtějí používat služby Firebase⁹ na svém serveru. Firebase Admin SDK umožňuje vývojářům spravovat a aktualizovat data, která jsou uložena ve Firebase cloudu, například aktualizovat databázi a nebo přidávat, či mazat uživatelské účty. Já jsem ho využil pouze pro vytváření a posílání notifikací uživatelům v reálném čase. Na to se využívá služba Firebase Cloud Messaging. Vytvořil jsem si tedy projekt na Firebase a stáhnul jsem si soubor `firebase_service_account.json`, který jsem si uložil na serveru. Dále jsem postupoval podle návodu od Akash Kamati, od kterého jsem převzal kód na inicializování Cloud Messaging na serveru. Pro posílání notifikací jsem si upravil pro mé potřeby následující funkci:

```
suspend fun sendNotification(
    notificationType: Int,
    db: Database,
    userId: Int
) : String
```

Ukázka kódu 1: V této funkci se nastaví název a tělo notifikace na základě typu notifikace – `notificationType`, poté se zjistí token patřící uživateli, kterému má notifikace přijít na základě jeho ID – `userId`. Nakonec se nastaví do notifikace i tento token, který je unikátní pro jednotlivá mobilní zařízení a na to zařízení se poté pošle notifikace. Funkce pak vrátí String hodnotu úspěšného či neúspěšného odeslání.

4.2.1 REST API

V této podsekci jsou popsány jednotlivé endpointy, které jsem implementoval. Všechny endpointy kromě přihlašování a registrace jsou obaleny v metodě `authenticate`. Tato metoda kontroluje, zda má uživatel, který poslal požadavek, oprávnění k požadovanému endpointu. Funguje to tak, že klient pošle při každém požadavku JWT, který v sobě obsahuje i ID přihlášeného uživatele a to je možné v kódu extrahovat a pracovat tak s ním. Jednotlivé endpointy jsou ve dvou skupinách, jedna se týká uživatelů a ta druhá knih, obě skupiny však mají i nějaké endpointy pro notifikace.

Co se týče notifikací, od návrhu oznamení v sekci 3.2 jsem se zde trochu odchýlil. Nyní existuje 8 typů oznámení na místo 7. Odstranil jsem oznamovací notifikaci, ve které majitel vidí, jakou knihu zamítl a komu ji zamítl. Namísto jedné interaktivní notifikace na hodnocení uživatele jsem udělal dvě, jedna pro hodnocení majitele a druhá pro hodnocení půjčujícího. Navíc jsem přidal ještě oznamovací notifikaci, která sděluje uživateli, že kniha, kterou chtěl, je nyní volná.

Uživatel

- **POST /auth/register**: Při registraci se nejprve získají vyplněné údaje od klienta v těle požadavku a uloží se do proměnné typu `User`, což je datová třída, která obsahuje

⁹<https://firebase.google.com/>

spoustu `Nullable` proměnných, takže se dá tato třída využívat častěji. Jeden z údajů je datum narození, který je ve `Stringu` a je potřeba ho převést na typ `LocalDate`, aby mohl být uložen do databáze, kde má typ `date`. Je zde také využita funkce `hash()` pro zašifrování hesla pomocí tajného klíče a šifrovacího algoritmu `HmacSHA1`. Nakonec využitím knihovny vložím data do databáze a vytvořím tak nového uživatele. V případě duplikátního e-mailu, telefonního čísla, nebo uživatelského jména je klientovi vrácen HTTP kód `409 Conflict`, v úspěšném případě `200 OK`.

- **POST /auth/login:** Dle údajů, které přijdou od klienta při přihlašování se zkонтroluje, zda takový uživatel existuje, a pokud ano, tak se mu vytvoří unikátní autentizační token – JWT, který si klient uloží v zařízení a bude ho posílat při každém požadavku na server. Také se zkонтroluje jestli má uživatel v databázi registrační token, který slouží k rozlišení mobilních zařízení, které komunikují se serverem. Pokud uživatel má jiný registrační token než je uložený v databázi, tak se aktualizuje. To znamená, že notifikace mu budou chodit na zařízení, na kterém se přihlásil naposledy.
- **POST /user/userId/review:** Na tento endpoint přijde požadavek, když uživatel hodnotí jiného uživatele v rámci oznámení typu `EVALUATE_BORROWER`, nebo `EVALUATE_OWNER`. Uloží se tedy do databáze nový záznam recenze s údaji, které přišli od klienta. Vypočítá se nové průměrné hodnocení uživatele a uloží do databáze. V poslední řadě se smaže to oznámení, z kterého uživatel hodnotil.
- **GET /user:** Tento endpoint pošle celý profil přihlášeného uživatele v rámci jedné odpovědi. Takový profil obsahuje kontakty, recenze, volné knihy, vypůjčené knihy a knihy k navrácení. Knihy k navrácení může uživatel vidět pouze na svém profilu.
- **GET /user/{userId}:** Stejně jako předchozí endpoint, s tím rozdílem, že na tento endpoint přijde požadavek, pokud se klient dostane v rámci aplikace na profil jiného uživatele. Pomocí příznaku `isUserPrivilegedToSeeContacts` je zde zajištěno, aby uživatel neměl přístup ke kontaktům druhého uživatele, pokud si mezi sebou momentálně nic nepůjčuje. Skrz tento endpoint se může uživatel dostat také svůj profil, což je zjištěno při porovnání ID uživatele v autentizačním tokenu a `userId` v parametru požadavku. V takovém případě se tento endpoint chová stejně jako ten předchozí.
- **GET /user/notifications:** Po zpracování tohoto požadavku dostane přihlášený uživatel všechna oznámení, která se ho týkají. Jsou uloženy do listu a jsou seřazeny dle data příchodu. Je nutno podotknout, že při oznámení pro hodnocení uživatele nemusí již existovat kniha, z které je toto oznámení vytvořeno. Proto si v databázi ukládám název knihy přímo do záznamu té notifikace.
- **GET /user/menuInfo:** Tento endpoint pouze posílá přihlášenému uživateli jeho profilový obrázek, uživatelské jméno a e-mail. Tato data jsou určena pro zobrazení v bočním menu.
- **PUT /user/contacts:** Při změně kontaktních údajů je poslán požadavek na tento endpoint. Pokud uživatel změní profilový obrázek, tak se ze serverového úložiště odstraní ten původní a to pomocí následující ukázky kódu s využitím knihovny `java.io.File`:
`val oldFile = File("${System.getProperty("user.dir")}/$oldPhotoPath")
oldFile.delete()`

Uživatel si nesmí změnit PSČ, jestliže si momentálně půjčuje nějakou knihu, která PSČ požaduje. Podobný případ platí i pro místo předání, které uživatel nesmí odstranit v případě, že nabízí knihu, která může být půjčena osobně na daném místě předání.

- **PUT /user/changePassword:** Uživatel pošle na tento endpoint požadavek při změně hesla. Nejprve se zašifruje původní heslo s využitím funkce `hash()`, porovná se s tím v databázi a pokud se rovnají, zašifruje se nové heslo a nahradí se to staré v databázi. V neúspěšném případě je klientovi zaslán HTTP kód 400 `Bad Request`.
- **DELETE /user/deleteAccount:** Aby si uživatel mohl odstranit svůj účet, je zde prvně zkонтrolováno, zda daný uživatel má vypůjčenou nějakou knihu od jiného uživatele, nebo nějakou někomu půjčil. Pokud ne, tak se odstraní z tabulky `uzivatel` v databázi, čímž se kaskádově odstraní všechny další záznamy spojené s tímto uživatelem. Nakonec se odstraní jeho obrázky ze serverového úložiště, a to rekurzivně pomocí funkce `deleteRecursively()` z knihovny `kotlin.io`.

Kniha

- **POST /book:** Tento endpoint slouží pro přidání knihy do databáze, včetně uložení obrázku na serverovém úložišti. Kromě vytvoření záznamu v tabulce `kniha`, je potřeba vytvořit záznamy ve spojovacích tabulkách `obsahuje` (žánr) a `ma` (možnosti zapůjčení). Pokud uživatel změní místo předání, aktualizuje se záznam v tabulce `uzivatel`, kde se vyskytuje a tím pádem se místo předání změní všude, kde jej uživatel používá.
- **POST /book/{bookId}/borrow:** Když si uživatel zažádá o výpůjčku knihy, pošle klient požadavek na tento endpoint. Pokud se kniha dá půjčit posláním na poštu, je potřeba aby daný uživatel měl nastavené PSČ v kontaktních údajích na profilu. Dále se vytvoří záznam ve spojovací tabulce `dostava` (oznámení) a odešle se oznámení ve funkci `sendNotification()` majiteli knihy. Jelikož je 8 typů oznámení, je třeba specifikovat typ na `BORROW_BOOK`. V tomto interaktivním oznámení majitel přijme, nebo odmítne požadavek.

