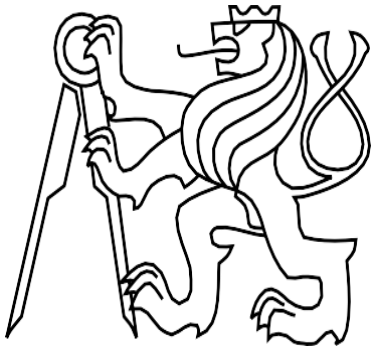
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Android aplikace pro úvěrovou společnost

Stanislav Benda

Vedouc práce: Ing. Jiří Šebek

24. května 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Benda** Jméno: **Stanislav** Osobní číslo: **508468**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Enterprise systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Android aplikace pro úvěrovou společnost

Název bakalářské práce anglicky:

Android application for online banking

Pokyny pro vypracování:

Cílem je vytvořit aplikaci, která bude pomáhat udržovat přehled o úvěrech, investicích, fondech, akciích a komoditách jako je zlato stříbro apod. Také tato aplikace bude centralizování pro všechny banky čili zde bude přehled všech těchto služeb ze všech bank, které uživatel používá. První částí bude analýza existujících řešení, která poskytne přehled o funkcích, které tyto aplikace obsahují. Druhým cílem práce je navrhnout architekturu aplikace, zanalyzovat možné způsoby řešení a vybrat ty nejvhodnější a ve druhé části je implementace a testování.

Postup:

1. Seznamte se s problematikou správy úvěrů soukromých osob.
2. Proveďte analýzu vám dostupných konkrétních aplikací a existujících řešení, která poskytne přehled o funkcích, které tyto aplikace obsahují.
3. Na základě provedené analýzy navrhněte základní funkcionality navrhované aplikace.
4. Zvolte architekturu aplikace a vyberte nejvhodnější technologie pro tvorbu mobilní aplikace.
5. Aplikaci implementujte a otestujte.
6. Zhodnoťte výsledky a navrhněte případné další funkcionality nebo jiná zlepšení.

Seznam doporučené literatury:

[1] Roger S. Pressmann Bruce Maxim: Software Engineering: A Practitioner's Approach, ISBN-10: 9780078022128
[2] Software pro správu úvěrů a půjček. Online. OneCore.com. 2024 Dostupné z: https://one-core.com/cs-cz/sprava-uveru-a-pujcek. [cit. 2024-02-12].
[3] Správa hypotéky - Raiffeisen Bank. Online. OneCore.com. 2024. Dostupné z: https://www.rb.cz/osobni/hypoteky/sprava-hypoteky. [cit. 2024-02-12].

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **15.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Prohlášení

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

V Praze dne 24.5.2024 .....

Stanislav Benda

Poděkování

Děkuji vedoucímu práce Ing. Jiřímu Šebkovi za pečlivé vedení práce, odborný dohled a cenné rady při vypracovávání jak dokumentu, tak prototypu aplikace.

Stanislav Benda

Abstrakt

Tato bakalářská práce se zaměřuje na vývoj mobilní aplikace pro sledování bankovních účtů a finančních operací s cílem zjednodušit správu financí uživatelů. Práce se věnuje analýze stávajících řešení na trhu a identifikaci klíčových požadavků uživatelů. Hlavním cílem je vytvořit aplikaci, která umožňuje integraci a přehledné zobrazení účtů z různých bankovních institucí, jako jsou Komerční banka, ČSOB, George a Coinbase.

Metodologie zahrnuje literární rešerši, analýzu moderních technologií a návrh a implementaci aplikace pro platformu Android. Aplikace bude vyvíjena v jazyce Kotlin a bude využívat služeb Firebase pro autentizaci a ukládání dat. Klíčové funkcionality zahrnují registraci a přihlášení uživatelů, zobrazení seznamu bankovních účtů, přidávání nových účtů a správu finančních dokumentů.

Vývoj aplikace zahrnuje použití architektonického vzoru MVVM pro oddělení logiky od uživatelského rozhraní, což zajišťuje lepší udržovatelnost a rozšiřitelnost kódu. Dále byly využity design patterny Adapter, Bridge a Decorator k řešení specifických problémů a zajištění flexibility aplikace.

Očekávaným výsledkem je funkční mobilní aplikace, která poskytne uživatelům snadný a intuitivní nástroj pro správu jejich finančních účtů a dokumentů. Závěry práce budou zahrnovat hodnocení úspěšnosti aplikace a diskusi o jejím potenciálním rozvoji do budoucích let.

Klíčová slova: Bankovní účty, Mobilní aplikace, Vývoj aplikací, Kotlin, Firebase, MVVM

Abstract

This bachelor's thesis focuses on the development of a mobile application designed to simplify financial management for users by tracking bank accounts and financial transactions. The thesis includes an analysis of existing market solutions and the identification of key user requirements. The primary goal is to create an application that allows for the integration and clear display of accounts from various banking institutions, such as Komerční banka, ČSOB, George, and Coinbase.

The methodology encompasses a literature review, analysis of modern technologies, and the design and implementation of the application for the Android platform. The application will be developed using the Kotlin language and will utilize Firebase services for authentication and data storage. Key functionalities include user registration and login, displaying a list of bank accounts, adding new accounts, and managing financial documents.

The development process involves using the MVVM architectural pattern to separate logic from the user interface, ensuring better maintainability and extensibility of the code. Additionally, design patterns such as Adapter, Bridge, and Decorator have been used to address specific issues and provide flexibility to the application.

The expected outcome is a functional mobile application that offers users an easy and intuitive tool for managing their financial accounts and documents. The conclusions of the thesis will include an evaluation of the application's success and a discussion of its potential development in future years.

Keywords: Bank Accounts, Mobile Application, Application Development, Kotlin, Firebase, MVVM

Seznam použitých zkratek

KB – Komerční banka

UI – User interface

UX – User expirience

API – Application programming interface

JSON – JavaScript Object Notation

MVVM – Model view viewModel

XML – eXtensible Markup Language

IDE – Integrated Development Environment

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době, kdy digitální technologie ovlivňují téměř každou oblast našeho života, je nevyhnutelným trendem i transformace bankovnictví a správy financí. S nástupem online bankovnictví a mobilních aplikací se uživatelé dostávají do nového světa možností, kde mohou spravovat své finance z pohodlí svého domova nebo z mobilního zařízení kdykoli a odkudkoli. Tento vývoj vytváří nové příležitosti, ale zároveň klade před nás nové výzvy, zejména v oblasti integrace a sledování účtů od různých poskytovatelů.

1.1 Cíl práce

Hlavním cílem této bakalářské práce je vytvořit komplexní mobilní aplikaci pro platformu Android, která bude fungovat jako centrální místo pro správu bankovních a investičních služeb. Aplikace bude umožňovat uživatelům sledovat a spravovat všechny své bankovní účty, půjčky, investice a spořicí účty od různých poskytovatelů. K dosažení hlavního cíle bude potřeba splnit několik průběžných cílů.

Prvním cílem bude seznámení se službami, které poskytují banky a jiné investiční stránky, analyzovat stávající trh a vyhodnotit jaké slabé stránky stávající aplikace mají.

Dalším důležitým cílem je navrhnutí funkcionalit aplikace a analyzovat je. Analýza bude provedena pomocí případů užití.

A jako posledním cílem je implementovat aplikaci, její zvolené funkcionality a následně je otestovat pro telefony s operačním systémem Android. Na konci bude prostor pro navrhnutí budoucí práce a možná rozšíření funkcionalit aplikace.

1.2 Motivace

Hlavní motivačním faktorem pro mě bylo to, že v jeden moment jsem vlastnil 3 různé druhy služeb od 3 rozdílných poskytovatelů a mimo to jsem měl další aplikaci kde jsem měl své finance investované v kryptoměnách. Abych si udržel přehled musel jsem pracně proklikávat mezi třemi aplikacemi místo toho, abych je měl možnost sledovat na jednom místě a otevřel si pouze jednu aplikaci.

Kapitola 2

Rešerše existujících řešení

Tato kapitola se zaměřuje na analýzu současných existujících řešení pro sledování bankovních účtů a finančních operací prostřednictvím mobilních aplikací. Průzkum trhu a existujících aplikací je klíčovým prvkem v procesu vývoje nové aplikace, protože poskytuje cenné poznatky o konkurenčním prostředí, uživatelských potřebách a trendech v oblasti mobilního bankovnictví.

2.1 Rozvoj mobilních aplikací

Od svých skromných začátků jako jednoduché aplikace pro komunikaci a zábavu se mobilní aplikace vyvinuly v komplexní nástroje, které tvoří základ moderní digitální společnosti. Jejich rozmanitost a funkčnost se staly nepostradatelnými pro každodenní život a usnadňují mnoho aspektů lidské činnosti.

Za posledních několik let došlo k explozivnímu nárůstu aplikací, které se staly klíčovými hráči v různých oblastech. Kromě tradičních aplikací pro komunikaci a zábavu, jako jsou sociální sítě a hry, existuje celá řada specializovaných aplikací, které usnadňují a zlepšují různé aspekty lidského života.

Mezi ně patří například mobilní aplikace pro zpracování obrázků, které umožňují uživatelům úpravu fotografií pomocí různých filtrů a efektů, což napomáhá vytváření kreativního obsahu pro sociální sítě a sdílení vzpomínek s ostatními. Aplikace pro plánování cestování umožňují uživatelům vyhledávat lety a ubytování, objednávat si taxi a prohlížet si recenze a hodnocení cestovních destinací, což usnadňuje plánování a organizaci dovolených.

Nicméně jednou z nejvíce využívaných kategorií mobilních aplikací zůstává oblast finančního řízení. Uživatelé používají mobilní aplikace pro sledování svých bankovních účtů, placení účtů, provádění plateb, správu investic a tvorbu rozpočtů. Tyto aplikace poskytují uživatelům přehled o jejich finanční situaci a umožňují jim lépe řídit své peníze, což je klíčovým prvkem finančního úspěchu v dnešní době.

2.2 Přehled Trhu

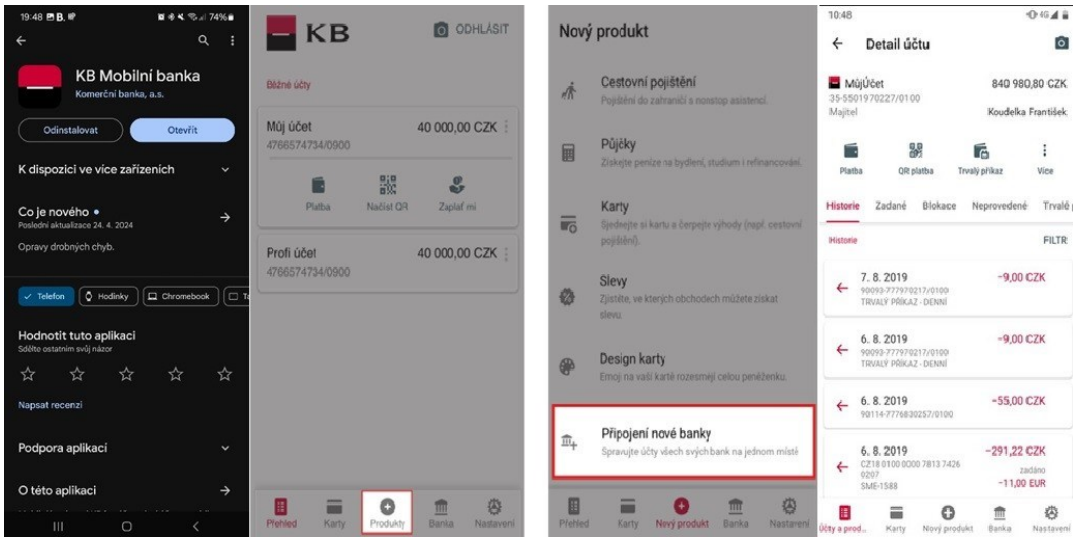
V rámci rešerše existujících řešení v oblasti mobilních aplikací pro úvěrové společnosti jsem se zaměřil na konkrétní aplikace, kterými jsou Mobilní banka od KB, George, ČSOB a Coinbase. Konkrétně jsem se zaměřil na služby typu osobní účet, půjčka, spořící účet a investice.

2.2.1 Mobilní banka od KB

Mobilní banka od KB [1] je mobilní android aplikace, která umožňuje sledovat účty jak osobní a spořící tak i investiční a úvěrové, provádět platby a sledovat veškeré svoje informace o finančních službách, které máte u společnosti KB. Tato aplikace disponuje

i funkcionalitou „Připojení nové banky“, ale tato funkcionalita je dostupná pouze ve webové verzi, což považuji za značnou nevýhodu.

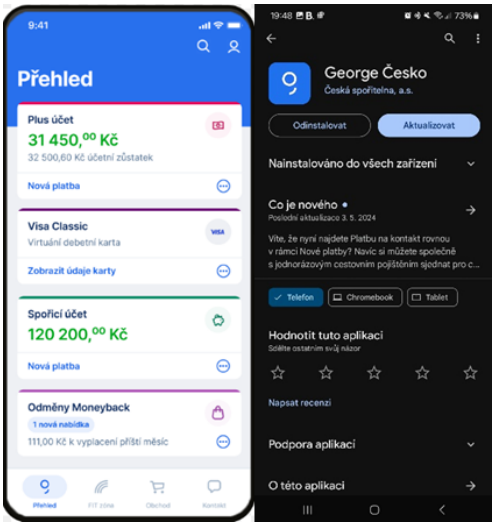
Existuje již i nástupce této aplikace, a to aplikace KB+, ale není ještě plně funkční a nejsou zde všechny funkce, které by měly být do budoucna implementovány, a proto jsem ji vynechal.



Obrázek 2.1: Série obrázku z KB Mobilní banka

2.2.2 Mobilní aplikace George od České spořitelny

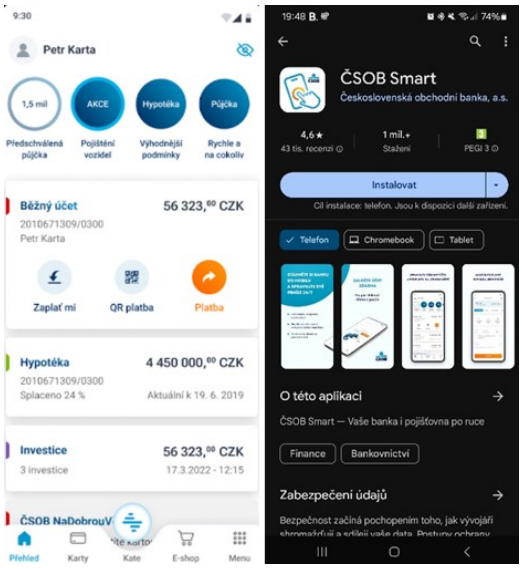
Mobilní aplikace George [2] je android aplikace stejně tak jako Mobilní banka od KB disponuje přehledem účtů, správou transakcí a správou dokumentů. Ale na rozdíl od aplikace od KB nedisponuje přidání externího účtu. Což je z hlediska sledování a správy financí ještě problematičtější než u zmíněné mobilní banky od KB.



