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

MOBILNÍ APLIKACE PRO PÁROVÁNÍ FILMŮ KE SHLÉDNUTÍ

MOBILE APP FOR PAIRING MOVIES TO WATCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ ZOUHAR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL KAPINUS**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146137

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Zouhar Tomáš**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Mobilní aplikace pro párování filmů ke shlédnutí**
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 skupinové vyhledávání filmů z dostupných VoD služeb ke shlédnutí na základě vzájemného souhlasu (všichni uživatelé ve skupině souhlasí že daný film chtějí vidět). Inspirujte se např. aplikací Tinder.
3. Aplikaci navrhněte tak, aby bylo možné vytvořit několik skupin, ve kterých probíhá výběr filmů nezávisle (jedna skupina s partnerem, druhá s kamarády a podobně). Navrhněte jednoduchý způsob jak do skupiny přidávat ostatní uživatele aplikace.
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

Tato práce se zabývá procesem návrhu a vývoje moderních mobilních aplikací pro platformu Android. Jejím výsledkem je mobilní aplikace pro párování filmů pro skupiny uživatelů. Uživatelé aplikace budou moci po registraci zakládat skupiny, či se do nich přidávat. V rámci skupiny pak mohou uživatelé hodnotit nabízené filmy. Výsledkem tohoto procesu je film, který získal nejvíce kladných ohodnocení, popřípadě film, který kladně ohodnotili všichni uživatelé.

Abstract

This thesis deals with designing and developing modern mobile applications for the Android platform. The result of this thesis is a mobile application for matching movies to watch in a group. Users of this app can create groups or join other users' groups. Users can rate movies based on their common interests. The final product of this process is the movie that has received the best ratings or the movie which all users agreed on.

Klíčová slova

Android, Firebase, mobilní aplikace, film, kino, skupina, přátelé, databáze, uživatelské rozhraní, Figma, návrh

Keywords

Android, Firebase, mobile application, movie, cinema, group, friends, database, user interface, Figma, mockup

Citace

ZOUHAR, Tomáš. *Mobilní aplikace pro párování filmů ke shlédnutí.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

Mobilní aplikace pro párování filmů ke shlédnutí

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.

.....
Tomáš Zouhar
8. května 2023

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Michalu Kapinusovi za časté konzultace, cenné poznámky a rady po celou dobu tvorby práce. Zároveň bych chtěl poděkovat rodině a přátelům za konstantní podporu jak při studiu, tak při tvorbě závěrečné práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato práce má za cíl zaměřit se na problematiku vývoje moderních mobilních aplikací a využití současných moderních technologií pro jejich vývoj. Výsledná aplikace má za účel umožnit skupině uživatelů dohodnout se na jednom snímku, který chtějí společně zhlédnout. Po procesu vybírání společného filmu je skupině představen ideální snímek, který by měl nejlépe reprezentovat potřeby a nároky jejich uživatelů.

Motivace za návrhem aplikace vychází ze situací reálného světa a reálných potřeb jedinců, které byly vypozorovány. Souvisí zároveň se současným rozšířením filmů a seriálů do digitálních médií a jejich přístupnosti on-line v podobě vysílacích (angl. Video on Demand) platforem. Tento fakt má za výsledek, že si lidé mohou v televizi nebo v prohlížeči pustit takřka libovolný film či seriál a také takto trávit čas i s přáteli (ať už fyzicky pospolu či on-line).

Když se lidé chtějí dívat na filmy či seriály s přáteli společně, mají nyní k dispozici nespočet různých snímků z různých platforem pro jejich sledování a může být zdlouhavé najít ten pravý film, který by se zavděčil všem. Tento problém by měla výsledná aplikace řešit.

Práce je řazena do kapitol, kdy každá kapitola popisuje danou část vývoje aplikace (jeden logický celek při vývoji moderních mobilních aplikací). Kapitoly jdou tématicky po sobě, tím reprezentují ideální chronologický postup při tvoření takovýchto moderních mobilních aplikací, jejich uživatelských rozhraní a testování.

Kapitola 2

Teorie

Před samotnou implementací aplikace je třeba provést průzkum existujících zdrojů a technologií, které lze využívat pro co největší uživatelskou přívětivost aplikace. Tyto zdroje a technologie mohou vývojáři aplikace značně zjednodušit proces tvorby aplikací.