Bylo potřeba však myslit i na jednu situaci, která může nastat. Pokud je kniha půjčená a majitel knihu vrátí, přijde oznámení typu `BOOK_IS_NOW_AVAILABLE` všem uživatelům, kteří si o to zažádali. V takovém případě může nastat tzv. Clickwar¹⁰, protože více uživatelů bude žádat o výpůjčku knihy a majitel vybere pouze jednoho z nich. Pokud se toto všechno stane v jeden moment, může se stát že uživatel bude žádat o již vypůjčenou knihu, protože se ještě neaktualizovalo grafické uživatelské rozhraní. V takovém případě bude uživateli zaslán HTTP kód 409 `Conflict`.

- **POST /book/{bookId}/notifyMe:** Prostřednictvím tohoto endpointu, se uživateli vytvoří záznam ve spojovací tabulce `ceka` (na notifikaci), ve které se uloží ID knihy i uživatele.
- **POST /book/{bookId}/answer:** Jakmile uživatel dostane interaktivní oznámení typu `BORROW_BOOK`, které mu sdělí, že si někdo chce půjčit jeho knihu, má dvě možnosti: přijmout, nebo odmítnout. Pokud majitel přijme nabídku, smaže se daná notifikace

¹⁰Clickwar je situace, kdy více uživatelů kliká na stejně tlačítko, a pouze omezené množství z nich může „vyhrát“.

a ještě navíc se smažou notifikace typu `BOOK_IS_NOW_AVAILABLE` na tuto knihu všem uživatelům. Jelikož zjišťuji, zda je kniha půjčená na základě sloupců `je_pujcena` a `datum_zapujceni` v tabulce `kniha`, tak se zde tyto sloupce vyplní. Nakonec se vytvoří notifikace typu `CONTACT_OWNER` a `CONTACT_BORROWER`, které se pošlou majiteli a půjčujícímu, aby se mohli navzájem kontaktovat. Majitelova druhá možnost, je uživateli odmítout nabídku na výpůjčku knihy. V takovém případě se opět smaže dané oznamení `BORROW_BOOK` a navíc se ještě vytvoří oznamení `OWNER_DECLINED_BORROW`, které se pošle danému zájemci o knihu a informuje ho o odmítnutí.

- **POST /book/{bookId}/returnSoon:** Tento endpoint slouží k vrácení knihy, před tím než vyprší maximální doba výpůjčky pro danou knihu. Pokud však uživatel pošle požadavek na tento endpoint poté, co doba výpůjčky vypršela, dostane HTTP kód 403 `Forbidden` s informací, že knihu musí vrátit v notifikacích, kde pošle požadavek na endpoint `returnLate`. Nyní lze vrátit knihu tím, že se nastaví `null` ve sloupcích `je_pujcena` a `datum_zapujceni` v tabulce `kniha` pro danou knihu. Dále se smažou notifikace typu `CONTACT_OWNER` a `CONTACT_BORROWER`, protože od této doby už tito uživatelé neuvidí navzájem své kontakty. Majitel má na výběr ze dvou možností při evidování vrácení knihy: kniha byla vrácena brzy, kniha nebyla vůbec vypůjčena. Pokud kniha byla vrácena dříve, vytvoří se dvě oznamení typu `EVALUATE_BORROWER` a `EVALUATE_OWNER` a pošlou se daným uživatelům, kteří se díky témtu notifikacím mohou navzájem ohodnotit. Pokud však kniha nebyla vůbec vypůjčena, bude moci hodnotit pouze majitel knihy půjčujícího. V tomto případě se vytvoří pouze oznamení typu `EVALUATE_BORROWER` a pošle se majiteli. Jsem toho názoru, že pokud kniha nebude vůbec vypůjčena, je s větší pravděpodobností chyba na straně půjčujícího, ale není stoprocentní, tudíž toto nemusí být nejideálnějším řešením této problematiky. Jelikož je kniha nyní opět volná, vytvoří se oznamení typu `BOOK_IS_NOW_AVAILABLE` a pošle se všem uživatelům, kteří si tuto knihu přáli v rámci endpointu `notifyMe`.
- **POST /book/{bookId}/returnLate:** Uživatel má dvě možnosti v interaktivní notifikaci typu `TIME_LIMIT_EXPIRED` a ty jsou: vráceno, nebo nevráceno. V tomto oznamení je majitel knihy vyzván k evidování vrácení knihy poté, co vypršela maximální délka výpůjčky dané knihy. Zvolením jedné ze dvou možností se pošle požadavek na tento endpoint a daná notifikace se smaže. V případě nevrácení knihy se ve skutečnosti smaže daná kniha z tabulky `kniha` a tím pádem se kaskádově smažou všechny další záznamy v databázi spojené s touto knihou, včetně oznamení a také obrázek knihy ze serverového úložiště. Dle mého názoru nemá smysl udržovat v databázi knihu, která nebyla majiteli vrácena a v případě pozdějšího vrácení by ji majitel musel znova přidat do aplikace. Dále se vytvoří pouze jedno oznamení, a to typu `EVALUATE_BORROWER`, které se pošle majiteli, aby ohodnotil půjčujícího, protože stejně jako v předchozím endpointu, je větší pravděpodobnost, že byla chyba na straně půjčujícího. Pokud majitel potvrdí vrácení knihy, proběhnou stejné akce jako v předchozím endpointu při brzkém vrácení knihy. S tím rozdílem, že se tu navíc ještě smaže daná notifikace, ze které se posílá požadavek na tento endpoint.
- **GET /books:** V tomto endpointu se nejdříve zavolá funkce `getDateOfBirth()`, která zjistí datum narození přihlášeného uživatele a dále vrátí uživateli list všech knih, ke kterým má přístup, na základě jeho věku.

Protože tento endpoint je nejčastěji využívaný, rozhodl jsem se v tomto endpointu kontrolovat, zda vypůjčeným knihám vypršela maximální doba výpůjčky. Všechny

knihy, kterým vypršel čas, jsou vloženy do listu. Prvně kontroluji pro každou knihu v listu, zda vypršela maximální doba výpůjčky + 14 dní navíc, které slouží jako rezerva pro pozdější vrácení knihy. Pokud tento čas vypršel, tak se odstraní kniha z databáze z tabulky `knihy` a také se kaskádově smažou všechny další záznamy v databázi spojené s touto knihou, včetně obrázku knihy ze serverového úložiště. V tomto případě je chyba na straně majitele, ale klidně i na straně půjčujícího, proto ani jeden uživatel nedostane oznámení na vzájemné ohodnocení. Teprve poté se zkонтroluje, zda vypršel čas maximální doby výpůjčky a pokud ano, vytvoří se a pošle se notifikace typu `TIME_LIMIT_EXPIRED` majiteli, který bude moci v rámci ohodnocení evidovat vrácení, nebo nevrácení dané knihy.

- **GET /books/{bookName}**: Na tento endpoint se pošle požadavek, když uživatel použije vyhledávač na vyhledání knihy podle nějakého řetězce. S využitím knihovny Ktorm, se v klauzuli `where` jednoduše využije klícové slovo `like`, které umožňuje porovnat hledaný řetězec s názvem knihy.
- **GET /books/filter**: Podobné jako předchozí endpoint, zde je však už složitější dotaz na databázi, protože uživatel má možnost vyfiltrovat až 5 podmínek na knihách. Vynechal jsem možnost filtrování dle přístupnosti, jak jsem navrhl na obrázku 3.5b, protože jsem došel k tomu závěru, že když už uživatel má předem vyfiltrované knihy dle jeho věku a navíc jsou v žánrech položky jako `Literatura pro děti a mládež` a `Erotika`, tak není potřeba filtrovat i přístupnost.
- **GET /book/deliveryPoint**: Když chce uživatel přidat knihu, pošle požadavek na tento endpoint, který mu pošle v odpovědi jeho místo pro předání, které má definované v kontaktních údajích. To je z toho důvodu, aby tuto informaci nemusel hledat, navíc je místo předání pro všechny jeho půjčované knihy stejné.
- **GET /book/{bookId}**: V tomto endpointu se klientovi pošlou v odpovědi informace o dané knize, kterou si chce prohlédnout. Mimo to jsou mu poslány také 3 příznaky. První je `myBook`, který reprezentuje knihu, kterou uživatel má půjčenou, nebo ji půjčuje a pokud je `true`, tak uživatel nesmí vidět tlačítko na vypůjčení nebo na oznámení, jakmile bude volná. Další příznaky jsou `requesting` a `requestingNotification`, které v případě `true` zaručí, že uživatel má možnost zrušit žádost, kterou na danou knihu podal.
- **PUT /book/{bookId}**: Tento endpoint slouží k editaci knihy, která uživateli patří, to znamená že se zde provádějí téměř stejně akce jako v endpointu `POST /book`. Při aktualizaci obrázku se však odstraní původní obrázek ze serverového úložiště. Co se týče spojovacích tabulek pro žánry a možnosti zapůjčení, tak zde je potřeba smazat původní záznamy a vytvořit nové, to je z toho důvodu, že spojovací tabulky jsou vytvořeny kvůli vztahu M:N mezi knihami a žánry/možnostmi zapůjčení, a proto nelze záznamy pouze aktualizovat.
- **DELETE /book/{bookId}**: Požadavek na tento endpoint se pošle, když klient chce odstranit svou knihu z databáze. provede se tak a navíc se smaže také obrázek knihy ze serverového úložiště.
- **DELETE /book/{bookId}/deleteRequest**: Uživatel, který žádá o vypůjčení nějaké knihy, nebo pouze o to, aby byl notifikovaný, jakmile bude kniha volná, má možnost tuto žádost zrušit a v takovém případě pošle požadavek na tento endpoint. Zde se

podle typu žádosti smaže buď notifikace typu `BORROW_BOOK` majiteli, nebo se smaže záznam ve spojovací tabulce `čeka` (na notifikaci).