Obrázek 2.2: Série obrázku z aplikace George

2.2.3 Mobilní aplikace ČSOB

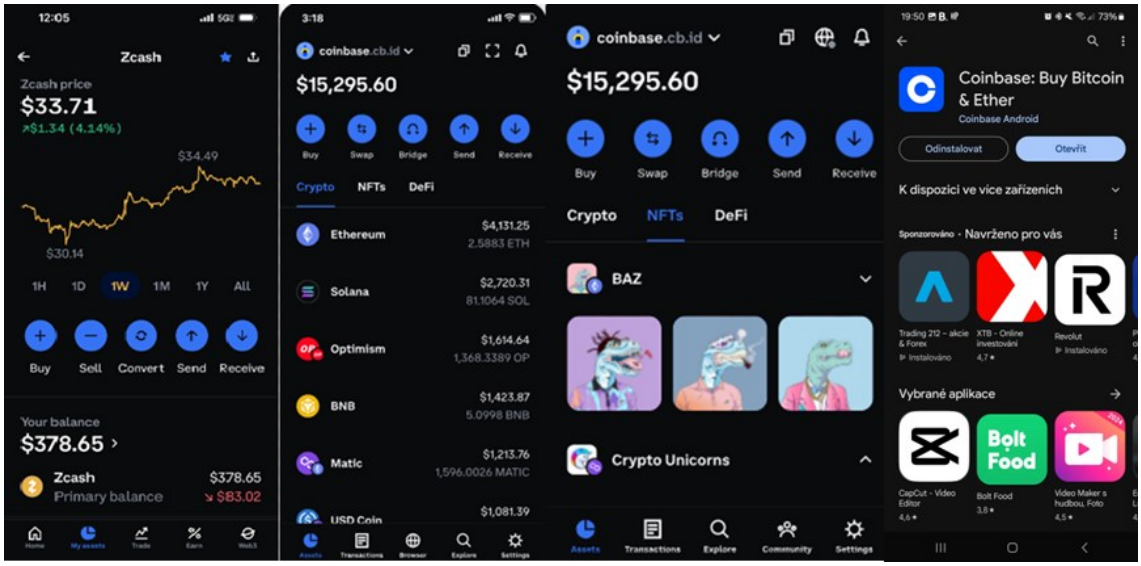
Aplikace od Československé obchodní banky ČSOB Smart [3] je jedna z nejvíce uživatelsky přívětivých aplikací. Tak jako její konkurence nabízí služby jako je sledování svých účtů, sledování transakcí a správa dokumentace, avšak bohužel pouze stále jen v kruhu služeb právě od jejích provozovatelů.



Obrázek 2.3:Série obrázku z aplikace ČSOB Smart

2.2.4 Mobilní aplikace Coinbase

Aplikace Coinbase [4] je kryptoměnová platforma pro Android, která nabízí možnost nákupu a prodeje s kryptoměnami jako je Bitcoin, Ethereum, Litecoin apod. Také jako ostatní aplikace disponuje správou všech svých transakcí a dokumentů. Navíc oproti ostatních výše zmíněných aplikacích je zde vidět pohyb momentálních cen nakoupené investice v reálném čase.



Obrázek 2.4: Série obrázku z aplikace Coinbase

2.3 Shrnutí a vyhodnocení slabých stránek

Během rešerše bylo zjištěno, že jediná aplikace, která podporuje něco podobného tomu, co je cílem mé bakalářky je starší verze mobilního bankovnictví KB, kde tato funkce momentálně není z neznámých důvodů dostupná anebo jejich webová aplikace. U jiných aplikací jsem nic tomuto podobného nenašel. UI jako takové mají všechny aplikace velice podobné. Je jednoduché poměrně intuitivní. Jediné, co bych vytkl všem aplikacím je to, že se liší lokace nalezení dokumentů a kolikrát je i schované níže v posuvném menu tím pádem i blbě viditelné na první pohled.

Kapitola 3

Analýza

Analýza je klíčovým prvkem v procesu vývoje aplikace, protože poskytuje základní povědomí o požadavcích uživatelů a prostředí, ve kterém bude aplikace provozována. V této kapitole provedeme detailní analýzu potřeb uživatelů, požadavků na funkční a nefunkční vlastnosti aplikace a provedeme průzkum dostupných technologií pro dosažení stanovených cílů.

3.1 Funkční požadavky

Funkční požadavky definují specifické chování nebo funkce, které systém musí splňovat. V případě aplikace pro sledování bankovních účtů a finančních operací je stanovení funkčních požadavků klíčové pro zajištění, že aplikace bude splňovat očekávání uživatelů a poskytne všechny potřebné funkcionality pro správu jejich financí.

3.1.1 Registrace

F1: Registrace – Pokud se jedná o uživatelovo první přihlášení bude se muset zadat své jméno, příjmení a email s heslem.

3.1.2 Přihlášení

F2: Přihlášení se – Po úspěšné registraci se uživatel bude moct přihlásit do aplikace.

F3: Přihlášení se zapamatování si uživatele – Při přihlášení si uživatel může zaškrtnout, jestli chce, aby si ho aplikace pamatovala. Po přihlášení bude vyzván k tomu, aby aktivoval přihlášení se pomocí otisku prstu.

3.1.3 Nastavení aplikace

F4: Přivítaní uživatele a zobrazení listu sledovaných – Po přihlášení uživatele aplikace přivítá a zobrazí mu list jeho sledovaných účtů.

F5: Odstranění účtu ze sledovaných – Aplikace bude umožňovat uživateli si odstranit účet z listu sledovaných.

F6: Přidání účtu do sledovaných – Aplikace bude umožňovat uživateli si přidat účet do listu sledovaných.

F7: Zobrazení všech přidaných účtů – Aplikace bude umožňovat uživateli zobrazit si všechny jeho přidané účty.

F8: Přidání účtu – Aplikace bude umožňovat si přidat účet s volbou od jaké instituce tento produkt je.

F9: Zobrazení svého profilu – Aplikace bude umožňovat si zobrazit svůj profil v aplikaci.

3.1.4 Profil

F10: Nahrání profilového obrázku do aplikace – Aplikace umožní uživatelovi si nahrát svůj obrázek jako profilovou fotku aplikace.

F11: Smazání fotky z aplikace – Aplikace umožní uživatelovi smazat svoji profilovou fotku z aplikace.

F12: Zobrazení si všech mých již nahraných dokumentů – Aplikace bude moci zobrazit všechny nahrané soubory do aplikace.

F13: Nahrání dokumentu v povoleném formátu do aplikace – Aplikace umožní uživateli nahrát soubor ve formátu PDF do aplikace.

F14: Stáhnutí si zvoleného dokumentu – Aplikace umožní uživateli si stáhnout zvolený soubor.

3.1.5 Detail účtu

F15: Zobrazení si detailu účtu – Aplikace umožní uživateli si zobrazit celý učet i jeho detaily po kliknutí.

F16: Zobrazení všech transakcích provedených v kontextu zvoleného účtu – Aplikace umožní uživateli si zobrazit transakce, které byly v rámci zvoleného účtu provedeny.

F17: Odhlášení se z aplikace – Aplikace umožní uživatelovi se odhlásit z aplikace

3.2 Nefunkční požadavky

N1: Příjemné UI – Do UI aplikace budou přidány plynulé animace, hezké barvy a dobře čitelné písmo pro zpříjemnění uživatelského zážitku.

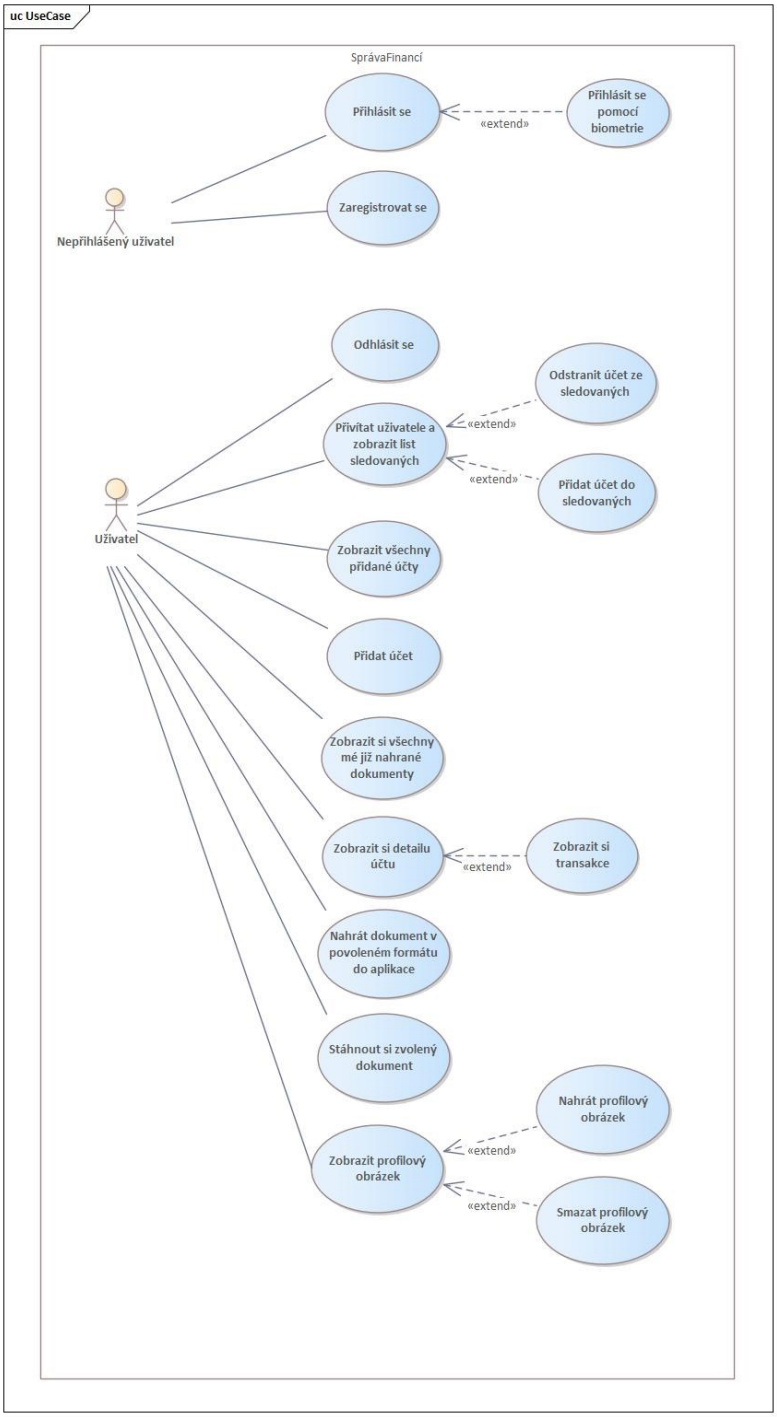
N2: Intuitivní UI – Aplikace bude velice podobná jako všechny bankovní aplikace zmíněné v rešerši, a tedy pro uživatele bude intuitivní a nebude potřeba orientační tutoriál zezačátku.

N3: Podporovatelnost – Prototyp aplikace bude podporován na každém telefonu, který má verzi androidu 7.0 Nougat a vyšší.

N4: Rozšiřitelnost – Architektura aplikace bude navržena tak aby do budoucna bylo možné přidat nové komponenty například změnu měny, či provádění transakcí přímo z aplikace.

3.3 Případy užití

Případy užití představují konkrétní scénáře, ve kterých uživatelé interagují s aplikací, a jsou klíčové pro definování funkcionality aplikace a jejích jednotlivých komponent. V diagramu případu užití jsou popsány všechny případy, níže jsou však popsány pouze vybrané nejdůležitější scénáře, které byly vypracovány na základě znalostí a materiálů z předmětu B6B36SMP [5].



Obrázek 3.1: Diagram případů užití

3.3.1 Registrace

Popis: Uživatel musí být schopen se zaregistrovat do aplikace a následně se přihlásit.

Aktéři: Nepřihlášený uživatel

Hlavní scénář:

1. Uživatel otevře aplikaci.
2. Uživatel zvolí možnost registrace na přihlašovací stránce.
3. Uživatel zadá své osobní údaje (jméno, příjmení, e-mail, heslo).
4. Systém ověří zadané údaje a vytvoří nový uživatelský účet.

Alternativní scénáře:

Pokud uživatel zadá neplatné nebo již existující údaje, systém zobrazí chybovou zprávu a nabídne opravu údajů.

3.3.2 Přivítání uživatele a zobrazení listu sledovaných účtů

Popis: Po přihlášení do aplikace je uživatel přivítán a zobrazuje se mu seznam sledovaných účtů.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Systém uživatele přivítá.
3. Systém zobrazí seznam sledovaných účtů uživatele.

Alternativní scénáře:

- Alternativní scénáře pro tento UC nejsou.

3.3.3 Odstranění účtu ze sledovaných

Popis: Uživatel může odstranit účet ze sledovaných.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel klikne na tlačítko, kde jsou 3 tečky.
3. Systém mu zobrazí možnost odebrat nebo přidat účet do sledovaných.
4. Uživatel zvolí sledovaný účet, který chce odstranit.
5. Uživatel klikne na tlačítko s křížkem jako volbu "Odstranit účet".
6. Systém odstraní vybraný účet ze seznamu sledovaných účtů.

Alternativní scénáře:

- Pokud se odstranění nezdaří, systém zobrazí chybovou zprávu a nabídne možnost opakování akce.

3.3.4 Přidání účtu do sledovaných

Popis: Uživatel může přidat nový účet do sledovaných.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí možnost "Přidat účet".
3. Uživatel zadá potřebné údaje pro propojení účtu (např. přihlašovací údaje, API klíče).
4. Systém ověří zadané údaje a propojí účet s aplikací.
5. Uživatel může vidět přidaný účet ve svém přehledu.

Alternativní scénáře:

- Alternativní scénáře pro tento UC nejsou.