2.1 Platforma

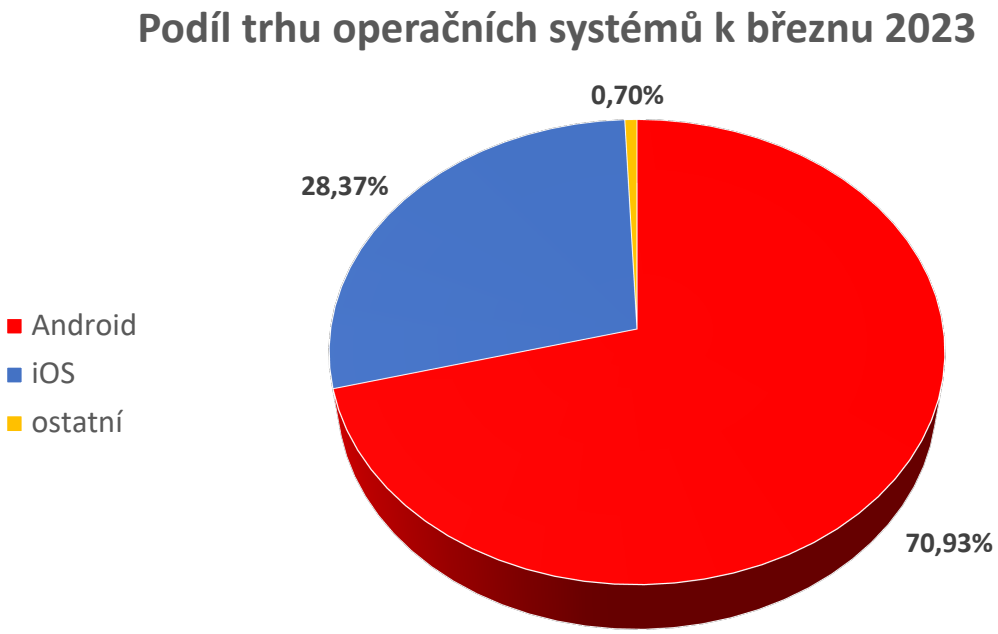
Termínem platforma rozumíme systém, na kterém bude aplikace postavena. Ve světě mobilních aplikací momentálně dominují dvě hlavní platformy – Android od firmy Google a iOS od firmy Apple.

Každá z těchto platforem má své výhody a nevýhody. Programování nativních aplikací (= aplikací určených pouze na jednu z platforem) pro tyto systémy je rozdílné, vyžaduje jiné prostředí, programovací jazyk i přístup. V případě programování multiplatformních aplikací (angl. cross-platfrom = aplikací, které jsou programovány v jednom programovacím jazyce pro více platforem najednou) můžeme aplikaci napsanou v jednom jazyce distribuovat na více platforem naráz, většinou však na úkor výkonu aplikace.

Android

Počátky platformy Android datujeme do roku 2003, kdy byla založena společnost Android Inc. Tuto firmu v roce 2005 zakoupil Google a v roce 2007 se tato platforma stala standardem pro značnou řadu mobilních výrobců (HTC, Sony, Samsung, Google, apod.) s prvním androidovým telefonem uvedeným na trh v roce 2008 [8].

Android je operační systém postavený na Linuxovém jádru. Jedná se o otevřený software (angl. open-source), což znamená, že každý programátor může nahlédnout do zdrojového kódu a lépe tak pochopit chování této platformy, což může zapříčinit lepší optimalizaci aplikací a pochopení celého ekosystému zařízení.



Obrázek 2.1: Podíl trhu operačních systémů k březnu roku 2023 [10]

Systém Android se v poslední době stává velkým ekosystémem, který nezahrnuje jen mobilní telefony, ale i chytrá zařízení, chytré domácnosti, příslušenství či auta. Všechna tato zařízení patří do skupiny IoT (internet věcí).