4.3 Implementace klienta

Implementace klientské části probíhala v integrovaném vývojovém prostředí Android Studio, kde jsem měl na výběr mezi dvěma jazyky: Java a Kotlin. Vybral jsem si Kotlin, jednak z toho důvodu, že tento jazyk je oficiálním jazykem pro vývoj mobilních aplikací a jednak proto, že s Javou už jsem zkušenosti měl a chtěl jsem se naučit další jazyk, který je si s Javou navíc hodně podobný. Blížší rozdíly mezi těmito jazyky jsou popsány v kapitole 2.1.1.

Při implementaci jsem zároveň aplikaci testoval, a to zpočátku na mé vlastním mobilním zařízení Xiaomi Redmi Note 8 Pro, které má operační systém Android 11. Při implementaci notifikací jsem potřeboval druhé zařízení, abych v reálném čase viděl, jak spolu dva uživatelé interagují. Na to jsem využil, v rámci Android studia, emulátor mobilního zařízení Pixel 6 s operačním systémem Android 13.

4.3.1 MVVM architektura

Pro architekturu mé aplikace jsem jsem si vybíral mezi MVVM (Model-View-ViewModel) a MVI (Model-View- Intent), ale kvůli mému nedostatečnému povědomí o obou těchto architekturách jsem se rozhodl pro MVVM. Důvodem byly také mé zkušenosti s MVC (Model -View - Controller), který je si s MVVM docela podobný. MVVM architektura se mi také líbí pro oddělení datové a UI vrstvy a také pro snadnou testovatelnost díky ViewModel vrstvě. Tyto dvě architektury jsou blíže popsány v kapitole 2.4. Dále jsem si vybíral mezi XML a Jetpack Compose pro implementaci View, tedy uživatelského rozhraní v MVVM architektuře. Tyto dvě možnosti jsou popsány v kapitole 2.2. I přesto, že Jetpack Compose nabízí více výhod a je modernější, přišlo mi XML jako jednodušší volba pro začátečníka. Avšak plánuji se v budoucnu naučit Jetpack Compose a přejít na tuto moderní technologii.

Správa dat a logika akcí v architektuře MVVM

Samotný ViewModel slouží k oddělení business logiky od uživatelského rozhraní a správy dat. ViewModel obsahuje různé metody a LiveData objekty pro správu dat a jejich interakce s uživatelským rozhraním. Důležitou součástí architektury MVVM je také repozitář, který slouží k oddělení zdrojů dat od ViewModełu. Repozitář může například obsahovat metodu pro získání dat ze vzdáleného serveru. ViewModel může poté volat tyto metody a získat potřebná data. Během získávání dat může být nastavena hodnota LiveData objektu na `Resource.Loading`, aby uživatel věděl, že se operace provádí. Po dokončení operace se výsledek vrátí zpět do ViewModełu a může být uložen do příslušného LiveData objektu. Uživatelské rozhraní (View) je poté informováno o změně v hodnotě tohoto objektu a může provést odpovídající akce, například zobrazení výsledku operace uživateli nebo zobrazení chybového hlášení.

4.3.2 Použité technologie

Při vývoji byla použit řada knihoven, mezi nejdůležitější patří `retrofit` pro komunikaci přes API rozhraní, dále `Firebase Messaging` pro přijímání notifikací ze serveru a jejich zobrazování uživatelům. S tím souvisí také `WorkManager`, který slouží k tomu, aby se notifikace

zobrazily pouze tehdy, když jsou data k dispozici a když je k dispozici síťové připojení. Pokud byla notifikace přijata, když zařízení nebylo připojeno k internetu, `WorkManager` může počkat, až bude připojení k internetu opět k dispozici a poté spustit práci (zobrazit notifikaci). V poslední řade jsem využil knihovnu `Datastore`, která slouží pro ukládání malých dat na zařízení.

4.3.3 Navigace

Aplikace používá navigační graf v XML souboru `nav_home`, kde jsou definovány cesty a fragmenty, které uživatel může navštívit. Tento navigační graf obsahuje informace o tom, které fragmenty jsou dostupné a jak jsou propojené. Pro navigaci se využívá třída `NavController`, která je inicializována v `HomeActivity` v metodě `onCreate`. Pomocí třídy `NavigationUI` je pak nastavena správa akcí v horní liště a menu, které mají ovládat navigaci v aplikaci.

Na hlavním fragmentu `HomeFragment` se zase využívá třída `NavigationView`, která umožňuje uživateli navigovat mezi fragmenty prostřednictvím menu. Po stisknutí položky v menu se spouští odpovídající akce, budě se zavolá metoda `logout` pro odhlášení uživatele z aplikace, nebo pomocí třídy `NavigationUI` přesměruje uživatele na příslušný fragment.

Ve fragmentech `ChangePasswordFragment` a `DeleteAccountFragment` se také využívá `ActionBarDrawerToggle`, který umožňuje zobrazení vysouvacího menu po stisknutí ikony hamburgeru v horní liště. V těchto fragmentech se také využívají rozhraní `MenuProvider` pro manipulaci s menu.

4.3.4 Uživatelské rozhraní

Při implementaci uživatelského rozhraní jsem vycházel z mého návrhu v podkapitole 3.2. V této podsekci jsou popsány jednotlivé sekce aplikace stejně jako v návrhu, ale více podrobněji a také jsou zde popsány rozdíly oproti návrhu, ke kterým při implementaci došlo. Základní informace o aktivitách a fragmentech, které v této podkapitole hrají velkou roli, lze najít v podsekcích 2.1.2.

Přihlašování a registrace

Zde byla využita knihovna `Datastore` pro uložení autentizačního tokenu, který uživatel obdrží po přihlášení a který posílá při každém požadavku na server. Protože je autentizační token uložen v mobilním zařízení po celou dobu, dokud se uživatel neodhlásí, tak není nutné se přihlašovat pokaždé když uživatel zapne aplikaci, ale uživatel je rovnou přesměrován na úvodní stranu. Je to také důvodem, proč jsem nepřidával checkbox pro zapamatování uživatele, jak jsem původně navrhoval. Při odhlášení uživatele se zavolá funkce `logout`, která odstraní autentizační token uživatele z `Datastoru` a přesměruje ho zpět do `AuthActivity`.

`AuthActivity` je aktivita, do které se dostane každý nový uživatel po prvním spuštění aplikace na jeho zařízení. Proto se zde nachází funkce `checkNotificationPermission`, která zajistí, aby se uživateli objevilo okno, ve kterém potvrdí oprávnění na notifikace pro tuto aplikaci. To se však týká pouze uživatelů s mobilním zařízením, který má verzi Android 13+. Daná aktivita obsahuje také dva fragmenty, jeden pro přihlašování a jeden pro registraci, mezi kterými lze snadno přepínat díky navigačnímu ovladači `NavController`, který je definován v aktivitě. `RegisterFragment` slouží pro vyplnění registračních údajů. Po stisknutí tlačítka na registraci se nejprve provede validace vstupů pomocí funkce `validateInput` a poté se pošle registrační požadavek na server, který pošle odpověď zpět. V případě úspěchu bude uživatel přesměrován na `LoginFragment`, kde se může přihlásit. Tyto dva

fragmenty a ještě `ChangePasswordFragment` jsou jedinými v celé aplikaci, které používají komponenty `TextInputLayout` a `TextInputEditText` pro získávání údajů. Tyto komponenty šetří místo na obrazovce kvůli absenci nadpisů nad jednotlivými polí, místo toho jsou obsaženy přímo v polích a po editaci jsou zmenšeny a umístěny nad psaný text.