3.3.5 Zobrazení všech přidaných účtů

Popis: Uživatel si může zobrazit přehled všech svých propojených bankovních účtů.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí v navigačním menu možnost "Moje účty".
3. Systém načte a zobrazí aktuální stav všech propojených účtů v přehledné formě (zůstatky, transakce atd.).

Alternativní scénáře:

- Pokud není jedna ze služeb dostupná či nebylo možné načíst detaily účtu, systém vyhodí chybovou hlášku.

3.3.6 Přidání účtu

Popis: Uživatel může přidat své bankovní účty z různých bankovních institucí (KB, ČSOB, George, Coinbase) do aplikace.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí možnost v navigačním menu "Přidat účet".
3. Systém zobrazí aktivitu pro přidání účtu.
4. Uživatel vybere bankovní instituci (KB, ČSOB, George, Coinbase).
5. Uživatel zadá potřebné údaje pro propojení účtu (číslo účtu).
6. Systém ověří zadané údaje a propojí účet s aplikací.

Alternativní scénáře:

- Pokud systém neověří zadané údaje, zobrazí uživateli chybovou zprávu a nabídne možnost opravy údajů nebo znovu zadání.

3.3.7 Zobrazení všech mých již nahraných dokumentů

Popis: Uživatel může zobrazit všechny dokumenty, které již nahrál do aplikace.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí v navigačním menu „Můj profil“
3. Systém zobrazí uživatelovi jeho profil.
4. Uživatel na aktivitě klikne na tlačítko „Moje dokumenty“
5. Systém zobrazí novou aktivitu se seznamem všech nahraných dokumentů uživatele.

Alternativní scénáře:

- Alternativní scénáře pro tento UC nejsou.

3.3.8 Nahrání dokumentu v povoleném formátu do aplikace

Popis: Uživatel může nahrát nový dokument do aplikace.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí možnost v navigačním menu „Můj profil“
3. Systém zobrazí uživatelovi jeho profil.
4. Uživatel vybere tlačítko „Moje dokumenty“.
5. Systém zobrazí uživatelovi jeho dokumenty s tlačítkem plus pro aktivitu "Nahrát dokument".
6. Uživatel klikne na tlačítko plus a zobrazí se mu výběr dokumentů z uložiště telefonu.
7. Uživatel vybere dokument v povoleném formátu (PDF).
8. Systém nahraje dokument a uloží ho do úložiště.

Alternativní scénáře:

- Pokud dokument není v povoleném formátu, systém zobrazí chybovou zprávu a nabídne možnost výběru správného formátu.

3.3.9 Stažení zvoleného dokumentu

Popis: Uživatel může stáhnout zvolený dokument z aplikace do svého zařízení.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí možnost v navigačním menu „Můj profil“
3. Systém zobrazí uživatelovi jeho profil.
4. Uživatel klikne na tlačítko "Moje dokumenty".
5. Uživatel vybere dokument, který chce stáhnout.
6. Uživatel zvolí možnost "Stáhnout".
7. Systém stáhne vybraný dokument do zařízení uživatele.

Alternativní scénáře:

- Pokud se stahování nezdaří, systém zobrazí chybovou zprávu a nabídne možnost opakování akce.

3.3.10 Zobrazení detailu účtu

Popis: Uživatel může zobrazit detailní informace o vybraném účtu.

Aktéři: Uživatel

Hlavní scénář:

1. Uživatel se přihlásí do aplikace.
2. Uživatel zvolí možnost v navigačním menu "Moje účty".
3. Uživatel vybere konkrétní účet ze seznamu a klikne na něj.
4. Systém zobrazí detailní informace o vybraném účtu (např. historie transakcí, aktuální zůstatek, úrokové sazby).

Alternativní scénáře:

- Pokud se detailní informace nezobrazí, systém zobrazí chybovou zprávu a nabídne možnost opakování akce.

3.4 Analýza dostupných technologií pro vývoj na platformě Android

Při výběru programovacího jazyka pro vývoj Android aplikace bylo důležité zohlednit několik klíčových faktorů, včetně kompatibility s platformou, dostupnosti knihoven a nástrojů, snadnosti učení a podpory komunity. Níže jsou popsány hlavní programovací jazyky, které byly zvažovány.

3.4.1 Programovací jazyky

Při vývoji aplikace pro sledování bankovních účtů a finančních operací byly zváženy různé programovací jazyky. Volba vhodného programovacího jazyka je zásadní pro efektivní vývoj, údržbu a škálovatelnost aplikace. Pro tento projekt byly vyhodnoceny následující programovací jazyky:

3.4.1.1 Kotlin

Moderní programovací jazyk vyvinutý pro platformu Android. Nabízí jednoduchou syntaxi, bezpečnost typů a interoperabilitu s existujícím kódem napsaným v Javě. Kotlin je preferovaným jazykem pro vývoj aplikací pro Android a má silnou podporu od Googlu. [6]

3.4.1.2 Java

Tradiční programovací jazyk pro vývoj aplikací pro platformu Android. I když už není tak populární jako dříve, stále je hojně používán a má silnou komunitní podporu. [7]

3.4.1.3 Dart

Jazyk vyvinutý společností Google a používaný v rámci frameworku Flutter pro vývoj multiplatformních mobilních aplikací. Dart je možné použít i pro vývoj aplikací určených specificky pro Android. [8]

Kritéria	**Kotlin**	**Java**	**Dart**
Popularita jazyka	Velmi populární	Stále hojně používán	Rozvíjející se
Komplexnost	Jednoduchá syntaxe, bezpečnost typů	Standardní syntaxe, široká podpora	Podobná syntaxe jako Java, moderní prvky
Komunitní podpora	Velká	Velká	Rychle rostoucí
Integrace s Android Studio	Ano	Ano	Ano
Aktualizace a podpora	Pravidelné aktualizace od JetBrains	-	Pravidelné aktualizace od Googlu

Tabulka 3.1: Porovnání vlastností programovacích jazyků

3.4.2 Vývojová prostředí

Pro vývoj aplikací na platformě Android existuje několik populárních vývojových prostředí (IDE), z nichž každé nabízí různé funkce a výhody. V této části se podíváme na hlavní IDE, která byla zvažována pro tento projekt.

3.4.2.1 Android Studio

Android Studio je oficiální vývojové prostředí pro platformu Android poskytované společností Google. Je postaveno na základech IntelliJ IDEA a je optimalizováno pro vývoj aplikací pro Android. Obsahuje širokou škálu nástrojů pro vývoj, ladění a testování aplikací. Díky svému oficiálnímu postavení a podpoře od Google je často doporučováno pro vývoj Android aplikací. [9]

3.4.2.2 IntelliJ IDEA s pluginem pro vývoj Androidu

IntelliJ IDEA je populární integrované vývojové prostředí poskytované společností JetBrains. S pluginem pro vývoj Androidu lze využívat většinu funkcí dostupných v Android Studiu. Je vhodné pro vývojáře, kteří již používají IntelliJ IDEA a nechtějí přepínat na jiné IDE. [10]

3.4.2.3 Eclipse s Android Development Tools (ADT) pluginem

Eclipse [11] je starší, ale stále používané integrované vývojové prostředí. S pluginem ADT [12] umožňuje vývoj aplikací pro platformu Android. Stále je dostupné pro vývojáře, kteří jsou s ním obeznámeni, a kteří preferují jeho uživatelské prostředí.

3.4.2.4 Visual Studio s Xamarin.Android

Visual Studio je vývojové prostředí poskytované společností Microsoft. S Xamarin.Android pluginem umožňuje vývoj Android aplikací pomocí jazyka C#. Je vhodné pro vývojáře, kteří preferují jazyk C# a chtějí využívat funkce a nástroje dostupné ve Visual Studiu. [13]

Vývojové Prostředí	**Popularita**	**Kompatibilita**	**Funkcionality**	**Podpora Google**	**Uživatelské Rozhraní**
Android Studio	Vysoká	Android	Široká škála nástrojů	Ano	Přehledné a intuitivní
IntelliJ IDEA s pluginem pro vývoj Androidu	Vysoká	Multiplatformní	Podpora většiny funkcí Android Studia	Ne	Podobné Android Studiu
Eclipse s Android Development Tools (ADT) pluginem	Střední	Multiplatformní	Základní funkce pro vývoj Android aplikací	Ne	Standardní pro Eclipse
Visual Studio s Xamarin.Android	Střední	Multiplatformní	Použití jazyka C#, integrace s Visual Studiem	Ne	Standardní pro Visual Studio

Tabulka 3.2 Porovnání vlastností vývojových prostředí

3.4.3 Databáze

Pro vývoj moderní mobilní aplikace je volba vhodné databáze klíčovým rozhodnutím, které ovlivňuje nejen výkon aplikace, ale i její udržovatelnost, škálovatelnost a uživatelský zážitek. Pro tento projekt byly zvažovány následující databázové systémy.

3.4.3.1 SQLite

SQLite je vestavěný relační databázový systém, který je součástí systému Android. Jedná se o jednoduchý a lehký systém, který je ideální pro menší až středně velké aplikace. SQLite poskytuje podporu pro standardní SQL operace a databáze jsou uloženy v souborech na úložišti zařízení. [14]

3.4.3.2 Room

Room je knihovna poskytovaná Googlem jako součást Android Jetpack. Jedná se o abstrakci nad SQLite databází, která zjednodušuje práci s databází a poskytuje vysokoúrovňové API pro provádění operací s daty. Room obsahuje funkce jako je kompilace dotazů při sestavení aplikace, což pomáhá odhalit chyby v dotazech před spuštěním aplikace. [15]

3.4.3.3 Firestore

Firestore je cloudová NoSQL databáze poskytovaná společností Google jako součást Firebase platformy. Tato databáze umožňuje ukládání a synchronizaci dat v reálném čase mezi klienty a serverem. Firestore je vhodná pro aplikace, které vyžadují reálný časový přístup k datům, jako jsou například chatovací aplikace, sociální sítě, ale i aplikace s funkcí offline režimu. Firestore je flexibilní, škálovatelná a nabízí širokou škálu funkcí pro práci s daty. [16]

Databázový Systém	**Typ**	**Vhodnost pro Aplikace**	**Klíčové Vlastnosti**
SQLite	Relační	Menší až Středně Velké Aplikace	Vestavěný v systému Android – Podpora pro standardní SQL operace – Ukládání dat v souborech na úložišti
Room	Relační	Všechny Velikosti Aplikací	Abstrakce nad SQLite – Poskytuje vysokoúrovňové API pro práci s daty – Kompilace dotazů při sestavení
Firestore	NoSQL	Aplikace vyžadující Reálný Čas	Cloudová databáze s reálným časem – Synchronizace dat mezi klienty a serverem – Offline podpora, Integrace s

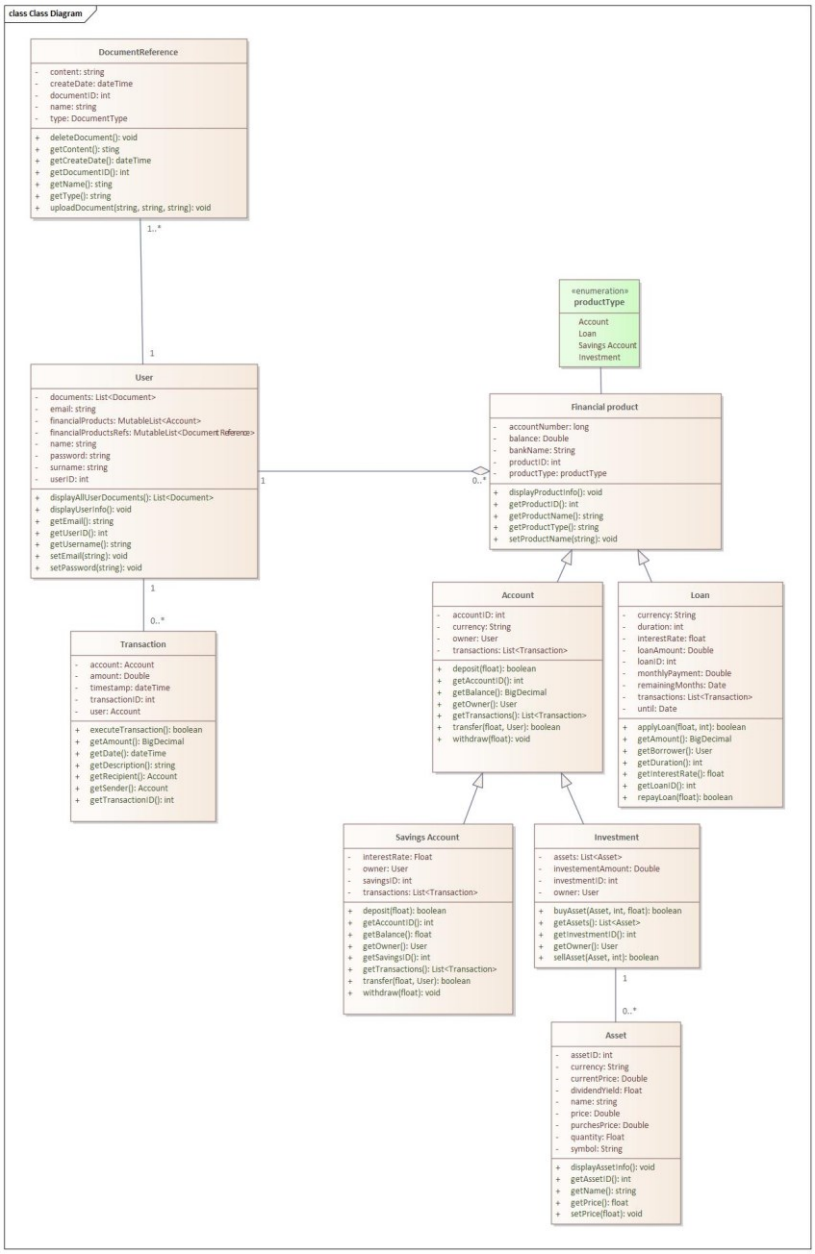
Tabulka 3.3: Porovnání vlastností dostupných databází

3.5 Datová analýza

V rámci této kapitoly byla provedena důkladná datová analýza, jejímž cílem bylo definovat všechny klíčové entity a jejich vzájemné vztahy, které jsou nezbytné pro fungování aplikace. Výsledkem této analýzy je vytvoření třídního diagramu, který znázorňuje strukturu systému a interakce mezi jednotlivými třídami.

3.6 Diagram tříd

Vypracovaný diagram obsahuje klíčové třídy, které reprezentují uživatele, jejich finanční produkty, transakce a související dokumenty. Níže je uveden popis jednotlivých tříd, jejich význam v systému a jejich atributy.