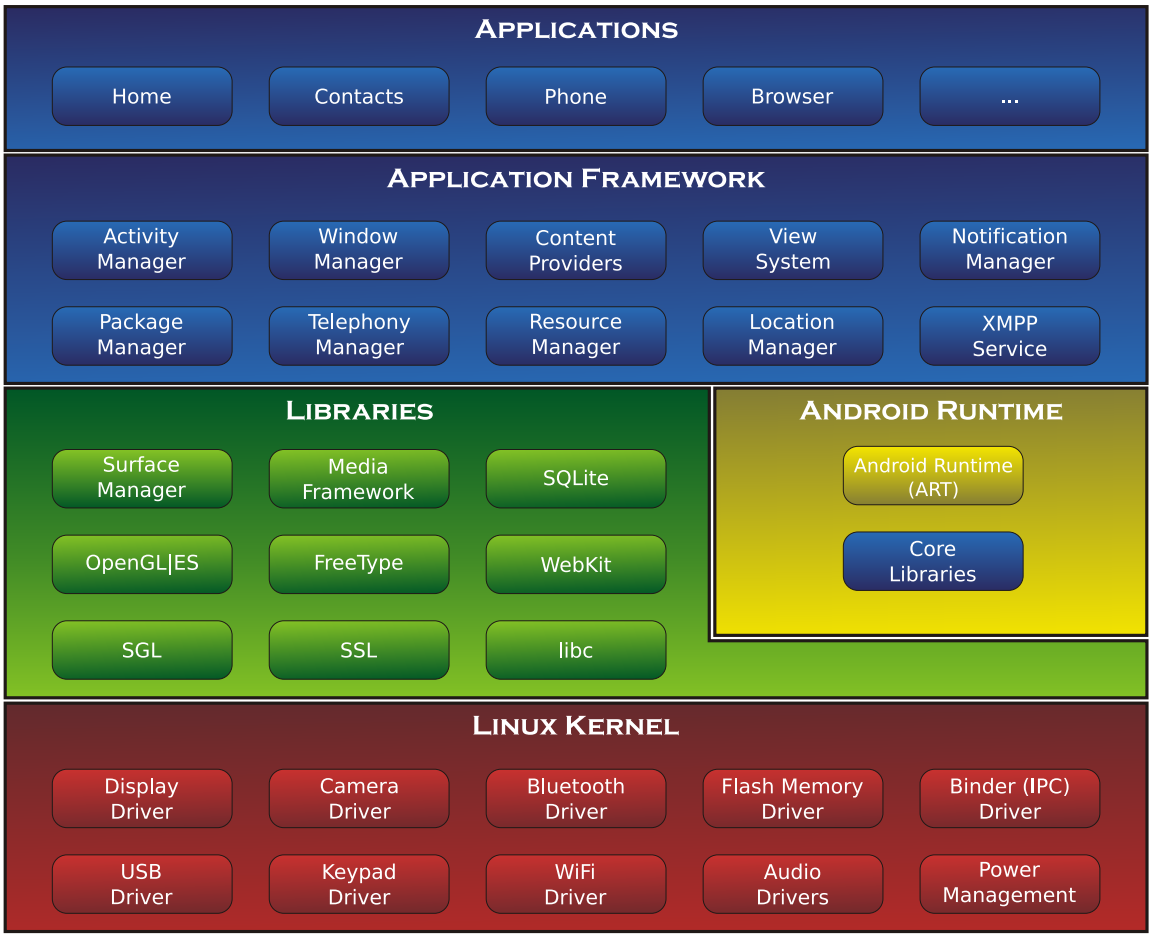
To vše zapříčinilo, že Android je jako operační systém preferován značnou částí mobilních zařízení k roku 2023, jak lze vidět na obrázku 2.1.

Architektura Androidu

Architekturu zařízení Android velice dobře vystihuje schéma na obrázku 2.2, kde je architektura popsána od nejnižší architektonické vrstvy (dole) po nejvyšší (nahoře), více o samotné architektuře a detailní popisy jednotlivých vrstev lze čerpat z knihy Vývoj aplikací pro Android od Ľuboslava Lacka [7].

Za zmínku stojí aplikační vrstva tohoto schématu, která mimo ostatní obsahuje i tzv. *ActivityManager*, který spravuje životní cykly aplikací a přepíná jejich kontexty. Tato vrstva je navrhnuta tak, aby její komponenty byly jednoduše použitelné uživatelem [11].

Pro programování systému Android je zapotřebí stáhnout Android SDK (soubor vývojářských nástrojů) a Android API [8]. Hlavní 2 jazyky, ve kterých lze nativně programovat pro Android platformu jsou Java a Kotlin, oba jazyky jsou popsány v kapitole 2.2.



Obrázek 2.2: Architektura struktury Android zařízení popisující jednotlivé vrstvy

Figma

Figma je jednoduchým nástrojem pro tvorbu maket návrhů aplikací (termínu maketa aplikace rozumíme vyšší úroveň návrhu, která zahrnuje i estetickou stránku aplikace, popřípadě i UX (uživatelská zkušenost), viz. kapitola 2.5). Přestože je nástroj Figma velmi jednoduchý a intuitivní na použití, nabízí uživatelům pokročilé funkce, jako jsou například možnost tvorby demonstrační aplikace pomocí přechodů mezi částmi návrhu a dynamickými změnami obsahu na základě akcí uživatele. Právě takové demonstrování celé aplikace se vyplatí při testování UX, protože lze pozorovat uživatele, jak s aplikací komunikuje v reálném čase.