Prohlížení knih a filtrování

Podoba úvodní strany (obrázek 4.2a) je zcela věrná mému návrhu 3.2. Úvodní strana je implementována v `HomeFragment`, kde je využita komponenta `RecyclerView`, která slouží jako kontejner pro jednotlivé `CardView`, které reprezentují karty jednotlivých knih. Pro nastavení jednotlivých karet využívám adaptér definovaný v souboru `BookListAdapter`. Tento adaptér má za úkol definovat, jak bude každá karta vypadat a také zpracovávat stisknutí na jednotlivé karty. Dále je ve fragmentu inicializováno menu, které je zobrazeno na obrázku 4.2c a které oproti návrhu 3.2 obsahuje navíc položku Odstranit účet, ale neobsahuje již Moje knihy, více informací o této změně se nachází v podkapitole 4.4. Přidal jsem do menu ještě hlavičku, která obsahuje e-mail a jméno přihlášeného uživatele a také jeho fotku, která je zobrazena v kulatém tvaru pomocí komponenty `CircleImageView`¹¹. Toto je jediný fragment, který má podporu obnovení stránky po přejetí prstem dolů. Komponenta `SearchView` slouží jako vyhledávač, na který je nastaven posluchač, neboli listener, který zavolá `ViewModel` metodu na získání knih s hledanou frází obsaženou v názvu knih. Listener reaguje na enter pro potvrzení vyhledávání, nebo na vyprázdnění vyhledávače, což opět ukáže všechny knihy. K tomu slouží funkce `onQueryTextChange`, kde kontroluje zda je prázdný vyhledávač. Docházelo zde k problému, že po procházení aplikací a následnému návratu na úvodní stranu se tato funkce zavolala, čímž se zavolali všechny knihy a smazali se tak všechny filtry. Tento problém jsem vyřešil pomocí nastavování příznaku, který se mění v této funkci a také ve funkci fragmentu `onPause()`, která se zavolá pokaždé když uživatel odejde z `HomeFragmentu`.

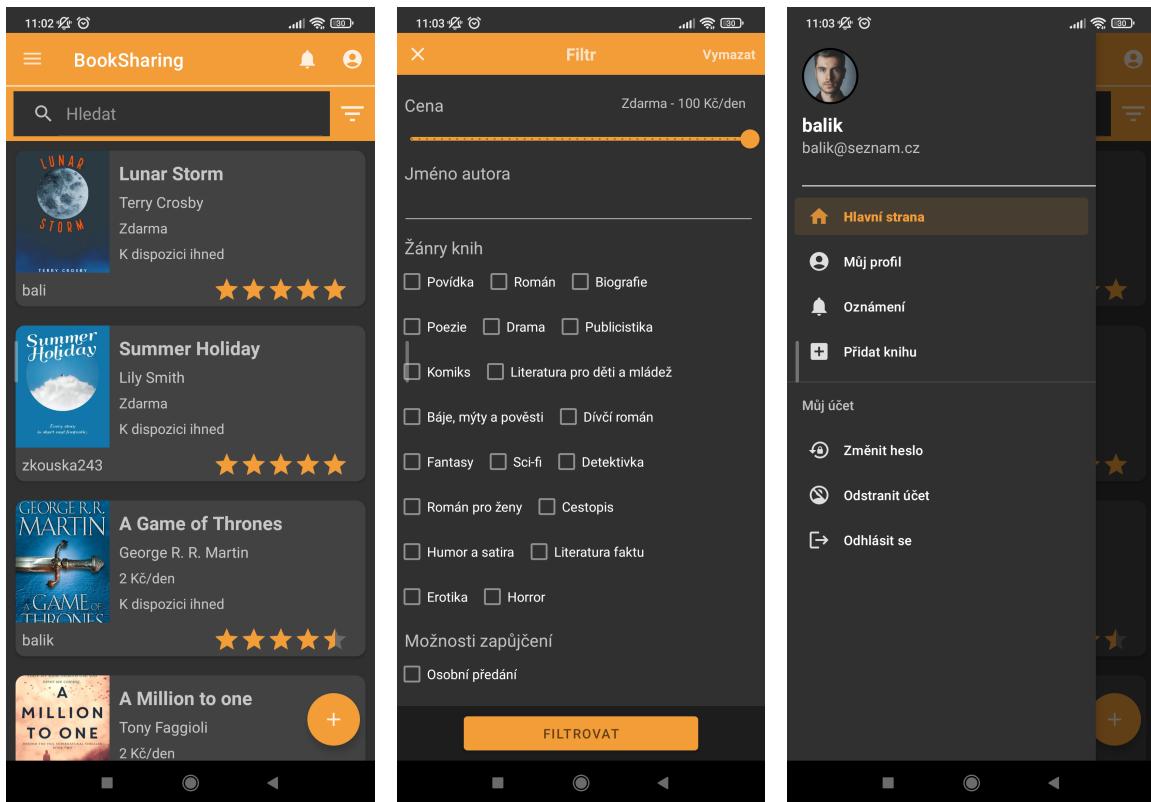
Uživatel se může přesměrovat na `FilterFragment` pomocí tlačítka vedle vyhledávače, kde jsem pozměnil následující věci oproti původnímu návrhu 3.2:

- Filtr již neobsahuje přístupnost, dospělý uživatel uvidí vždy všechny knihy a neplnoletý uvidí pouze knihy pro všechny.
- Žánry ve filtru jsou nyní poskládány tak, aby se jich vešlo co nejvíce na malý prostor.
- Cena má pouze jedno posuvné tlačítko na místo dvou a začíná na konci, protože uživatel by měl zpočátku vidět všechny knihy, i ty placené.

Tento fragment, jak lze vidět na obrázku 4.2b, neobsahuje stejnou horní lištu jako ostatní fragmenty, namísto toho je zde schována a vytvořena nová, která odpovídá návrhu 3.2. Po odchodu z fragmentu se opět zobrazí původní lišta pomocí funkce `onDestroyView()`.

Při přecházení mezi fragmenty `HomeFragment` a `FilterFragment` využívám tzv. arguments, ve kterých jsou uloženy nastavení filtru. Jelikož lze předávat mezi fragmenty pouze jednoduché datové typy, musím například převést list vybraných žánrů na String a oddělit je od sebe tečkou.

¹¹<https://github.com/hdodenhof/CircleImageView>



(a) Úvodní strana

(b) Filtrování

(c) Menu

Obrázek 4.2: Na prvním snímku obrazovky se nachází úvodní strana, na které se mimo vyhledávač a tlačítka filtru nachází nabízené knihy, volné i vypůjčené, seřazené sestupně od nejlepšího hodnocení po nejhorší s příznakem volné, a poté s příznakem vypůjčené. Na druhém snímku lze vidět různé typy filtrů, pomocí kterých si uživatel může zobrazit na úvodní straně pouze knihy s vybranými parametry. Na posledním snímku je menu aplikace, které umožňuje navigovat uživatele do všech částí aplikace

Informace o knize

Po stisknutí na nějakou knihu na úvodní straně se uživatel dostane na `BookDetailsFragment`, kde jsou zobrazeny všechny důležité informace o knize, které uživatel potřebuje vědět. Uživatelské rozhraní je téměř v souladu s již dříve definovaným návrhem 3.2 s tím rozdílem, že zde není zobrazena mapa, ale je zde pouze tlačítko, které uživatele přesměruje na Google mapy, tedy v případě, že je má nainstalované. Žánry, které kniha obsahuje, jsou zobrazeny ve `FlowLayout`¹². Tato komponenta zobrazuje orámované žánry takovým způsobem, aby zaplnila co nejméně řádků v závislosti na šířce obrazovky.

V případě, že uživatel má danou knihu vypůjčenou, nebo ji sám vypůjčuje, neuvidí v tomto fragmentu žádná tlačítka. V dalším případě kdy je kniha volná k vypůjčení, může uživatel vidět tlačítko „Požádat“ v dolní části obrazovky, nebo tlačítko „Upozornit mě“, pokud je kniha momentálně vypůjčená. Jakmile uživatel jedno z těchto tlačítek stiskne, pošle se požadavek na server, způsobí to zamknutí tlačítka a objeví se navíc tlačítko „Zrušit“ pro zrušení dané žádosti. K tomu slouží jedna z mých vytvořených funkcí `switchButtons()`.

¹²<https://github.com/nex3z/FlowLayout>

Pokud se uživatel rozhodne zrušit žádost o výpůjčku, či o oznamení a stiskne tedy tlačítko „Zrušit“, zavolá se ViewModel metoda pro zrušení požadavku a funkce `unSwitchButtons()`, která vrátí tlačítka do původního stavu.

Přidání i editování knihy

Zpočátku jsem vytvořil fragment pro přidávání knih zvaný `AddBookFragment`, přičemž později jsem tento fragment rozšířil o funkcionalitu na editování vybrané knihy. Z toho důvodu jsem tento fragment přejmenoval na `AddOrEditBookFragment`. Tento fragment se od mého definovaného návrhu 3.2 vůbec neliší. Pokud se v `arguments` nachází hodnota `bookId`, znamená to že uživatel přišel na tento fragment s účelem editovat knihu. Zavolá se tedy ViewModel metoda `getBookInfo(bookId)`, která vrátí všechny informace o dané knize a vyplní všechna pole odpovídajícími daty pomocí funkce `updateUI()`. Při přidávání fotografie dané knihy se zavolá funkce `openImagePicker()`, která uživateli umožní najít a vybrat obrázek knihy v jeho galerii. Stejně jako ve `FilterFragment`, jsou zde žánry rozloženy tak, aby vyplnily co nejvíce volného prostoru. Stejně jako v každém jiném formuláři, zde kontrolují vstup uživatele pomocí funkce `validateInput()`.