Obrázek 3.2: Diagram tříd

3.6.1 Promítnutí bankovních produktů v diagramu tříd

Během analýzy byly identifikovány čtyři hlavní typy bankovních produktů: osobní účet, spořící účet, půjčka a investice. Tyto produkty byly integrovány do třídního diagramu jako specializované třídy, které dědí společné vlastnosti z nadřazené třídy FinancialProduct.

Osobní účet (Account)

Osobní účet je v třídním diagramu reprezentován třídou Account, která dědí základní atributy a metody z třídy FinancialProduct. Třída Account obsahuje specifické atributy, jako je číslo účtu, měna a seznam transakcí, což umožňuje detailní sledování operací spojených s osobním účtem uživatele. Tyto atributy zajišťují, že každá transakce a finanční operace může být sledována a analyzována pro přesnou správu osobních financí.

Spořící účet (SavingsAccount)

Spořící účet je reprezentován třídou SavingsAccount, která rovněž dědí z třídy FinancialProduct. Tato třída přidává specifické atributy, jako je úroková sazba a identifikátor spořícího účtu (savingsID), což umožňuje správu a sledování úspor a úroků. Spořící účet je navržen tak, aby uživatelé mohli sledovat růst svých úspor díky připsaným úrokům, což je klíčové pro dlouhodobé finanční plánování.

Půjčka (Loan)

Půjčka je znázorněna třídou Loan, která dědí z třídy FinancialProduct. Třída Loan obsahuje atributy specifické pro půjčky, jako je měna, úroková sazba, částka půjčky, měsíční splátka a délka půjčky. Tato struktura umožňuje detailní správu a sledování půjček, včetně výpočtu splátek a úroků. Díky těmto atributům může systém přesně vypočítat splátkové kalendáře a zajistit, že uživatelé mají přehled o svých závazcích.

Investice (Investment)

Investice jsou reprezentovány třídou Investment, která také dědí z třídy FinancialProduct. Třída Investment obsahuje atributy a metody potřebné pro správu investičních produktů, jako jsou seznam aktiv, identifikátor investice (investmentID) a metody pro nákup a prodej aktiv. Tato třída umožňuje sledování investičního portfolia uživatele a správu jednotlivých aktiv. Investiční účet je navržen tak, aby uživatelé mohli monitorovat a optimalizovat své investice, sledovat výkonnost jednotlivých aktiv a přijímat informovaná investiční rozhodnutí.

Finanční produkty

FinancialProduct je abstraktní třída, která sdružuje společné vlastnosti všech finančních produktů. Každý finanční produkt má číslo účtu, zůstatek, název banky nebo finanční instituce, jedinečný identifikátor (productID) a typ produktu, který může být běžný účet, půjčka, spořicí účet nebo investice.

3.6.2 Ostatní systémové třídy

Níže je popsán zbytek datových tříd, které se netýkají financí, ale jsou důležité pro systém a aplikaci.

User (Uživatel)

Třída User reprezentuje jednotlivé uživatele systému. Každý uživatel má seznam dokumentů, které jsou s ním spojeny, e-mailovou adresu pro komunikaci a autentizaci, a heslo pro zabezpečení přístupu. Uživatel je identifikován svým jedinečným identifikátorem (userID), jménem a příjmením. Dále vlastní několik finančních produktů a má seznam referencí na dokumenty týkající se těchto produktů.

DocumentReference (Referenční dokument)

Třída DocumentReference představuje dokumenty, které jsou relevantní pro uživatele nebo jeho finanční produkty. Každý dokument má svůj obsah, jedinečný identifikátor (documentID), URI pro stažení a název, který usnadňuje jeho identifikaci.

Transaction (Transakce)

Třída Transaction reprezentuje finanční transakci spojenou s účtem. Každá transakce je přiřazena k určitému účtu a zahrnuje částku transakce, datum a čas jejího provedení, a jedinečný identifikátor (transactionID). Transakce je také spojena s uživatelem, který ji provedl. Tyto informace umožňují sledování a auditování finančních operací v systému.

3.7 Závěr datové analýzy

Tato datová analýza poskytuje strukturovaný a detailní pohled na požadavky systému a jejich implementaci v třídním diagramu. Definice tříd a jejich vzájemných vztahů umožňuje efektivní správu osobních financí uživatelů, včetně různých typů bankovních produktů a souvisejících dokumentů. Díky této analýze bylo možné vytvořit flexibilní a rozšiřitelný návrh, který podporuje různé aspekty finanční správy a zajišťuje intuitivní a uživatelsky přívětivé rozhraní.

3.8 Shrnutí

V této kapitole jsme se zaměřili na volbu technologií pro vývoj naší mobilní aplikace pro platformu Android. Představení a odůvodnění zvolených technologií je klíčové pro pochopení celkového procesu vývoje a jeho optimalizace pro dosažení co nejlepšího uživatelského zážitku.

Všechny technologie byly vybrány na základě důkladné analýzy jejich silných stránek a přínosu pro uživatelský zážitek.

3.8.1 Vývojové prostředí

Pro vývoj aplikace bylo použito vývojové prostředí Android Studio, které je přímo podporováno společností Google. Toto prostředí, ačkoli není multiplatformní, bylo zvoleno díky své vysoké intuitivnosti a široké škále integrovaných funkcí specializovaných pro vývoj aplikací na platformě Android. Android Studio nabízí rozsáhlou podporu, která usnadňuje práci vývojářům prostřednictvím nástrojů, jako je integrace Firebase přímo v IDE.

Nejbližší alternativou bylo vývojové prostředí IntelliJ IDEA od společnosti JetBrains, které je velmi podobné Android Studiu. Nicméně, IntelliJ IDEA postrádá přímou podporu od Google a některé specifické nástroje, které byly pro tento projekt klíčové, což vedlo k upřednostnění Android Studia.

3.8.2 Programovací jazyk

Pro vývoj prototypu aplikace byl zvolen programovací jazyk Kotlin. Kotlin je moderní jazyk vyvinutý speciálně pro vývoj aplikací na platformě Android a nabízí mnoho výhod a vylepšení oproti staršímu jazyku Java. Díky své podobné syntaxi a struktuře jako Java umožňuje Kotlin rychlé osvojení a efektivní práci, což je výhodné zejména pro nováčky v oboru.

3.8.3 Databáze

Firestore, jakožto součást Firebase platformy poskytované společností Google, byl zvolen pro správu databází. Hlavními důvody pro jeho volbu byla přímá integrace s Android Studiem a intuitivní použití. Firestore nabízí bezproblémovou integraci s vývojovým prostředím, což výrazně usnadňuje práci s databází a umožňuje rychlé a efektivní využití jeho funkcionalit během vývoje aplikace. Tato integrace a snadnost použití jeho API výrazně přispěly ke zkrácení času potřebného k implementaci databázových funkcí.

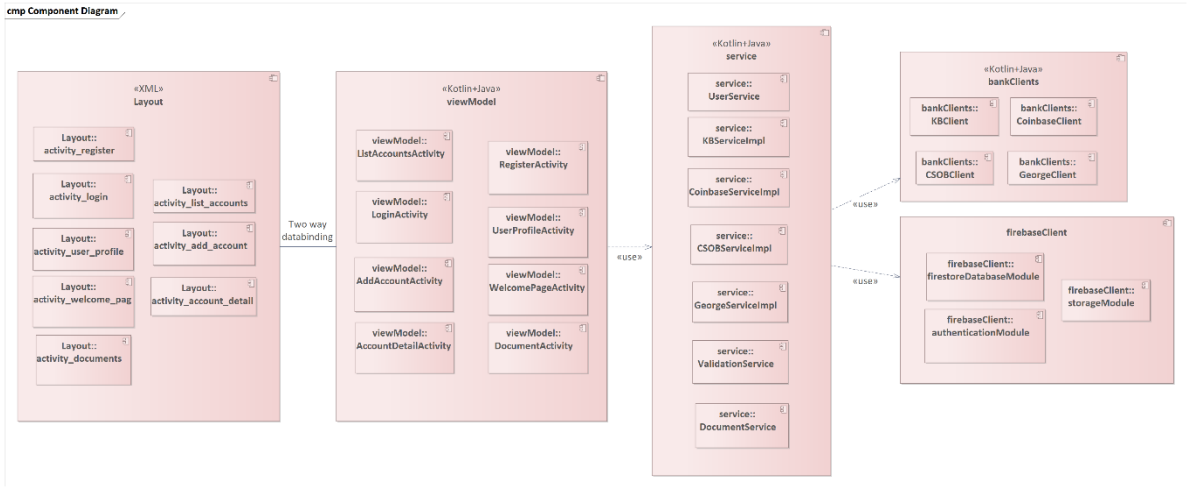
Kapitola 4

Návrh

V této kapitole se zaměříme na návrh softwarové architektury projektu, který zahrnuje strukturu systému a jeho hlavní komponenty. Hlavním cílem je ilustrovat, jak je aplikace pro správu uživatelských finančních produktů rozčleněna do jednotlivých částí, které spolu komunikují a zajišťují její celkovou funkčnost.

4.1 Softwarová architektura projektu

Diagram komponent zobrazuje strukturu systému, rozdělenou na jednotlivé komponenty, které spolu komunikují. Tento diagram ilustruje rozdělení funkcionality aplikace pro správu uživatelských finančních produktů do několika hlavních komponent, které zahrnují uživatelské rozhraní, ViewModel, služby a klienty.



Obrázek 4.1: Diagram komponent

4.1.1 Komponenta Layout

Komponenta Layout se skládá z různých XML souborů, které definují rozložení uživatelského rozhraní pro různé části aplikace. Každý layout je zodpovědný za specifickou obrazovku nebo stránku a zahrnuje různé UI elementy jako tlačítka, textová pole, seznamy a další vizuální komponenty. Tyto layouty poskytují uživatelům přehledné a intuitivní prostředí, které umožňuje interakci s aplikací a přístup k jejím funkcím.

4.1.2 Komponenta viewModel

Komponenta viewModel je zodpovědná za logiku a data, která podporují různé UI komponenty v Layout. Obsahuje třídy, které implementují logiku pro různé obrazovky a zajišťují interakci mezi uživatelským rozhraním a daty. ViewModely spravují stavy UI, provádějí operace s daty a komunikují se službami a databázemi, aby poskytovaly potřebná data uživatelům.

4.1.3 Komponenta service

Komponenta service obsahuje služby, které poskytují obchodní logiku a interakci s externími systémy nebo API. Tyto služby zahrnují různé implementace pro správu uživatelů, komunikaci s bankovními systémy, validaci dat a správu dokumentů. Služby slouží jako prostředníci mezi viewModely a externími systémy, poskytují data, zpracovávají požadavky a provádějí různé operace na základě obchodní logiky aplikace.

4.1.4 Komponenta bankClients

Komponenta bankClients obsahuje klienty, kteří se připojují a komunikují s různými bankovními systémy. Tyto klienty zahrnují implementace pro různé banky, jako jsou KB, Coinbase, CSOB a George. Každý klient je zodpovědný za správnou komunikaci s příslušným bankovním API, zajištění autentizace, načítání dat a provádění transakcí, což umožňuje aplikaci poskytovat uživatelům integrované bankovní služby.

4.1.5 Komponenta firebaseClient

Komponenta firebaseClient obsahuje moduly pro interakci s platformou Firebase. Firebase je platforma pro vývoj mobilních a webových aplikací, která poskytuje různé služby jako databáze, autentizace a úložiště. Tato komponenta umožňuje aplikaci využívat Firebase pro ukládání a načítání dat, správu uživatelských účtů a autentizaci, a ukládání souborů do cloudového úložiště.

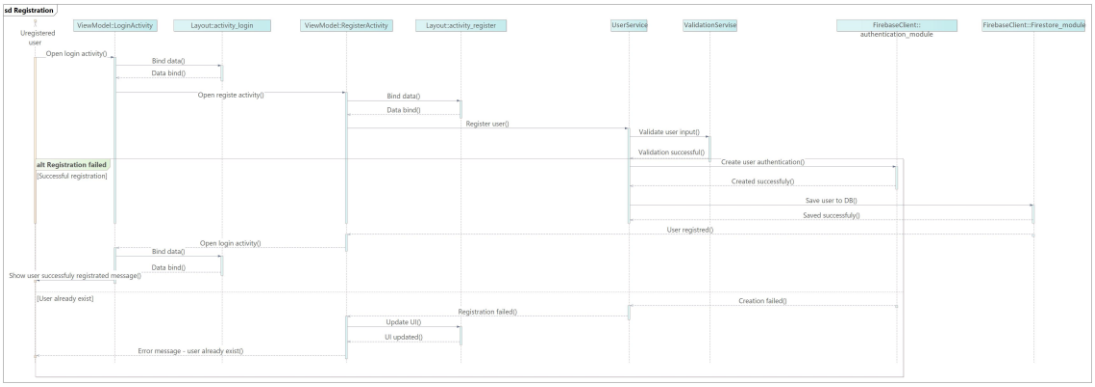
4.2 Sekvenční diagram

Sekvenční diagram zobrazuje interakce mezi jednotlivými komponentami systému v časovém sledu, konkrétně pro klíčové scénáře aplikace pro sledování bankovních účtů a finančních operací. Diagram zahrnuje pět hlavních průchodů, které byly identifikovány jako nejdůležitější pro funkčnost aplikace: Registrace, Přihlášení, Zobrazení seznamu účtů, Přidání účtu a Zobrazení dokumentů. Kvůli moc velkému celkovému diagramu jsem byl nucen digram roztrhnout na menší digramy, které se týkají přímo výše zmíněných průchodů a jejich alternativních scénářů.

4.2.1 Hlavní průchody

Registrace

Uživatel spustí přihlášení, následně spustí registraci prostřednictvím uživatelského rozhraní, zadá požadované údaje (jméno, příjmení, e-mail, heslo), které jsou následně validovány pomocí ValidationService. Po úspěšné validaci jsou údaje předány UserService, která vytvoří uživatele v Autentizačním modulu Firestore a následně uloží uživatelská data do Firestore.