Změna hesla a Odstranění účtu

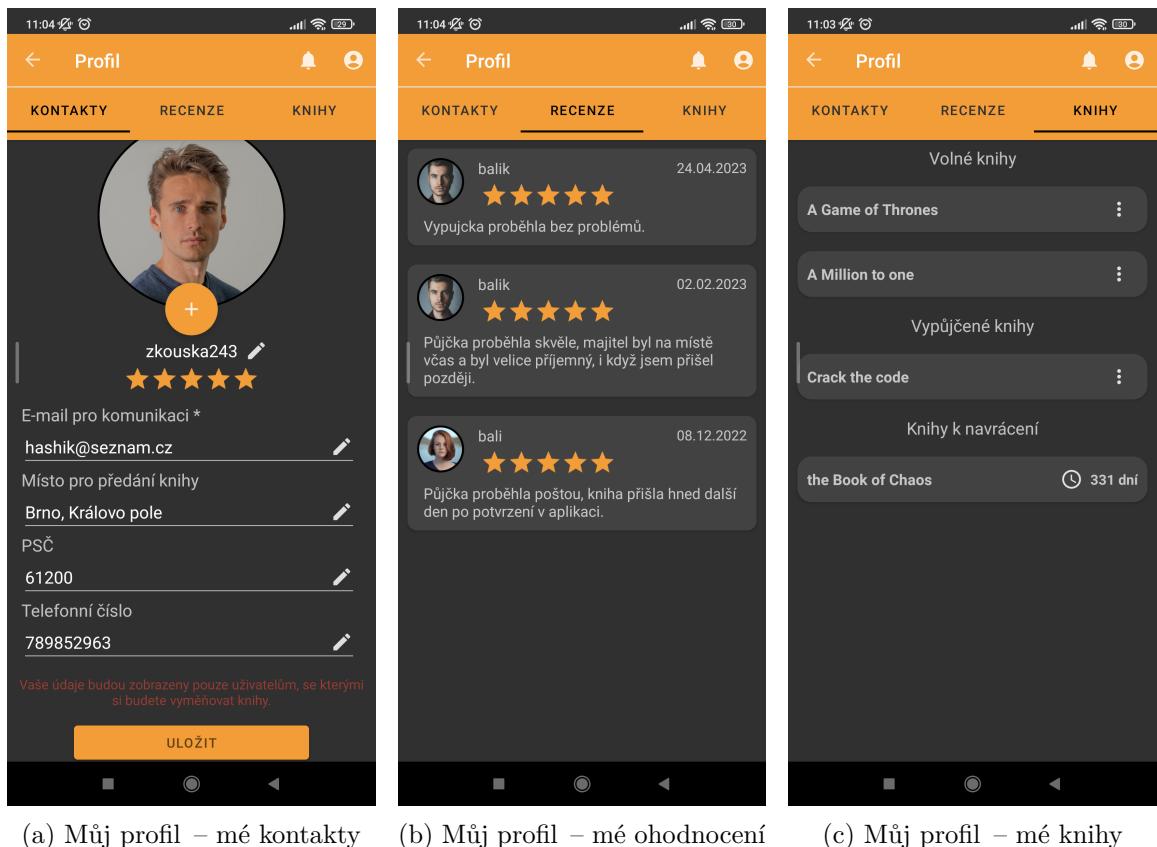
Pro změnu hesla jsem vytvořil fragment pojmenovaný `ChangePasswordFragment`, ve kterém se nachází 3 vyplňovací pole. Uživatel zde vyplní aktuální heslo a poté dvakrát nové heslo. Tlačítko pro změnu hesla je zamknuté a odemkne se až po vyplnění všech polí. Vyplněná pole jsou po stisknutí tlačítka zkонтrolována funkcí `validateInput()`. Během implementace klienta jsem se rozhodl ještě přidat `DeleteAccountFragment`, kde má uživatel možnost odstranit jeho účet. Je zde informován o tom, že účet lze odstranit jen tehdy, kdy nepůjčuje žádnou knihu, a ani nemá žádnou vypůjčenou. Při stisknutí tlačítka na odstranění účtu vyskočí dialogové okno, aby si uživatel případně odstranění účtu ještě rozmýšlel. Dialogové okno je vytvořeno pomocí funkce `PopupWindow` a zobrazeno pomocí funkce `showAtLocation`. Pokud odstranění potvrdí, pošle se požadavek na server skrz ViewModel metodu `deleteAccount()` a v případě úspěchu se zavolá funkce `logout()`, aby se odstranil autentizační token ze zařízení.

Profil uživatele

Uživatel se může dostat na svůj profil, nebo profil jiného uživatele z více míst v aplikaci. Uživatel je nasměrován na `ProfileFragment`, přičemž jestli se uživatel dostal na vlastní, nebo cizí profil je zjištěno pomocí Boolean příznaku `clickedOnLoggedInUser`, který přišel ze serveru v rámci odpovědi na požadavek pro získání profilových dat. Na základě tohoto příznaku se změní grafické uživatelské rozhraní v takovém smyslu, že přihlášený uživatel může na vlastním profilu aktualizovat své kontakty, nebo provádět interakce s vlastními knihami, přičemž na cizím profilu takové věci dělat nemůže. `ProfileFragment` obsahuje komponenty `TabLayout` a `ViewPager`, což jsou dva základní komponenty v Androidu, které umožňují vytvářet uživatelská rozhraní s více záložkami. Kombinace těchto dvou prvků vytváří interaktivní rozhraní s možností přepínání mezi různými fragmenty. Adaptér `ProfilePagerAdapter`, který je vytvořen v `ProfileFragment`, umožňuje zobrazovat různé fragmenty na každé stránce `ViewPageru`. V tomto konkrétním případě je nastaven na tři fragmenty, které odpovídají třem záložkám v `TabLayoutu`: Kontakty, Recenze a Knihy. Na `ProfileFragment` jsou stažena ze serveru data, která naplní celý profil a to tak, že jsou předána

adaptéru, který vytvoří instance tří fragmentů a předá každé instanci vybraná data do **arguments**.

Další důležitý Boolean příznak, který přijde ze serveru se nazývá **isUserPrivileged**, na jehož základě jsou schovány, nebo naopak zobrazeny kontaktní údaje uživatele v **ContactsFragment** (obrázek 4.3a). Za předpokladu, že si přihlášený uživatel nepůjčuje žádnou knihu s daným uživatelem, na jehož profilu se nachází, tak jsou jeho kontaktní údaje schovány a je upozorněn na to, že pro jejich zobrazení musí mít s tímto uživatelem vypůjčenou knihu. Oproti dříve definovanému návrhu 3.2 se nachází fotografie, uživatelské jméno a hodnocení uživatele pouze v tomto fragmentu. K této změně došlo z toho důvodu, že pro přihlášeného uživatele se změna jeho údajů musí potvrdit tlačítkem „Uložit“, který se nachází pouze na tomto fragmentu a také z toho důvodu, že ušetří místo na zbylých fragmentech. Tato změna poté vedla ke změně pořadí jednotlivých záložek. **TabLayout** začíná na první záložce, a proto jsem se rozhodl dát na první místo kontakty, protože uživatel bude chtít v první řadě vidět zda stiskl na správného uživatele a nebo bude chtít vidět jeho hodnocení.



Obrázek 4.3: Na prvním snímku obrazovky lze vidět kontakty uživatele, které si může upravit dle uvázení. Mimo jiné lze v této záložce změnit profilovou fotku a uživatelské jméno. Na druhém snímku jsou vidět hodnocení a recenze udělené přihlášenému uživateli a na třetím snímku jsou zobrazené knihy, které přihlášený uživatel půjčuje, rozdělené na volné a vypůjčené. Navíc se zde uživateli zobrazí i knihy, které má od někoho vypůjčené s časem vypršení výpůjčky.

V záložce Recenze na fragmentu **ReviewsFragment**, jež je zobrazen na obrázku 4.3b, uživatel vidí karty (komponenta **CardView**) jednotlivých recenzí, které byly na daného uživatele

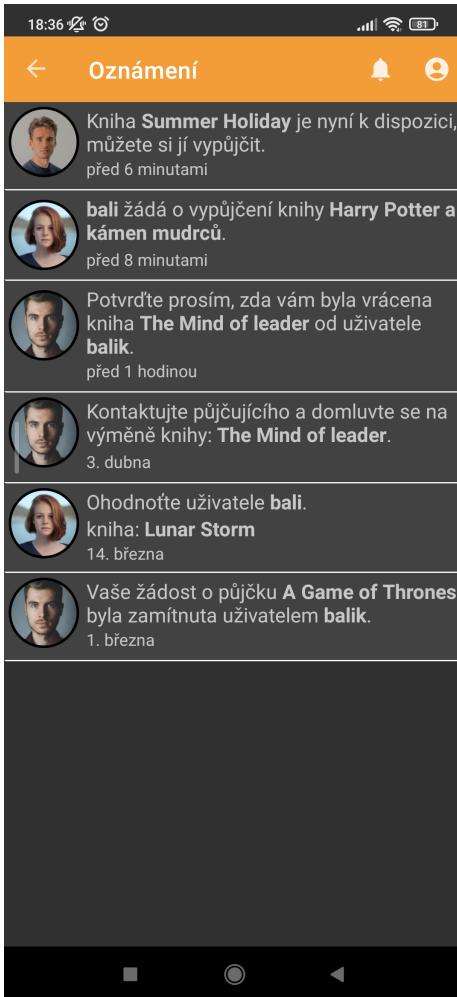
vytvořeny. Karty jsou zobrazovány v `RecyclerView` a jejich obsah je spravován v adaptéru `ReviewListAdapter`. Po stisknutí na recenzi je uživatel pomocí navigačního ovladače přesměrován na profil uživatele, který danou recenzi udělil.