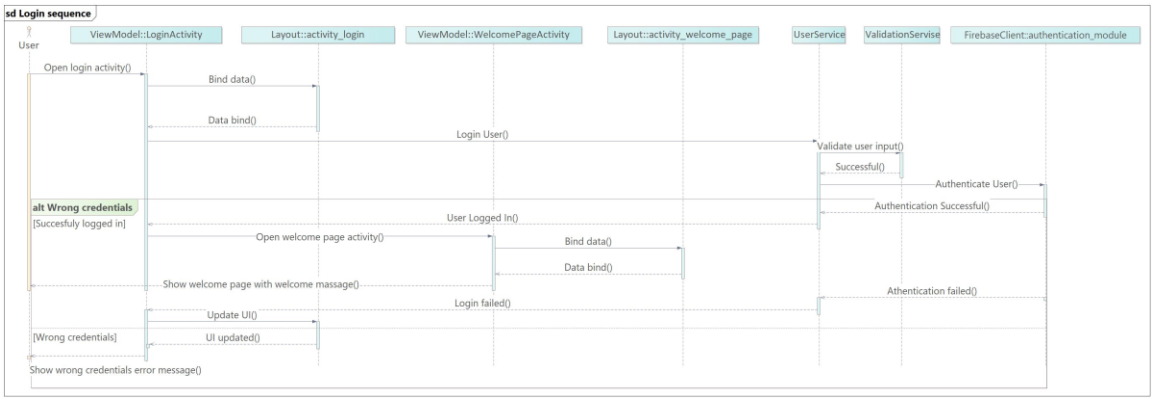
Obrázek 4.2: Sekvenční diagram průchodu Registrací

Alternativní scénáře: Pokud již uživatel existuje autentizační modul to vrátí jako chybu a uživatelovy vyskočí okénko se zprávou „Uživatel již existuje“

Přihlášení

Uživatel zadá své přihlašovací údaje (e-mail a heslo) do uživatelského rozhraní, které jsou předány ValidationService k ověření. Pokud jsou údaje v pořádku, UserService ověří uživatele proti záznamům ve Firebase autentizačním modulu a umožní přístup do aplikace a zobrazí aktivitu WelcomePage.

Alternativní scénáře: Při neúspěšném ověření přihlašovacích údajů je uživatel informován o nesprávném e-mailu nebo heslu.

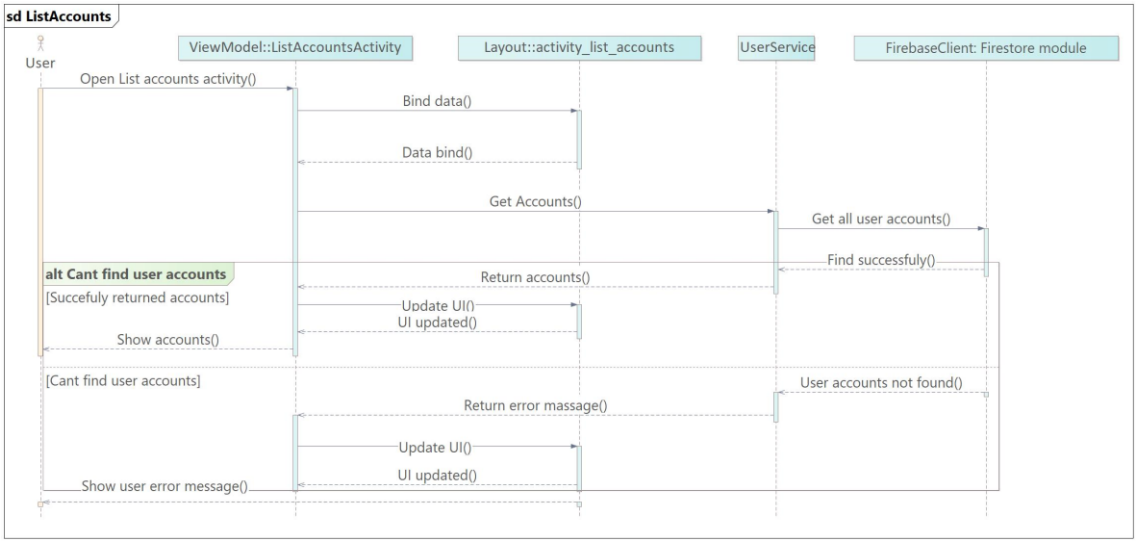


Obrázek 4.3: Sekvenční diagram průchodu Přihlášení

Zobrazení seznamu účtů

Po přihlášení může uživatel zobrazit seznam svých bankovních účtů. Uživatelské rozhraní požádá UserService o načtení účtů z Firestore, které jsou následně zobrazeny prostřednictvím ViewModel v uživatelském rozhraní.

Alternativní scénáře: Pokud nejsou nalezeny žádné účty nebo dojde k chybě při načítání, je uživatel informován o situaci.

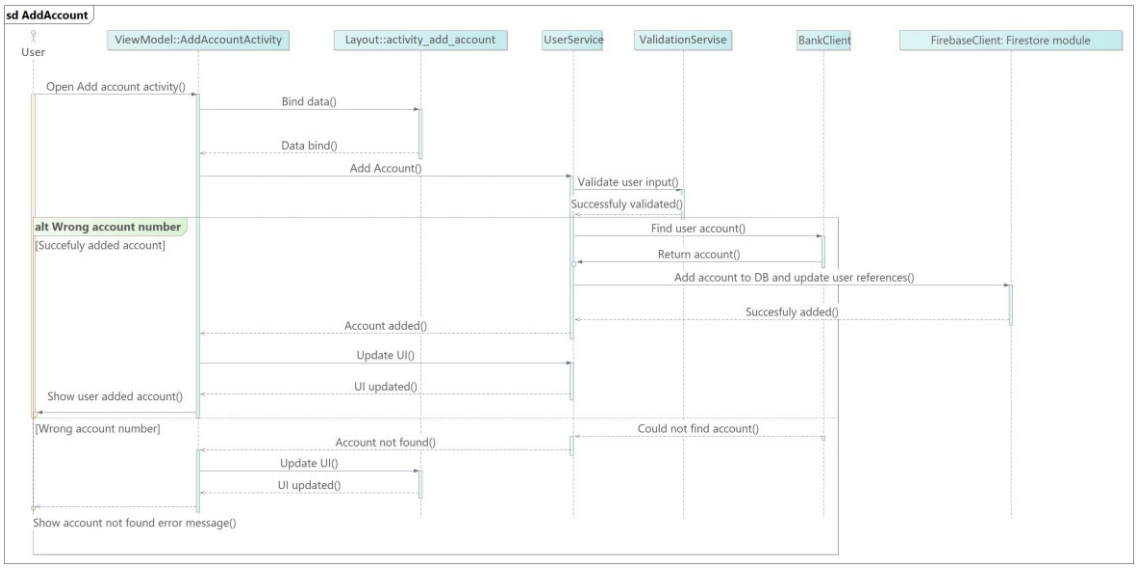


Obrázek 4.4: Sekvenční diagram průchodu Zobrazení si účtů

Přidání účtu

Uživatel má možnost přidat nový bankovní účet. Po zadání údajů o účtu jsou tyto údaje validovány pomocí ValidationService. Po úspěšné validaci UserService uloží nové údaje do Firestore.

Alternativní scénáře: Při zadání neplatných údajů je uživatel upozorněn na chybu a je mu zobrazena chybová hláška „Účet nebyl nalezen“.

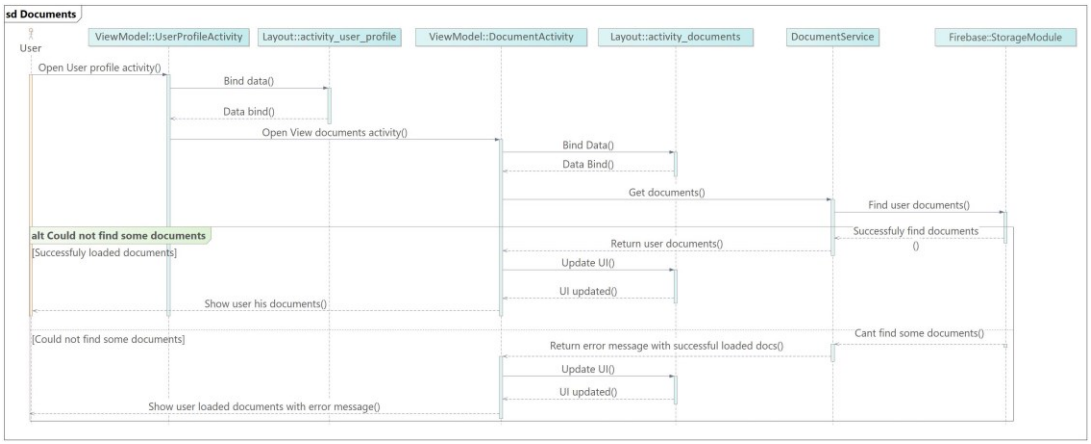


Obrázek 4.5: Sekvenční diagram průchodu přidání účtu

Zobrazení dokumentů

Uživatel může zobrazit seznam svých nahraných dokumentů. Uživatelské rozhraní požádá UserService o načtení dokumentů z Firestore, které jsou následně zobrazeny prostřednictvím ViewModel v uživatelském rozhraní.

Alternativní scénáře: Pokud nejsou nalezeny žádné dokumenty nebo dojde k chybě při načítání, je uživatel informován a je mu zobrazena informační zpráva že účet nelze načíst.



Obrázek 4.6: Sekvenční diagram průchodu zobrazení dokumentů

4.3 Shrnutí

Tato kapitola poskytla detailní přehled o návrhu softwarové architektury a hlavních scénářích interakcí v aplikaci. Díky jasnému rozdělení systému na jednotlivé komponenty a detailnímu popisu klíčových scénářů je možné lépe pochopit, jak aplikace funguje a jak různé části spolupracují na poskytování služeb uživatelům. Tento návrh zajišťuje flexibilitu, škálovatelnost a udržovatelnost kódu, což je klíčové pro dlouhodobý úspěch a rozvoj aplikace.

Kapitola 5

Implementace

Implementační fáze představuje klíčový bod v procesu vývoje mobilní aplikace pro sledování bankovních účtů a finančních operací. V této kapitole podrobně rozebereme postup implementace jednotlivých funkcí a prvků navržených v předchozích fázích vývoje, včetně technologií použitých k vytvoření mobilní aplikace. Dále se zaměříme na integraci s bankovními API a testování aplikace před uvedením do provozu. Implementace představuje klíčový krok k dosažení cílů stanovených v rámci této bakalářské práce a umožňuje praktickou realizaci navrženého řešení pro uživatele.

5.1 Architektura

Architektonický styl MVVM [17] (Model-View-ViewModel) byl zvolen pro návrh aplikace. Tato architektura je známá svou schopností oddělit prezentaci dat od jejich logiky a zajišťuje tak lepší organizaci a udržovatelnost kódu.

Model reprezentuje datovou vrstvu aplikace, která obsahuje podnikovou logiku a datové operace. V našem případě modely obsahují třídy a funkce pro práci s daty získanými z databáze Firestore, manipulaci s nimi a poskytování potřebných dat pro uživatelské rozhraní.

View je vizuální vrstva aplikace, která zobrazuje uživateli grafické rozhraní a interaguje s uživatelem. V naší aplikaci jsou Views reprezentovány uživatelskými obrazovkami, které jsou vytvořeny pomocí XML layoutů v kombinaci s Kotlin kódem pro obsluhu událostí a aktualizaci zobrazení.

ViewModel slouží jako spojovací článek mezi View a Modelem. Obsahuje logiku pro zpracování dat mezi View a Modelem a udržuje stav uživatelského rozhraní nezávislý na životním cyklu aktivity nebo fragmentu. V naší aplikaci jsou ViewModely navrženy tak, aby poskytovaly potřebná data z Modelu do View a zajišťovaly aktualizaci zobrazení v souladu s daty získanými z databáze Firestore.

Takto strukturovaná architektura MVVM poskytuje čistý a oddělený kód, což usnadňuje testování, údržbu a další rozvoj aplikace.

5.2 Verzovací systém

Jako verzovací systém jsem zvolil GitLab [18], webovou platformu pro správu verzí, která je postavena na systému Git. Důvodem pro volbu GitLabu byla jeho dostupnost ve školním prostředí a již existující zkušenosti s touto platformou při tvorbě školních projektů.

V porovnání s jinou populární platformou pro správu verzí, GitHubem, nabízí GitLab podobné funkce, ale s několika rozdíly. GitHub umožňuje zdarma hostovat veřejné repositáře a zpoplatněné soukromé repositáře, zatímco GitLab poskytuje možnost zdarma hostovat jak veřejné, tak i soukromé repositáře. To může být výhodné pro školní nebo soukromé projekty, kde není žádoucí, aby byl kód veřejně dostupný.

5.3 Návrh GUI

Celý prototyp byl nejdříve vypracován jako HighFidelity prototyp v nástroji Figma. Tento návrh byl otestován ještě před přechodem na přímou implementaci aplikace. Vzhled HighFidelity prototypu je ukázán na obrázku 5.1.

Následně při tvoření a implementaci již funkčního prototypu aplikace jsem se řídil Android Developers guides [19] a byl zde brán ohled na „Deset heuristik použitelnosti“ od profesora Jakoba Nielsena [20]. Příklad vzhledu již implementovaného prototypu aplikace je na viditelný na obrázku 5.2.

V prototypu jsem zvolil 4 hlavní barvy:

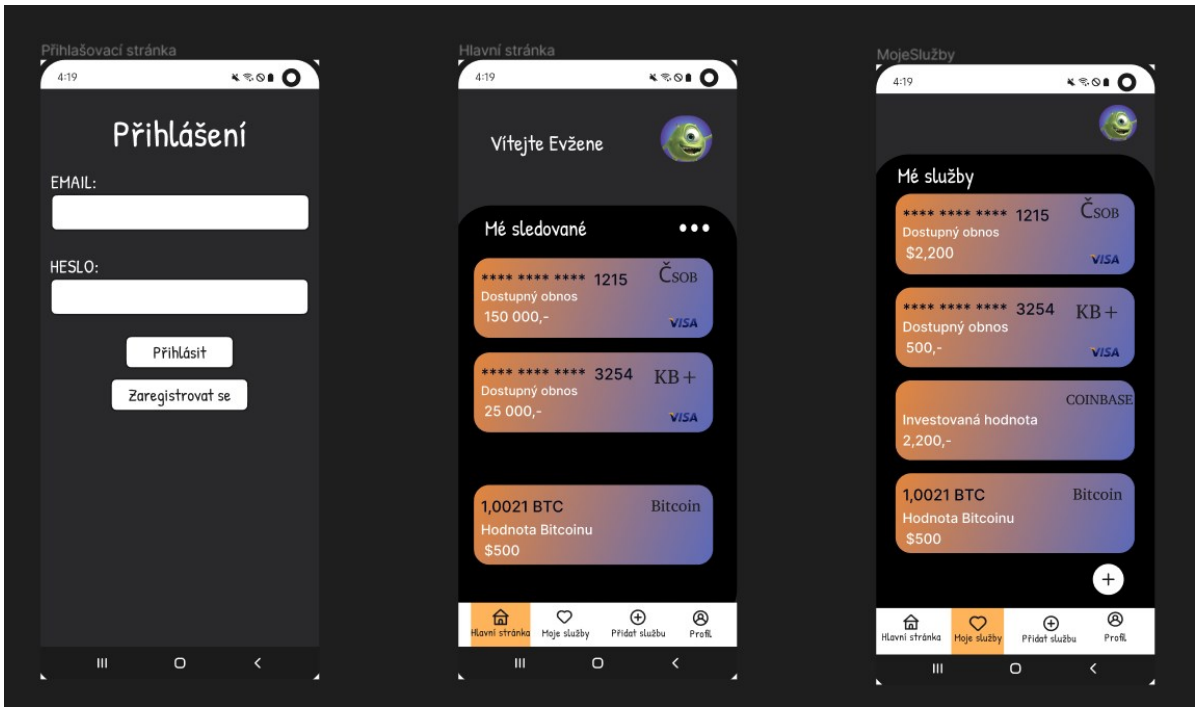
- Hlavní barva - #212121 (Tmavě šedá) Pozadí celé aplikace
- #000000 (Černá barva) Prvek na pozadí – Zaoblený obdélník, text v navigačním menu a text v položce účtu.
- #FFFFFFFF (Bílá barva) Text a navigační menu.
- Začátek #E18640(Oranžová) a konec (Modrá) prolínající se barva – Zobrazení účtů a transakcí

Hlavní barva je tmavě šedá, protože tmavý režim používá až 82 % lidí [21] a také je to neutrální barva, a nevztahuje se na ni Ittenův kruh výběru primárních, sekundárních a terciálních barev ale ani nemohou narušit harmonii tohoto kruhu [22].

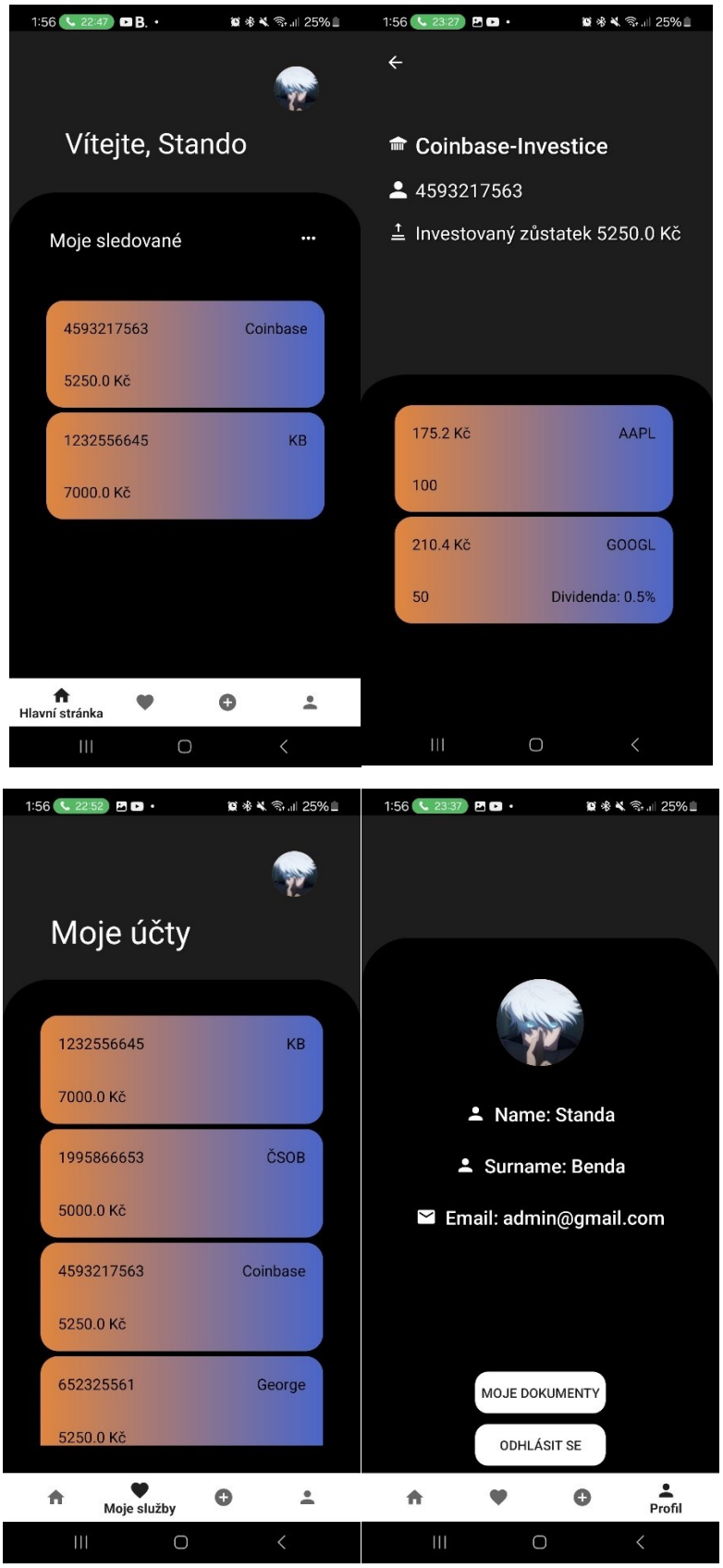
Dále byl použit černý prvek, který na sebe upoutá hned pozornost uživatele a má veliký kontrast čili je na šedé barvě dobře vidět.

Následně byla vybrána primární barva Modrá a sekundární barva oranžová jako dvě barvy gradientu pro zobrazení prvku účtu. Na tyto dvě barvy byl již použit již zmiňovaný barevný kruh.

Také byly do aplikace přidané plynulé animace při zobrazování tlačítek, přechodu mezi aktivitami pomocí navigačního panelu.



Obrázek 5.1: Ukázka z HighFidelity prototypu



Obrázek 5.2: Série obrázku z již implementované aplikace

5.4 Validace vstupních dat

Validace vstupních dat je klíčovým aspektem každé aplikace, která pracuje s uživatelskými daty, zejména v oblasti finančních služeb. V naší aplikaci jsme se zaměřili na robustní validaci vstupních dat při přihlašování a registraci uživatelů, aby byla zajištěna bezpečnost a správnost zadaných informací. Pro tento účel jsme implementovali třídu ValidationService, která poskytuje metody pro ověření různých typů dat.

5.4.1 Ověřování neprázdného řetězce

Prvním krokem ve validaci vstupních dat je zajištění, že žádné pole nezůstane prázdné. Pro tento účel používáme metodu, která ověřuje, zda je vstupní řetězec neprázdný. Tato metoda vrací hodnotu, která signalizuje, zda vstupní řetězec není **null** a není prázdný. Tento krok je základní prevencí proti zadávání prázdných polí, která by mohla způsobit chyby nebo nekonzistentní data v systému.

5.4.2 Ověřování e-mailové adresy

Dalším krokem je ověření, že uživatel zadal platnou e-mailovou adresu. Pro tento účel jsme implementovali metodu, která používá regulární výraz k ověření formátu e-mailové adresy. Tato metoda zajišťuje, že e-mailová adresa má správný formát, což je nezbytné pro následnou komunikaci s uživatelem a pro přihlášení do systému.

5.4.3 Ověřování hesla

Bezpečnost hesla je kritickým faktorem při registraci uživatele. Naše metoda ověřuje, zda heslo splňuje požadavky na složitost, jako je délka minimálně 8 znaků, obsahuje číslice, malá a velká písmena a speciální znaky. Tímto způsobem zajišťujeme, že uživatelská hesla jsou dostatečně silná, aby odolala pokusům o neoprávněný přístup.

5.4.4 Ověřování čísla účtu

Pro validaci čísla účtu používáme metodu, která ověřuje, zda číslo účtu odpovídá požadovanému formátu 10 číslic. Tato validace zajišťuje, že čísla účtů jsou konzistentní a správná, což je klíčové pro finanční transakce a správu účtů.

5.4.5 Ověřování poskytovatele služby

V neposlední řadě je důležité ověřit, že zadaný poskytovatel služby je platný. Metoda porovnává zadaného poskytovatele s předdefinovaným seznamem platných poskytovatelů. Tato metoda je klíčová pro zajištění, že uživatelé zadávají pouze povolené poskytovatele služeb, což pomáhá udržovat integritu a bezpečnost systému.

5.5 Zpracování chyb

V celé aplikaci byly implementovány robustní mechanismy pro zpracování chyb, které zajišťují, že uživatelé jsou informováni o případných problémech a aplikace může správně reagovat na různé situace. Zpracování chyb se provádí především prostřednictvím zobrazení chybových zpráv uživatelům, což zajišťuje lepší uživatelský zážitek a pomáhá uživatelům rychle pochopit, co se stalo špatně. Tyto chybové hlášky jsou zobrazené v českém jazyce tudíž se zde neřeší vícejazyčnost a jsou natvrdo vepsané do kódu aplikace.

Pro zpracování chyb používáme následující přístupy:

1. **Použití Toast zpráv:** Při neúspěšných operacích informujeme uživatele prostřednictvím Toast zpráv. Tyto zprávy poskytují okamžitou zpětnou vazbu o chybách, které se v aplikaci vyskytly.
2. **Logování chyb:** Chyby jsou také logovány pomocí exception.printStackTrace(), což usnadňuje ladění a identifikaci problémů během vývoje a testování aplikace.
3. Podmíněné zpracování úspěchu a neúspěchu: Každá asynchronní operace (např. přihlášení uživatele, nahrávání dat do databáze) má definované způsoby zpracování jak pro úspěch, tak pro neúspěch operace, aby bylo zajištěno, že aplikace bude správně reagovat v obou případech.

5.5.1 Příklady zpracování chyb v kódu

Příklad 1: Přihlašování uživatelů

Při přihlašování uživatelů zpracováváme chyby pomocí **Toast** zpráv, které informují uživatele o případných problémech, jako je například nesprávné přihlašovací údaje.

```
userService.loginUser(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {

            val rememberMeChecked = binding.rememberMeCheckBox.isChecked
            if (rememberMeChecked) {
                val editor = sharedPreferences.edit()
                editor.putBoolean("isLoggedIn", true)

                val userId = userService.getAuthenticatedUserId()
                editor.putString("userId", userId)
                editor.apply()
            }
            val userId = userService.getAuthenticatedUserId()
            val intent = Intent( packageContext: this, WelcomePageActivity::class.java)
            intent.putExtra( name: "userId", userId)
            startActivity(intent)
            finish()
        } else {
            Toast.makeText(
                baseContext, text: "Špatný email nebo heslo.",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
```

Obrázek 5.3: Ukázka kódu, kde se řeší zachytávání chyb

Příklad 2: Nahrávání souborů

Při nahrávání souborů do databáze zpracováváme chyby tak, že informujeme uživatele o neúspěšné operaci pomocí **Toast** zpráv a logujeme výjimku pomocí **printStackTrace()**.

```
private fun uploadFileToFirebaseStorage(fileUri: Uri) {
    val userId = intent.getStringExtra( name: "userId")
    if (userId != null) {
        documentService.uploadFileToFirebaseStorage(userId, fileUri,
            onSuccess = { it: Uri
                Toast.makeText( context: this,  text: "Soubor byl úspěšně nahrán", Toast.LENGTH_SHORT).show()
            },
            onFailure = { exception ->
                exception.printStackTrace()
                Toast.makeText( context: this,  text: "Nahrání souboru se nepovedlo", Toast.LENGTH_SHORT).show()
            }
        )
    }
    recreate()
}
```

Obrázek 5.4: Ukázka kódu, kde se řeší zachytávání chyby a logování

5.6 Řešení problémů pomocí design patternů

Při vývoji aplikace jsem narazil na několik problémů, které bylo potřeba vyřešit pro zajištění flexibility, škálovatelnosti a udržovatelnosti kódu. Pro řešení těchto problémů jsme využili několik design patternů, konkrétně Adapter, Bridge a Decorator. Všechny problémy, které těmito patterny byly řešeny jsou níže popsány.

5.6.1 Adapter

Design pattern Adapter byl použit zejména při práci s RecyclerView pro zobrazení seznamů účtů v uživatelském rozhraní. Vzhledem k tomu, že data pro tyto účty byla strukturována různými způsoby, bylo potřeba najít způsob, jak je jednotně zobrazit. Adapter pattern umožnil přizpůsobit data z různých zdrojů do podoby, která je kompatibilní s RecyclerView. Tím bylo dosaženo jednotného zobrazení účtů bez nutnosti změn v základním kódu. [23]

5.6.2 Bridge

Dalším klíčovým patternem, který byl použit, byl Bridge. Tento pattern byl implementován pro oddělení rozhraní bankovních služeb od jejich konkrétních implementací. Tímto způsobem byla získána flexibilita přidávat nové bankovní služby bez nutnosti měnit stávající kód. Bridge pattern umožnil oddělit abstrakci od její implementace, což vedlo k nezávislému vývoji jednotlivých částí systému a zlepšilo udržovatelnost a rozšiřitelnost aplikace. [24]

5.6.3 Decorator

Kromě těchto dvou výše zmíněných patternů byl také využit Decorator, který pomohl přidat funkci pro dynamické odsazení položek v seznamu účtů v RecyclerView. Tato funkcionalita byla přidána bez nutnosti změny základního kódu položek, což zajistilo čistý a udržovatelný kód. Decorator pattern umožnil rozšířit funkcionalitu objektů, aniž by bylo nutné zasahovat do jejich základní struktury. [25]

5.7 Bankovní APIs

V této části projektu jsme se zabývali propojením aplikace s bankovními API. V reálném projektu by bylo nutné získat certifikáty a přístupy pro integraci s bankovními systémy, což by mohlo být časově náročné. Proto jsme se rozhodli vytvořit vlastní bankovní klienty, které simulují komunikaci s bankovními servisy. Tento přístup nám umožnil efektivně testovat a vyvíjet aplikaci, aniž bychom museli čekat na schválení a přístup k reálným bankovním API. V následujících podkapitolách je popsána implementace klientů, struktura bankovních API a konkrétní implementace bankovních klientů.