Záložka Knihy je implementována ve fragmentu `MyBooksFragment` a uživatel zde má možnost vidět až 3 kategorie knih. V mém definovaném návrhu 3.2 byly pouze dvě kategorie, ale při implementaci jsem přidal na profil přihlášeného uživatele také kategorii „Knihy k navrácení“, kde uživatel vidí všechny knihy, které má od někoho vypůjčené. Má u nich také označen čas, do kdy musí být kniha vrácena majiteli. Tato změna lze vidět na obrázku 4.3c. Po vypršení času daný čas uživateli zčervená a odpocítává dalších 14 dní, po jehož uplynutí se kniha odstraní. V tomto časovém rozmezí nemůže majitel zaevidovat vrácení knihy na profilu, ale v notifikacích, protože se jedná o pozdní vrácení a ne dřívější, majitel je na to však upozorněn pomocí `Snackbar`. Tlačítko na vrácení vypůjčené knihy bylo přesunuto do výběrové nabídky. Pro zobrazení jednotlivých karet knih v každé kategorii v `RecyclerView` využívám až 3 instance adaptérů `MyBooksListAdapter`. Využívám stejnou instanci `ViewModel` u adaptérů a tohoto fragmentu, kde ukládám a aktualizuju list každé kategorie knih. Díky tomu se mi aktualizuje grafické uživatelské rozhraní, ale i pozice daných knih pro případnou další interakci, když například smažu knihu, nebo ji vrátím.

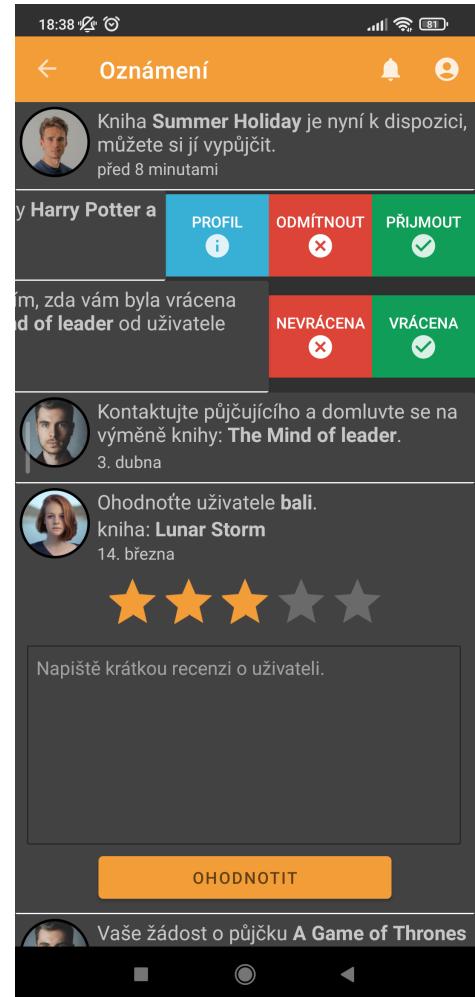
Oznámení

Všechny notifikace, které přijdou na zařízení, jsou čistě informační, poté co uživatel stiskne notifikaci, bude přesměrován do fragmentu `NotificationsFragment`, kde se stáhnou oznámení ze serveru a vykreslí se do komponent `CardView` v `RecyclerView`. Jak lze vidět na obrázcích 4.4a a 4.4b, v grafickém uživatelském rozhraní tohoto fragmentu došlo k výrazné úpravě ve srovnání s původním návrhem 3.2. Zmenšily se jednotlivá oznámení tak, aby měly přibližně stejnou výšku. Důvodem této změny bylo zlepšení přehlednosti a zobrazení většího množství oznámení na obrazovce. Existuje 8 typů notifikací, které jsou nezávisle na typu seřazeny sestupně dle data příchodu. V adaptéru `NotificationsListAdapter` jsem si vytvořil funkci `getNotificationArrivalTimeDate`, která má za úkol vybrat správnou gramatickou verzi času příchodu a to v případě, že daná notifikace přišla v ten den, jinak vrátí pouze datum příchodu.

Veškerá funkcionalita oznámení se děje v adaptéru `NotificationsListAdapter`, kde mám vytvořenou `ViewHolder` třídu pro každý typ oznámení, která slouží k držení odkazů na prvky uživatelského rozhraní. Pro vytvoření správné třídy pro daný typ oznámení slouží ID notifikace, které je součástí každé položky v listu notifikací, jež přijde v rámci odpovědi na klienta. Ve funkci `onBindViewHolder` se poté naplní každé oznámení daty dle výše zmíněného ID. Poslouchání změn v `LiveData` objektu z `ViewModel` probíhá ve fragmentu `NotificationsFragment`. V případě, že se zavolá nějaká `ViewModel` metoda v adaptéru při interakci s interaktivním typem oznámení pro zaslání požadavku na server, bude odpověď zachycena ve fragmentu. V případě úspěšné odpovědi se opět zavolá `ViewModel` metoda `getNotifications()`, která získá všechny notifikace ze serveru, aby se aktualizoval fragment. Například když uživatel potvrdí výpůjčku knihy, odstraní se daná notifikace a přijde nová, ve které může kontaktovat půjčujícího.



(a) Oznámení, část 1.



(b) Oznámení, část 2.

Obrázek 4.4: Na prvním snímku obrazovky lze vidět oznámení ve výchozím stavu, zatímco druhý snímek ukazuje vzhled interaktivních oznámení po jejich stisknutí. Po stisknutí oznámení, která jsou stejná, se uživatel přesměruje buď na profil uživatele, nebo na stránku s podrobnostmi o knize, podle toho, o jaký typ oznámení se jedná.

4.4 Nedostatky implementace

Během implementace registrační funkcionality v aplikaci jsem narazil na problém s ověřením e-mailové adresy na mém serveru s použitím frameworku Ktor. Z tohoto důvodu není ověření e-mailové adresy při registraci dostupné a uživatelé se tak mohou registrovat s jakýmkoli e-mailem. Původně jsem plánoval pro ověřování použít SMTP server od Googlu, nicméně od května 2022 Google přestal podporovat ověření e-mailových adres pro nezabezpečené aplikace.

Dále jsem se rozhodl vynechat funkcionalitu administrátora, který by měl být zodpovědný za kontrolu nahlášených obsahů knih a případné odstranění daných knih a dalších souvisejících funkcionalit. Toto rozhodnutí bylo motivováno vysokou náročností na čas.

Během implementace oznámení jsem byl v domnění, že mohu poslat plánované oznámení na Firebase Cloud Messaging, které by bylo klientovi doručeno po určité době. Zjistil

jsem však, že při použití Admin SDK to není možné a lze to provést pouze manuálně přes Firebase webovou stránku. Potřeboval jsem tuto funkčnost pro odeslání plánované notifikace majiteli knihy, kterému by oznámila, že kniha, kterou někomu vypůjčil, již překročila maximální dobu výpůjčky. Nejlepší alternativa, která mě napadla, bylo kontrolovat každý den, zda někomu již vypršela maximální doba výpůjčky, což vyžaduje mít neustále spuštěný server. Jelikož však nemám nasazený server v nějakém cloudovém prostředí, protože ty jsou většinou placené, tak jsem vyvinul ne příliš ideální řešení. Toto řešení spočívá v kontrolním zjišťování na hlavní straně aplikace, kde je posílan požadavek na endpoint `GET /books`, zda některému uživateli již vypršela doba výpůjčky a pokud ano, tak je mu odeslána notifikace. To je z toho důvodu, že hlavní strana aplikace je v rámci aplikace nejvíce používaná.

Měl jsem v úmyslu přidat možnost odstraňování notifikací v rámci fragmentu `NotificationsFragment`. Endpoint pro odstraňování notifikací již mám implementovaný. Nicméně, vznikl problém, že pouze asi 3 z 8 typů notifikací bylo možné odstranit, jelikož ostatní notifikace zmizí automaticky při určité akci. Pro uživatele by bylo obtížné rozpoznat, které notifikace mohou být odstraněny a které ne, takže jsem se nakonec rozhodl tuto funkcionality neimplementovat v klientské části.