5.7.1 Bankovní klienti

Bankovní klienty jsem implementoval specificky pro práci s bankovními APIs. Čili pracují s JSON souborem, a tedy mapují pomocí knihovny Jackson [26] struktury v JSON souboru přímo na objekty definované v modelu, a to vždy na ten typ, který je v souboru zvoleného účtu.

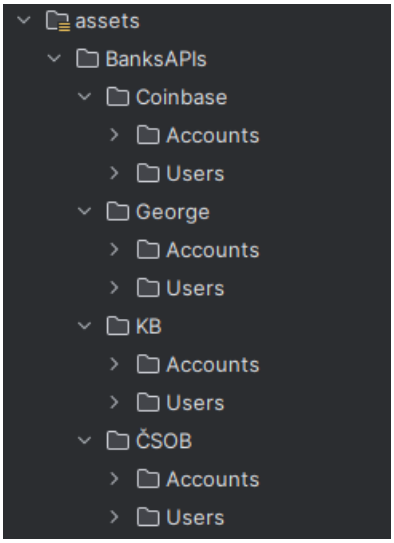
V aplikaci jako takové se pak k nim přistupuje pomocí Bridge design patternu který spojuje servisovou vrstvu pro banky s klientskou vrstvou.

```
open class KbClient(private val context: Context): Client {
    private val accountTypes: Map<String, KClass<out FinancialProduct>> = mapOf(
        "Account" to Account::class,
        "Savings" to SavingsAccount::class,
        "Loan" to Loan::class,
        "Investment" to Investment::class
    )

    bendasta
    override fun getUserAccounts(userId: String): List<Account> {
        val jsonFile = File( pathname: "BanksAPIs/KB/Users/$userId.json")
        val json = jsonFile.readText()

        val mapper = jacksonObjectMapper()
        val userData: Map<String, Any> = mapper.readValue(json)
        val accountDataList: List<Map<String, Any>> = userData["accounts"] as List<Map<String, Any>>

        val accounts = mutableListOf<Account>()
        for (data in accountDataList) {
            val accountDescription = data["account_type"] as String
            val accountClass = accountTypes[accountDescription.lowercase()]
            val account = accountClass?.java?.getDeclaredConstructor()?.newInstance() as Account?
            if (account != null) {
                accounts.add(account)
            }
        }

        return accounts
    }
```

Obrázek 5.5: Ukázka konkrétního bankovního klienta (KBClient)

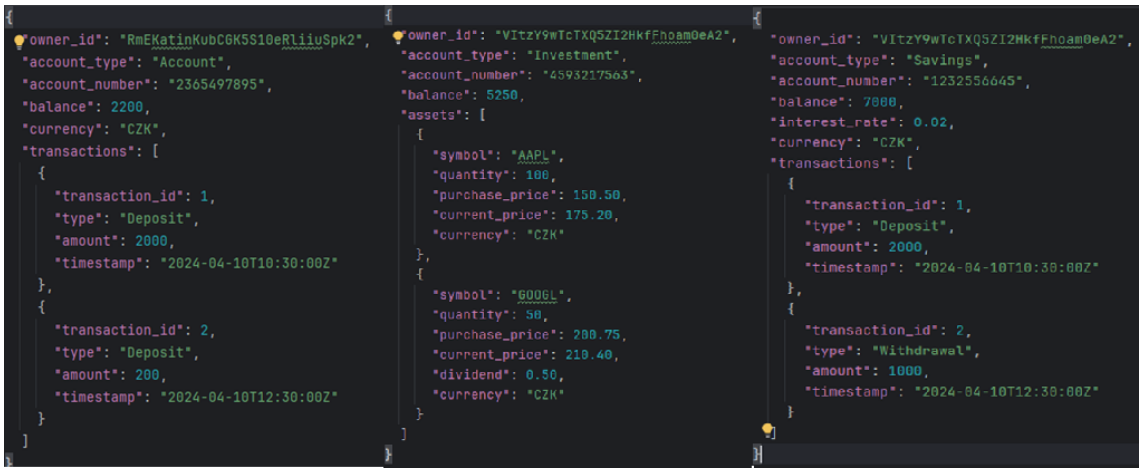
5.7.2 Struktura

Struktura bankovních APIs vypadá následovně:

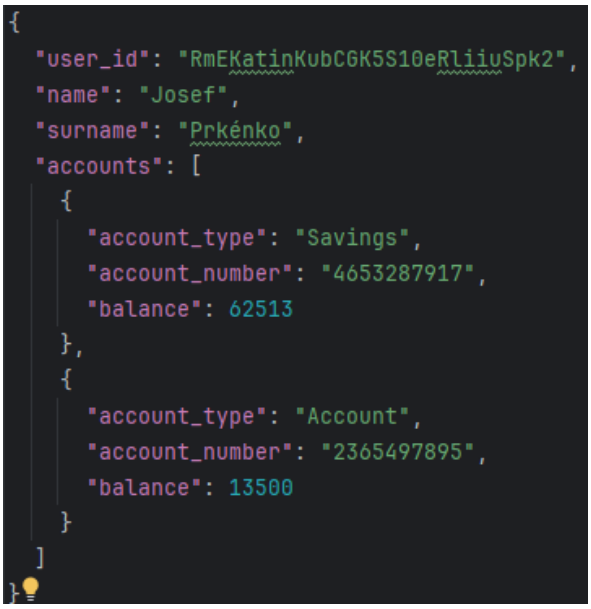


Obrázek 5.6: Struktura bankovních APIs

V každé složce „Accounts“ je JSON soubor, který se jmenuje podle čísla účtu. Každý typ účtu má svoji JSON strukturu, která se liší v atributech, které jsou navíc nebo rozdílné od ostatních entit v modelu, jak můžete vidět na obrázku číslo 5.5. Zato JSON soubor v „Users“, který je vyobrazen na obrázku 5.6 je pojmenován po UID každého uživatele ve Firebase autentizačním modulu a struktura těchto souborů je totožná.



Obrázek 5.7: Ukázka struktury bankovních účtů



Obrázek 5.8: Ukázka struktury uživatele

5.8 Shrnutí

V této kapitole byla podrobně rozebrána implementační fáze vývoje mobilní aplikace pro sledování bankovních účtů a finančních operací. Byly popsány technologie, které byly zvoleny pro vývoj aplikace, včetně vývojového prostředí (Android Studio), programovacího jazyka (Kotlin), databáze (Firestore), architektury (MVVM) a verzovacího systému (GitLab).

Dále byl popsán návrh uživatelského rozhraní (GUI), který byl vytvořen s ohledem na heuristiky použitelnosti a zohlednění preference uživatelů pro tmavý režim. Byly diskutovány hlavní barvy použité v aplikaci a důvody jejich výběru, stejně jako implementace plynulých animací pro zlepšení uživatelského zážitku.

Kapitolou byla také popsána validace vstupních dat, která zajišťuje bezpečnost a správnost zadaných informací při přihlašování a registraci uživatelů. Byly uvedeny metody pro ověřování neprázdných řetězců, e-mailových adres, hesel, čísel účtů a poskytovatelů služeb.

Mechanismy pro zpracování chyb byly implementovány tak, aby uživatelé byli informováni o případných problémech a aplikace mohla správně reagovat na různé situace. Byly uvedeny konkrétní příklady zpracování chyb při přihlašování uživatelů a nahrávání souborů.

Design patterny, jako Adapter a Bridge, byly použity pro zajištění flexibility, škálovatelnosti a udržovatelnosti kódu. Adapter pattern byl využit při práci s RecyclerView pro zobrazení seznamů účtů, zatímco Bridge pattern umožnil oddělení rozhraní pro bankovní služby od jejich konkrétních implementací.

Poslední část kapitoly se zaměřila na integraci s bankovními API. Bylo vysvětleno, jak byly implementovány bankovní klienty, které simulují komunikaci s bankovními servisy, a jak byla strukturována bankovní API. Tímto způsobem bylo možné efektivně testovat a vyvíjet aplikaci bez nutnosti čekat na schválení a přístup k reálným bankovním API.

Celkově tato kapitola poskytla komplexní pohled na implementační fázi vývoje aplikace, včetně technologií, návrhu, validace dat, zpracování chyb a použití design patternů.

Kapitola 6

Testování

Testování je nezbytnou součástí vývoje softwaru, která zajišťuje správné fungování aplikace a minimalizuje výskyt chyb. V této kapitole se zaměříme na testovací scénáře, které byly vytvořeny pro různé části naší aplikace.

6.1 Testování funkcionality servisní vrstvy

Prototyp aplikace byl otestován řadou testů. Všechny z níže uvedených testů měly pouze integrační testy své testovací scénáře, které budou v následující podkapitole popsány. Pro testování byly použity tyto tipy testů [27]:

6.1.1 Integrační testy

Integrační testy probíhaly na vrstvě servisové abych zjistil, jestli jsou všechny servisy funkční a správně propojené. Pro integrační testy nebyla použita žádná testovací databáze a bylo jimi pokryto 85% servisní vrstvy.

6.1.2 Smoke testy

Tyto testy jsem používal nejčastěji, a to z důvodu otestování nových aktivit či pouze jen přidaných funkcionalit. Testy probíhaly tak, že jsem například při přidání aktivity přidání účtu nejdříve proklikal všechny aktivity související s danou funkcionalitou a následně, pokud aplikace nespadla či nevyhodila nežádoucí chybu jsem přistoupil na testování funkcionality jako takové. Takto jsem postupoval u každé aktivity, čí funkcionality, která byla implementována.

6.1.3 Alpha a beta testy

Tento druh testů jsem prováděl já a pár mých kamarádů. Alpha testy jako takové byly provedeny na počátku vytváření prototypu aplikace, aby se vyladily nepřesnosti a detaily ke grafickému rozhraní. Beta testy se pak prováděly až na úplném závěru implementace, a to na plně funkčním prototypu aplikaci. [28]

6.2 Testovací scénáře

Testovací scénáře byly vytvořeny pro integrační testy servisové vrstvy. Níže budou popsány testy UserService a konkrétně jedné (KBService) bankovní servisy. Bankovní servisa zde bude popsána jedna z důvodu opakujících se scénářů. DocumentService byla otestována uživatelskými testy, proto zde nefiguruje.

6.2.1 Integrační testy UserService

TC1: Vytvoření uživatele (testCreateUserIntegration)

Cíl: Ověřit, že služba umožňuje správné vytvoření nového uživatele s danými údaji.

Předpoklady:

- Žádný uživatel s e-mailem testuser1@example.com neexistuje.

Testovací data:

- Jméno: John
- Příjmení: Doe
- E-mail: testuser1@example.com
- Heslo: testpassword

Kroky:

1. Vytvořte uživatele s jménem "John", příjmením "Doe", e-mailem testuser1@example.com a heslem testpassword.
2. Získejte údaje o nově vytvořeném uživateli.

Očekávaný výsledek:

- Uživatel s výše uvedenými údaji bude vytvořen úspěšně.
- Údaje o vytvořeném uživateli budou odpovídat zadaným hodnotám (jméno, příjmení, e-mail).

Aktuální výsledek:

- Vytvoření uživatele proběhne úspěšně.
- Údaje o vytvořeném uživateli odpovídají zadaným hodnotám.

TC2: Přihlášení uživatele (testLoginUserIntegration)

Cíl: Ověřit, že služba umožňuje uživateli úspěšné přihlášení pomocí správných přihlašovacích údajů.

Předpoklady:

- Uživatel s e-mailem testuser2@example.com existuje a má heslo testpassword.

Testovací data:

- E-mail: testuser2@example.com
- Heslo: testpassword

Kroky:

1. Přihlaste se pomocí e-mailu testuser2@example.com a hesla testpassword.

Očekávaný výsledek:

- Uživatel se úspěšně přihlásí pomocí zadaného e-mailu a hesla.
- Přihlášení proběhne úspěšně.

Aktuální výsledek:

- Přihlášení proběhne úspěšně.

TC3: Získání ID aktuálně přihlášeného uživatele (testGetAuthenticatedUserId)

Cíl: Ověřit, že služba umožňuje správné získání ID aktuálně přihlášeného uživatele.

Předpoklady:

- Uživatel s e-mailem testuser3@example.com existuje a má heslo testpassword.

Testovací data:

- E-mail: testuser3@example.com
- Heslo: testpassword

Kroky:

1. Přihlaste se pomocí e-mailu testuser3@example.com a hesla testpassword.
2. Získejte ID aktuálně přihlášeného uživatele.

Očekávaný výsledek:

- Získání ID aktuálně přihlášeného uživatele proběhne úspěšně.
- Získané ID se shoduje s ID aktuálně přihlášeného uživatele.

Aktuální výsledek:

- Získání ID aktuálně přihlášeného uživatele proběhne úspěšně.
- Získané ID se shoduje s ID aktuálně přihlášeného uživatele.

TC4: Získání účtů uživatele (testGetUserAccounts)

Cíl: Ověřit, že služba umožňuje správné získání seznamu účtů přihlášeného uživatele.

Předpoklady:

- Uživatel s e-mailem Pepik@gmail.com existuje a má heslo Heslo.123456.
- Uživatel má účet s číslem 6511231111 a poskytovatelem ČSOB.

Testovací data:

- E-mail: Pepik@gmail.com
- Heslo: Heslo.123456
- Číslo účtu: 6511231111
- Poskytovatel služeb: ČSOB

Kroky:

1. Přihlaste se pomocí e-mailu Pepik@gmail.com a hesla Heslo.123456.
2. Přidejte účet uživatele s číslem 6511231111 a poskytovatelem ČSOB.
3. Získejte účty uživatele.

Očekávaný výsledek:

- Získání účtů uživatele proběhne úspěšně.
- Seznam získaných účtů odpovídá očekávanému seznamu.

Aktuální výsledek:

- Získání účtů uživatele proběhne úspěšně.
- Seznam získaných účtů odpovídá očekávanému seznamu.

TC5: Aktualizace uživatele (testUpdateUser)

Cíl: Ověřit, že služba umožňuje správnou aktualizaci údajů uživatele.

Předpoklady:

- Uživatel s e-mailem testuser5@example.com existuje a má heslo testpassword.

Testovací data:

- E-mail: testuser5@example.com
- Heslo: testpassword
- Nové jméno: Jane Updated

Kroky:

1. Přihlaste se pomocí e-mailu testuser5@example.com a hesla testpassword.
2. Aktualizujte jméno uživatele na "Jane Updated".
3. Získejte aktualizované údaje o uživateli.

Očekávaný výsledek:

- Aktualizace údajů uživatele proběhne úspěšně.
- Aktualizované údaje odpovídají zadaným hodnotám.

Aktuální výsledek:

- Aktualizace údajů uživatele proběhne úspěšně.
- Aktualizované údaje odpovídají zadaným hodnotám.