Za situace, kdy v databázi existuje velké množství knih, dochází k výraznému zpomalení načítání hlavní stránky. Tento problém může být způsoben buď velkou velikostí obrázků, nebo tím že jsou obrázky přenášeny v JSON odpovědi zakódované v Base64, což vyžaduje následné dekódování. Jako možné řešení by bylo vhodné zobrazit karty knih co nejdříve a nechat samotné obrázky načítat následně.

V budoucím vývoji bude kladen důraz na odstranění nedostatků, které byly v této podkapitole identifikovány.

Kapitola 5

Testování aplikace

Testování je nezbytnou součástí vývoje aplikací, neboť umožňuje identifikovat a odstranit chyby a problémy, které by mohly ovlivnit kvalitu a funkčnost aplikace. V následující kapitole se zaměřuji na testování responzivity aplikace na různých velikostech mobilních zařízení, včetně tabletů. Dále popisují průběh testování s uživateli, jejich zpětnou vazbu a výsledné zhodnocení aplikace.

Testování responzivity a kompatibility na různých zařízeních

Testování kompatibility mobilních aplikací je nezbytnou součástí vývoje, jelikož je nutné otestovat chování aplikace na různých velikostech displeje a verzích operačního systému. Uživatelské rozhraní je tedy nutné vytvořit dostatečně responzivní a zabezpečit funkčnost všech prvků v rozsahu podporovaných verzí. V případě mé aplikace je důležité zmínit, že je určena pro operační systémy Android verze 8+ (SDK 26+), proto mi šlo o to zjistit, zda funguje na všech podporovaných verzích Androidu 8+.

Pro testování responzivity na různých zařízeních byl využit emulátor dostupný v Android Studiu pro nejmenší i největší obrazovky včetně tabletu. V průběhu testování byly identifikovány grafické problémy, které byly následně opraveny.

K otestování kompatibility na různých zařízeních byl vygenerován APK soubor a distribuován mezi několika známými, kteří měli za úkol ověřit, že aplikace funguje na jejich zařízení bez problémů. Všechny testy byly úspěšné a nebyl zaznamenán žádný problém.

5.1 Průběh testování s uživateli

Pilotní testování mé aplikace, které má za účel ověřit samotný proces testování, bylo provedeno na dálku prostřednictvím platformy Discord. Testovací subjekt spustil mou aplikaci v prostředí Android Studio, přičemž sdílel svou obrazovku, zatímco můj server běžel na mém lokálním počítači s veřejnou IP adresou. Při této fázi jsem ověřil mé testovací postupy a ujistil se, že aplikace funguje správně. Tento testovací subjekt nebyl zahrnut do závěrečného hodnocení aplikace.

V rámci osobního testování jsem testoval aplikaci se šesti uživateli: čtyři byli ve věku okolo 20 let, a další dva byli ve věku 31 a 50 let. Testování probíhalo na mém mobilním zařízení, kde jsem měl aplikaci spuštěnou, zatímco na mém lokálním počítači běžel server a emulátor, který jsem použil pro interagování s uživatelem v rámci testování. S jejich souhlasem jsem nahrával jejich hlasy a obrazovku na mobilu s využitím aplikace AZ Screen

Recorder a zobrazoval, kam uživatel stiskl na obrazovce. Dále jsem uživatelům nabídl výběr mezi světlým a tmavým režimem, což je nastavováno přímo v mobilním zařízení.

Během testování jsem žádal uživatele, aby hlasitě vyjadřovali své myšlenky a zároveň jsem je instruoval, aby se mě neptali na konkrétní postup řešení úkolů. Tento přístup mi umožnil zjistit, zda jsou schopni používat aplikaci samostatně. Prvním scénářem bylo vypůjčit svou knihu, což zahrnovalo registraci, přihlášení a přidání dané knihy ve formě formuláře v aplikaci. Poté jsem na svém emulátoru požádal o vypůjčení právě přidané knihy a testovací subjekt mohl vypůjčení přijmout v rámci oznámení, nebo si nejprve prohlédnout mé hodnocení a recenze. Po potvrzení výpůjčky jsem uživatele informoval, že uplynulo několik týdnů a knihu jsem mu vrátil. Jeho dalším úkolem bylo zaevidovat vrácení knihy v aplikaci, aby si ji mohli půjčit další uživatelé. Po vrácení knihy uživatel ohodnotil půjčujícího v rámci oznámení.

Druhý scénář testování byl specifický. Uživatelům jsem zadával následující úkoly:

- Použijte filtr a najděte knihu, která je zdarma, patří do žánru fantasy a je volná k vypůjčení.
- Prohlédněte si recenze majitele té knihy.
- Požádejte o vypůjčení dané knihy (kterou majitel knihy přijme).
- Zjistěte, kolik času zbývá do vrácení knihy, kterou jste si vypůjčili.
- Najděte již vypůjčenou knihu a vysvětlete mi, co si myslíte, že dělá tlačítko „Upozornit mě“.
- Vyzkoušejte další funkcionality, jako jsou změna hesla, aktualizace kontaktních údajů, aktualizace dříve přidané knihy, jejího následného odstranění a odstranění účtu.

Na závěr jsem testovacím subjektům zadal úkol vyplnit dotazník v Google Forms, který se týkal uživatelského prožitku. Dotazník se skládal ze dvou částí. První část obsahovala 10 oficiálních otázek z SUS¹ (System Usability Scale). Respondenti měli na každou otázku odpovědět číslem od 1 do 5, kde 1 znamená silně nesouhlasím a 5 silně souhlasím. Druhá část dotazníku obsahovala osm otázek z krátké verze dotazníku UEQ² (User Experience Questionnaire). Uživatelé měli vybrat číslo od 1 do 7, kde čísla 1 a 7 byly spojeny s přídavným jménem. Respondenti měli vybrat číslo, které se blíží k přídavnému jménu, jež podle nich lépe popisuje danou aplikaci.

5.2 Zpětná vazba uživatelů

Během testování aplikace jsem získal několik cenných zpětných vazeb od uživatelů, které jsou důležité pro budoucí vývoj a rozšíření aplikace. Tyto zpětné vazby budou zohledněny při plánování a implementaci budoucích funkcionalit.

Jedna z významných zpětných vazeb byla ohledně neefektivity zadávání data narození v neobvyklém formátu při registraci. Někteří uživatelé navrhovali vylepšení pomocí tzv. pickers, což jsou nástroje pro rychlejší a přesnější výběr data, než pomocí ručního zadávání.

Další důležitá zpětná vazba se týkala zaevidování dřívějšího vrácení knihy, ale také její editace a odstranění. Někteří uživatelé měli potíže s nalezením tlačítka pro vrácení knihy,

¹<https://designdokapsy.cz/metody/vyzkum/system-usability-scale-sus/>

²<https://www.ueq-online.org/>

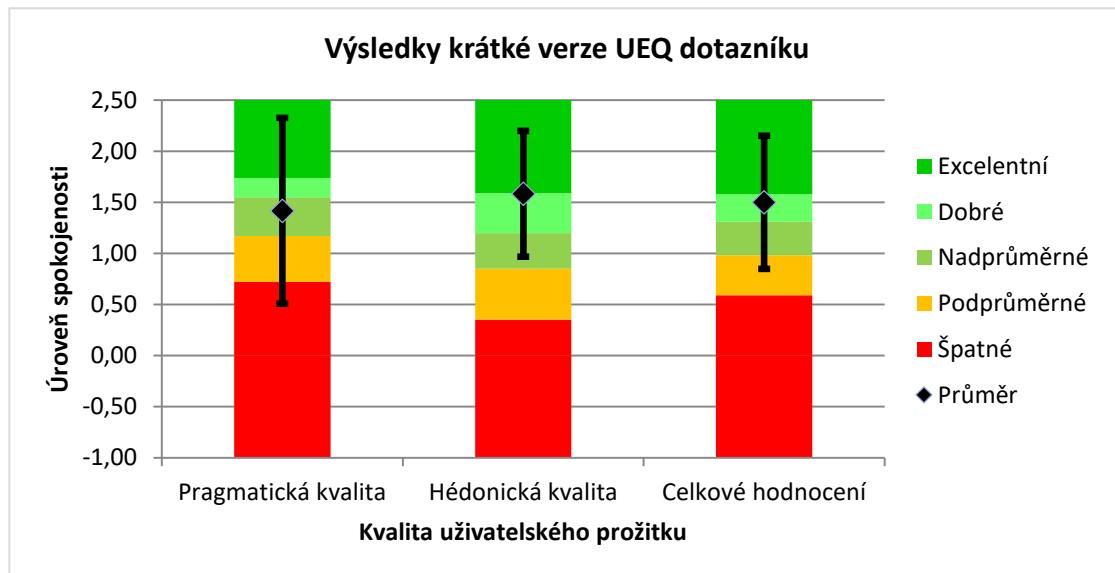
editaci a odstranění, protože očekávali, že se tato funkcionality nachází na stránce, kde jsou obsaženy podrobnější informace o dané knize. Také si nevšimli výběrové nabídky v profilu u karty knihy, kde je tato funkcionality k dispozici.

5.3 Závěrečné hodnocení aplikace

V rámci hodnocení aplikace bylo zjištěno, že je spíše vhodná pro mladší generaci a lidi středního věku. To bylo potvrzeno testováním, když pět mladých testovacích subjektů dokázalo aplikaci otestovat v průběhu 11-16 minut, zatímco jeden padesátiletý testovací subjekt testoval 24 minut, přičemž měl spoustu dotazů a potřeboval pomoc při používání aplikace.

Výsledné hodnocení uživatelského prožitku pomocí SUS se udává na škále 0-100, kde nejlepší možné hodnocení odpovídá hodnotě 100. Hodnoty 68 a vyšší jsou považovány za nadprůměrné, hodnoty mezi 50 a 68 jsou průměrné a hodnoty pod 50 jsou podprůměrné. V případě, že hodnocení uživatelského prožitku klesne pod 68, je potřeba aplikaci dále vylepšovat. Vzhledem k tomu, že výsledek testování mé aplikace ukázal hodnotu 82,5, lze tedy konstatovat, že uživatelský prožitek mé aplikace obdržel nadprůměrné hodnocení.

Vyhodnocování druhého dotazníku (krátký UEQ) už není tak jednoznačné. Oficiální tabulkový soubor, který je k dispozici na webové stránce pro UEQ dotazníky, přijímá hodnoty z dotazníku a zpracovává je do formy tabulek a grafů. Tyto výsledky jsou rozděleny na pragmatickou kvalitu a hedonickou kvalitu. Pragmatická kvalita zahrnuje, jak snadno se v aplikaci hledá a otevírá obsah, jaké funkce jsou k dispozici a jak efektivně jsou použitelné. Hedonická kvalita se týká zážitku uživatele z produktu. Zahrnuje to, jak příjemné a uspokojivé je používání produktu a jak uživatelé vnímají jeho design a vzhled. Pokud jde o aplikaci, hedonická kvalita zahrnuje to, jak atraktivní je design aplikace a jak příjemně a snadno se s ní pracuje. Po zprůměrování výsledných hodnot pragmatických a hedonických kvalit mi vyšlo 1,5, což znamená, že má aplikace v celkovém hodnocení obdržela dobré hodnocení. Tyto hodnoty jsou podrobněji znázorněny na následujícím grafu v obrázku 5.1.



Obrázek 5.1: Graf vygenerovaný z krátkého UEQ dotazníku. Čára znázorňuje rozsah hodnot, kterých uživatelé dosáhli při vyplňování dotazníku a černá tečka ukazuje jejich průměr. Celkové hodnocení uživatelského prožitku se pohybuje průměrně v dobrém hodnocení.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit mobilní aplikaci pro zařízení s operačním systémem Android, která umožní uživatelům půjčovat si knihy navzájem a interagovat mezi sebou prostřednictvím oznamení. V rámci této práce jsem se zaměřil na studium architektur, uživatelského prožitku a uživatelského rozhraní, které jsem poté využil při návrhu vlastní aplikace. Pro inspiraci jsem vycházel z již existujících aplikací s podobnou funkcionalitou, které mi poskytly několik užitečných prvků. Kromě návrhu uživatelského rozhraní, bylo třeba také navrhnout ER diagram databáze a navrhnout API ve webové aplikaci SwaggerHub. Poté jsem přistoupil k implementaci aplikace, kterou jsem rozdělil na dvě části – server a klient, obě napsány v jazyce Kotlin. Během implementace jsem provedl několik drobných změn od původního návrhu, které vylepšily výslednou aplikaci. V poslední řadě jsem provedl testování aplikace s šesti uživateli, kde jsem získal převážně pozitivní zpětnou vazbu. Kromě toho jsem vytvořil video¹, demonstrující jak spolu interagují dvě osoby při půjčování knih.

Při případném budoucím vývoji plánuji přidat chat, který by usnadnil komunikaci mezi uživateli. Dále chci vyřešit problémy, které se vyskytly v průběhu implementace a doimplementovat další funkcionality navržené uživateli během testování.

Jedná se o mou první mobilní aplikaci jakou jsem kdy vytvořil. Odnesl jsem si spoustu zkušeností v oblasti klientské i serverové části vývoje. Naučil jsem se posílat notifikace přes server daným klientům s využitím Firebase Cloud Messaging, které jsou neodmyslitelnou součástí této aplikace. Naučil jsem se také navrhovat a implementovat API rozhraní pro vzájemnou komunikaci mezi klientem a serverem. Velkou výzvou pro mě bylo testování s uživateli, na které jsem přistoupil s maximálním úsilím. Celkově se dá konstatovat, že tato zkušenosť mi nejen obohatila znalosti, ale rovněž mě inspirovala se vydat cestou vývoje aplikací.

¹https://youtu.be/iBCinCb_s1M

Literatura

- [1] BLAŽKOVÁ, T. *Nahradí Kotlin Javu? V čem psát aplikace pro Android?* [online]. Září 2021 [cit. 2023-03-22]. Dostupné z: <https://www.itnetwork.cz/blog/nahradi-kotlin-javu-v-cem-psat-aplikace-pro-android>.
- [2] DESIGNMATCH. *UX Design Guide for Startup Owners / Essentials you need to know* [online]. Březen 2022 [cit. 2023-04-25]. Dostupné z: <https://www.designmatch.io/ux-design-guide-startup/>.
- [3] DORFMANN, H. *Model-View-Intent on Android* [online]. Březen 2016 [cit. 2023-04-25]. Dostupné z: <http://hannesdorfmann.com/android/model-view-intent/>.
- [4] EUM, J. *Design Patterns and Architecture: The Android Developer Roadmap – Part 4* [online]. Září 2022 [cit. 2023-03-20]. Dostupné z: <https://getstream.io/blog/design-patterns-and-architecture-the-android-developer-roadmap-part-4/>.
- [5] GARRETT, J. J. *The elements of user experience*. 2. vyd. Upper Saddle River, NJ: New Riders Publishing, prosinec 2010. ISBN 978-0-321-68368-7.
- [6] GOOGLE DEVELOPERS. *Activity*. Březen 2023 [cit. 2023-03-23]. Dostupné z: <https://developer.android.com/reference/kotlin/android/app/Activity>.
- [7] GOOGLE DEVELOPERS. *Fragment*. únor 2023 [cit. 2023-03-23]. Dostupné z: <https://developer.android.com/reference/kotlin/android/app/Fragment>.
- [8] GOOGLE DEVELOPERS. *Layouts*. Březen 2023 [cit. 2023-03-23]. Dostupné z: <https://developer.android.com/develop/ui/views/layout/declaring-layout>.
- [9] JEMEROV, D. a ISAKOVA, S. *Kotlin in Action*. 1. vyd. New York, NY: Manning Publications, prosinec 2016. ISBN 978-161-7293-290.
- [10] JUNEK, P. *Operační systém Android* [online]. Říjen 2008 [cit. 2023-03-21]. Dostupné z: <https://www.itnetwork.cz/mobilni-zarizeni/android/android-operacni-system-google/>.
- [11] PARONAI, T. *Java vs. Kotlin, který programovací jazyk je lepší pro vývoj aplikací pro Android?* [online]. Únor 2023 [cit. 2023-03-22]. Dostupné z: <https://www.goodrequest.com/cs/blog/java-vs-kotlin-ktery-programovaci-jazyk-je-lepsi-pro-vyvoj-aplikaci-pro-android>.
- [12] STEVENS, E. *What Is User Experience (UX) Design? Everything You Need to Know* [online]. Březen 2023 [cit. 2023-03-23]. Dostupné z: <https://careerfoundry.com/en/blog/ux-design/what-is-user-experience-ux-design-everything-you-need-to-know-to-get-started/>.

- [13] THEY MAKE DESIGN. *What is UI design? What is UX design? UI vs UX: What's the difference* [online]. Únor 2019 [cit. 2023-03-22]. Dostupné z:
<https://uxplanet.org/what-is-ui-vs-ux-design-and-the-difference-d9113f6612de>.
- [14] URBAŃCZYK, M. *Lekce 1 - Úvod do Jetpack Compose* [online]. Únor 2023 [cit. 2023-03-22]. Dostupné z:
<https://www.itnetwork.cz/kotlin/compose/uvod-do-jetpack-compose>.

Příloha A

Obsah přiloženého paměťového média

```
/Bachelor-Thesis
└── /BookSharingClient - zdrojové soubory klientské části
    ├── /BookSharingClient - zdrojové soubory klientské části
    ├── /BookSharingServer - zdrojové soubory serverové části
    ├── /BookSharingLatex - zdrojové soubory technické zprávy
    ├── BookSharingReport.pdf - technická zpráva
    ├── BookSharingVideo.mp4 - video zobrazující funkcionalitu aplikace
    ├── BookSharingApp.apk - instalovační soubor na mobilní aplikaci Android
    ├── booksharing.sql - soubor obsahující databázové tabulky a data
    └── README.md - návod na instalaci
```