TC6: Aktualizace účtů uživatele z bankovního API (testUpdateUserAccountsFromBankApi)

Cíl: Ověřit, že služba umožňuje správnou aktualizaci účtů uživatele z bankovního API.

Předpoklady:

- Uživatel s e-mailem testuser6@example.com existuje a má heslo testpassword.

Testovací data:

- E-mail: testuser6@example.com
- Heslo: testpassword

Kroky:

1. Přihlaste se pomocí e-mailu testuser6@example.com a hesla testpassword.
2. Aktualizujte účty uživatele z bankovního API.
3. Získejte aktualizované údaje o uživateli.

Očekávaný výsledek:

- Aktualizace účtů uživatele z bankovního API proběhne úspěšně.
- Seznam aktualizovaných účtů obsahuje finanční produkty.

Aktuální výsledek:

- Aktualizace účtů uživatele z bankovního API proběhne úspěšně.
- Seznam aktualizovaných účtů obsahuje finanční produkty.

6.2.2 Integrační testy KBService

TC1: Získání účtů uživatele (testGetUserAccounts)

Očekávaný výsledek:

- Metoda **getUserAccounts** získá seznam účtů uživatele pomocí klienta **kbClient**.
- Seznam získaných účtů odpovídá očekávaným účtům z testovacích dat.

Aktuální výsledek:

- Metoda **getUserAccounts** získá seznam účtů uživatele pomocí klienta **kbClient**.
- Seznam získaných účtů odpovídá očekávaným účtům z testovacích dat.

TC2: Získání účtu (testGetAccount)

Očekávaný výsledek:

- Metoda **getAccount** získá účet uživatele pomocí klienta **kbClient**.
- Získaný účet odpovídá očekávanému účtu z testovacích dat.

Aktuální výsledek:

- Metoda **getAccount** získá účet uživatele pomocí klienta **kbClient**.
- Získaný účet odpovídá očekávanému účtu z testovacích dat.

V obou testech jsou očekávané výsledky shodné s aktuálními výsledky. To naznačuje, že implementace **KBServiceImpl** správně komunikuje s klientem **kbClient** a vrací očekávané výsledky pro získání účtů uživatele a jednotlivých účtů.

6.3 Budoucí práce

Ačkoli v implementaci aplikace bylo dosaženo všech cílů, je zde několik oblastí, které by mohli být v budoucnu dále rozvíjeny.

6.3.1 Rozšíření podporovaných bankovních klientů

V současné době aplikace podporuje integraci účtů z Komerční banky, ČSOB, George a Coinbase. Budoucí práce by mohla zahrnovat rozšíření podpory pro další bankovní instituce jako například AirBank, Raiffeisenbank a ostatních finanční služeb, čímž by se zvýšila uživatelská základna a poskytla větší flexibilita pro správu financí.

6.3.2 Implementace reálných bankovních APIs

Dosud byla aplikace testována pomocí simulovaných bankovních klientů. Budoucí práce by měla zahrnovat integraci reálných bankovních APIs, což by umožnilo aplikaci přistupovat k aktuálním datům z bankovních systémů. Implementace reálných bankovních APIs by zlepšila přesnost a aktuálnost informací poskytovaných uživatelům, a zároveň by zvýšila důvěryhodnost a užitečnost aplikace.

6.3.3 Implementace převodu peněz

Dalším krokem v rozvoji aplikace by mohla být implementace funkcionality pro převod peněz mezi bankovními účty. Tato funkce by uživatelům umožnila provádět finanční transakce přímo z aplikace, čímž by se zvýšila její praktická využitelnost. Implementace převodu peněz by zahrnovala bezpečnostní opatření, jako jsou více faktorová autentizace a šifrování, aby byla zajištěna bezpečnost uživatelských dat a finančních transakcí.

6.3.4 Uživatelská zkušenost a rozhraní

Zlepšování uživatelské zkušenosti (UX) a uživatelského rozhraní (UI) je nikdy nekončící proces. Na základě zpětné vazby od uživatelů by mohla být aplikace vylepšována tak, aby byla intuitivnější a snadněji použitelná. To zahrnuje nejen vizuální design, ale také optimalizaci výkonu a plynulost interakcí.

6.4 Shrnutí

V této kapitole jsme se zaměřili na testování různých částí aplikace, abychom zajistili její správnou funkčnost a minimalizovali výskyt chyb. Hlavním cílem bylo ověřit, že servisní vrstva aplikace funguje správně a je správně propojena. K dosažení tohoto cíle byly použity různé typy testů, včetně integračních, smoke testů, a alpha a beta testů.

Kapitola 7

Závěr

Tato bakalářská práce se zabývala nejen analýzou dostupných řešení v sektoru online bankovnictví a dostupných technologií pro vývoj aplikací na platformě android, ale také samotným vývojem mobilní aplikace pro sledování jak bankovních, tak i investičních účtů a finančních operací. Na základě proběhlé analýzy byly definovány klíčové funkční a nefunkční požadavky, které pak napomohly výběru správným technologiím a následně návrhu architekturu prototypu aplikace. Výsledkem této práce je prototyp aplikace pro správu a sledování bankovních či investičních účtů. Tento prototyp byl následně řádně otestován. Toto testování zahrnovalo jak alfa, tak beta testy. Výsledek uživatelského testování ukázal že aplikace je funkční a splňuje stanovené cíle.

Literatura

[1] KB – Naše aplikace. Online. KB. 2024. Dostupné z: https://www.kb.cz/cs/nase-aplikace. [cit. 2024-05-07].

[2] Banking has a name. George. Online. 2023. Dostupné z: https://george-labs.com/. [cit. 2024-12-31].

[3] Internetové a mobilní bankovnictví – ČSOB. Online. Csob.cz. 2024. Dostupné z: https://www.csob.cz/lide/ucty/internetove-a-mobilni-bankovnictvi/smart. [cit. 2024-05-07].

[4] Coinbase: Buy Bitcoin & Ether. Online. Google Play. 2024. Dostupné z: https://play.google.com/store/apps/details?id=com.coinbase.android. [cit. 2024-05-07].

[5] Sběr a modelování požadavků. Online. Moodle. 2024. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=8557. [cit. 2024-05-15].

[6] Kotlin. Online. Kotlin. 2024. Dostupné z: https://kotlinlang.org/. [cit. 2024-05-12].

[7] THE Java™ Programming Language, Fourth Edition. Online. August 17, 2005. Addison Wesley Professional, 2005. ISBN 0-321-34980-6. Dostupné z: https://www.acs.ase.ro/Media/Default/documents/java/ClaudiuVinte/books/ArnoldGoslingHolmes06.pdf. [cit. 2024-05-12].

[8] Dart. Online. Dart. 2024. Dostupné z: https://dart.dev/. [cit. 2024-05-12].

[9] Android Studio. Online. Developers. 2024. Dostupné z: https://developer.android.com/studio. [cit. 2024-05-13].

[10] Tutorial: Create your first Android application – IntelliJ IDEA. Online. JetBrains. 2024. Dostupné z: https://www.jetbrains.com/help/idea/create-your-first-android-application.html#0. [cit. 2024-05-13].

[11] Developer Tools & IDE – Eclipse. Online. Eclipse Foundation. 2024. Dostupné z: https://www.eclipse.org/topics/ide/. [cit. 2024-05-13].

[12] Android Development Tools for Eclipse. Online. Eclipse Marketplace. 2024. Dostupné z: https://marketplace.eclipse.org/content/android-development-tools-eclipse. [cit. 2024-05-13].

[13] MICROSOFT. Visual Studio Tools for Xamarin. Online. Microsoft. 2024. Dostupné z: https://visualstudio.microsoft.com/cs/xamarin/. [cit. 2024-05-13].

[14] SQLite. Online. SQLite. 2024. Dostupné z: https://www.sqlite.org/. [cit. 2024-05-13].

[15] Room – Jetpack. Online. Developers. 2024. Dostupné z: https://developer.android.com/jetpack/androidx/releases/room. [cit. 2024-05-13].

[16] Cloud Firestore. Online. Firebase. 2024. Dostupné z: https://firebase.google.com/docs/firestore. [cit. 2024-05-13].

[17] Anderson, C. (2012). The Model-View-ViewModel (MVVM) Design Pattern. In: Pro Business Applications with Silverlight 5. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-3501-9_13

[18] Gitlab. Online. Gitlab. 2024. Dostupné z: https://about.gitlab.com/. [cit. 2024-05-13].

[19] Design for mobile. Online. Developers. 2024. Dostupné z: https://developer.android.com/design/ui/mobile. [cit. 2024-05-13].

[20] NIELSEN, Jakob. Ten Usability Heuristics. 2009. Dostupné také z: https://pdfs.semanticscholar.org/5f03/b251093aee730ab9772db2e1a8a7eb8522cb.pdf.

[21] HOW MANY PEOPLE USE DARK MODE IN 2024? (USAGE STATISTICS). Online. Earthweb. 2023. Dostupné z: https://earthweb.com/how-many-people-use-dark-mode/. [cit. 2024-05-13].

[22] Teorie míchání barev. Online. Draw planet. 2020. Dostupné z: https://www.drawplanet.cz/teorie-michani-barev/. [cit. 2024-05-23].

[23] Adapter. Online. Refactoring guru. 2014. Dostupné z: https://refactoring.guru/design-patterns/adapter. [cit. 2024-05-24].

[24] Bridge. Online. Refactoring guru. 2014. Dostupné z: https://refactoring.guru/design-patterns/bridge. [cit. 2024-05-19].

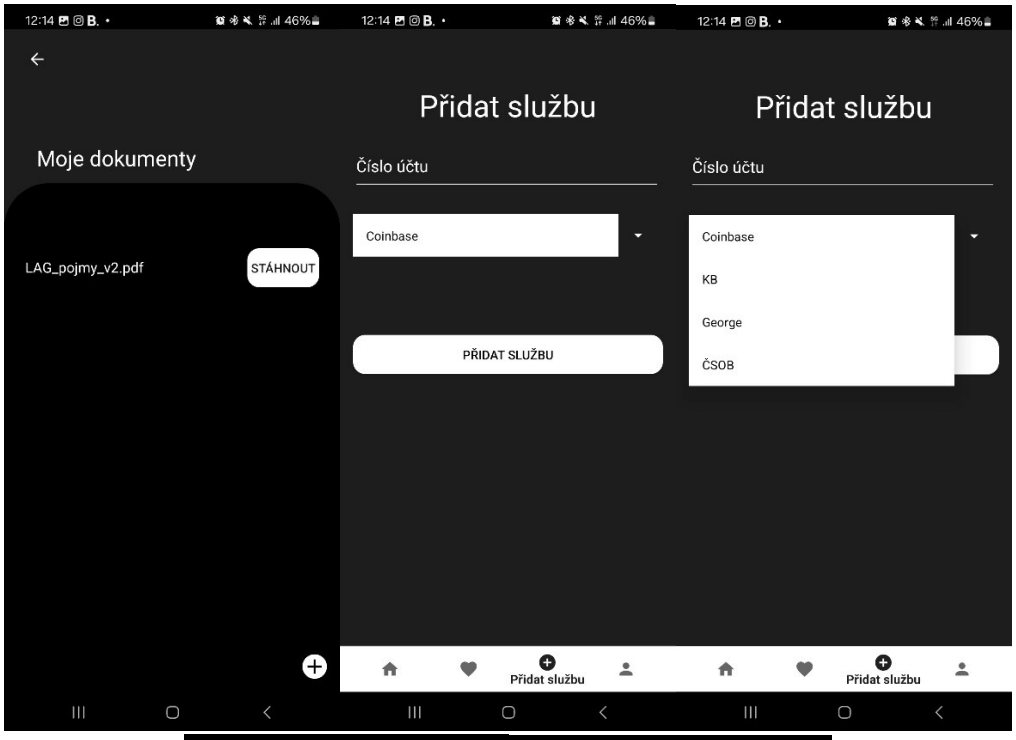
[25] Decorator. Online. Refactoring guru. 2014. Dostupné z: https://refactoring.guru/design-patterns/decorator. [cit. 2024-05-24].

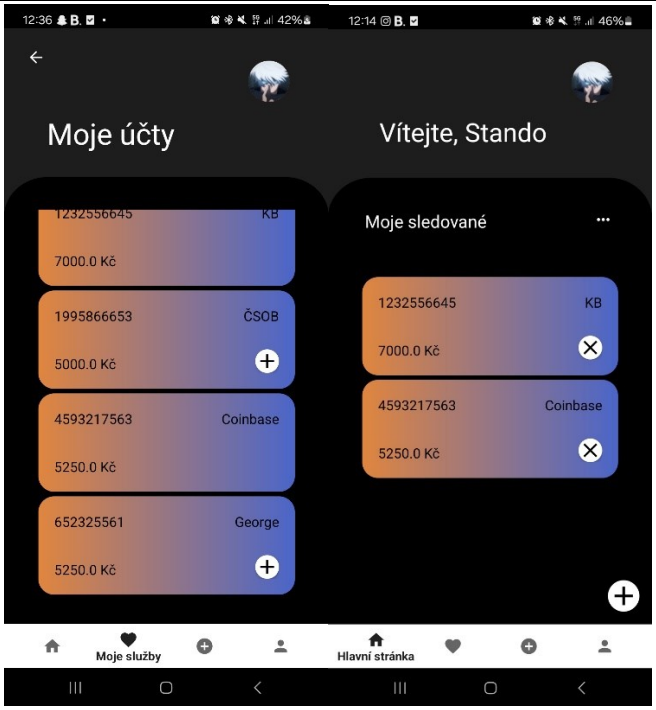
[26] Jackson Support for Kotlin. Online. Baeldung. 2024. Dostupné z: https://www.baeldung.com/kotlin/jackson-kotlin. [cit. 2024-05-23].

[27] Druhy testů (Typy testování software). Online. KITNER. 2024. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/. [cit. 2024-05-16].

[28] Alfa testování – co to je, typy, proces, vs. beta testy, nástroje a další! Online. ZAPTEST. 2024. Dostupné z: https://www.zaptest.com/cs/alfa-testovani-co-to-je-typy-proces-vs-beta-testy-nastroje-a-dalsi. [cit. 2024-05-16].

Příloha A
Další ukázky aplikace





Obrázkek-Příloha A.1:Další ukázka aplikace

Příloha B

Důležité odkazy

- Odkaz na Gitlab repositář s implementovaným prototypem android aplikace: https://gitlab.fel.cvut.cz/bendasta/bankingapp