2.2 Programovací jazyk

Programovací jazyk je komunikačním nástrojem mezi programátorem a počítačem (v našem případě mobilním telefonem). Programovací jazyk slouží jako mezi-jazyk, kterému obě strany dobře rozumí (dobrá střední cesta). Programátor si může programovací jazyk interpretovat do svého rodného jazyka, například při rekonstrukci kódu, ladění a hledání chyb. Na druhou stranu mobilní telefon (jeho systém) si tento kód převede na instrukce, které poté na úrovni hardware vykonává.

Ve světe mobilních aplikací se můžeme setkat s programovacími jazyky, které jsou jak pro nativní programování (Java, Kotlin, Swift), tak pro multiplatformní (aplikaci lze spustit na všech podporovaných platformách) programování (Flutter, React Native).

- Programovací jazyky pro nativní programování mají většinou velikou výhodu v tom, že jsou optimalizovány přímo na daný operační systém. Mohou tak využít jeho zdroje co nejefektivněji a nejlépe. Nevýhoda však je nemožnost přenést aplikaci na jiný operační systém, což je ve světě mobilních aplikací problém z důvodu, že neexistuje systém, který by na trhu měl monopol, tím pádem se aplikace musí programovat „nadvakrát“.
- Programovací jazyky pro multiplatformní vývoj řeší tento problém s kompatibilitou, avšak narozdíl od nativních jazyků nedokáží využít plný potenciál jádra a jeho zdrojů a tím pádem jsou značně pomalejší.

Java

Základy programovacího jazyka Java se začaly budovat v 90. letech 20. století. V této době vznikalo stále více platforem pro běh programů. Každá z těchto platforem vyžadovala kód napsaný v jazyce, který byl pro tuto platformu kompatibilní. Programátoři měli v této době na výběr buďto psát programy vícekrát v jazycích, které byly nativní pro dané platformy (více popsáno v kapitole 2.2), nebo použít více generický jazyk (například jazyk C++) a používat různé programy pro překlad kódu na moduly či spustitelné soubory spustitelné na dílčích platformách [9].

Hlavní premise jazyka Java byla možnost spustit stejný kód na všech platformách (Linux, Windows, později kupříkladu macOS). Tato funkcionalita byla docílena navržením tzv. „Java Virtual Machine“, což funguje jako virtuální prostředí. Java kompilátor tedy zkompiluje program a poskytne pseudokód, který je dále spouštěn právě v JVM. Tímto je zajištěna kompatibilita mezi zařízeními a téměř dokonalá přenositelnost kódu, kdy jej může spustit jakákoliv platforma, dokud má JRE (angl. Java Runtime Enviroment), které obsahuje mimo JVM i další prostředky pro spouštění Java aplikací.

Kotlin

Počátky jazyka Kotlin datujeme k roku 2016, jedná se tedy o relativně mladý jazyk. U zrodu tohoto programovacího jazyka stála firma JetBrains, která spustila „Projekt Kotlin“ v roce 2011. Jedná se o nový jazyk, který cílí na JRE, navazuje tedy na jazyk Java.

V posledních letech je Kotlin stále více upřednostňován před ostatními jazyky. Velký zlomový bod pro tento jazyk nastal v roce 2019, kdy Google prosadil Kotlin jako upřednostňovaný programovací jazyk platformy Android. Podpora pro Kotlin je tedy navzdory jeho relativní novotě ohromná. Z důvodu, že Kotlin cílí na JRE prostředí znamená, že je kompatibilní s Java kódem a tyto jazyky tak mohou fungovat pospolu.

Kotlin zároveň nabízí mnoho způsobů, jakými psát kód. Podporuje procedurální programování, deklarativní programování, objektově orientované programování nebo funkcionální programování [6].

Flutter

Soubor vývojářských nástrojů (SDK) Flutter se řadí mezi multiplatformní nástroje, jenž byl vyvinut firmou Google v roce 2017. Dovoluje uživateli vyvíjet aplikace pro různé platformy od Androidu, iOS, webových aplikací až po vestavěné systémy.

Flutter využívá jazyk Dart, který byl vyvinut taktéž firmou Google, avšak již v roce 2011. Jazyk Dart vychází z programovacích jazyků typu JavaScript, ale i objektově orientovaných jazyků typu Java a C#. Flutter lze využít pro klientské i serverové aplikace s podporou například REST API [1].

2.3 Vývojová prostředí

S řadou programovacích jazyků pro tvorbu mobilních aplikací přichází také řada různých prostředí pro vývoj těchto typů aplikací.

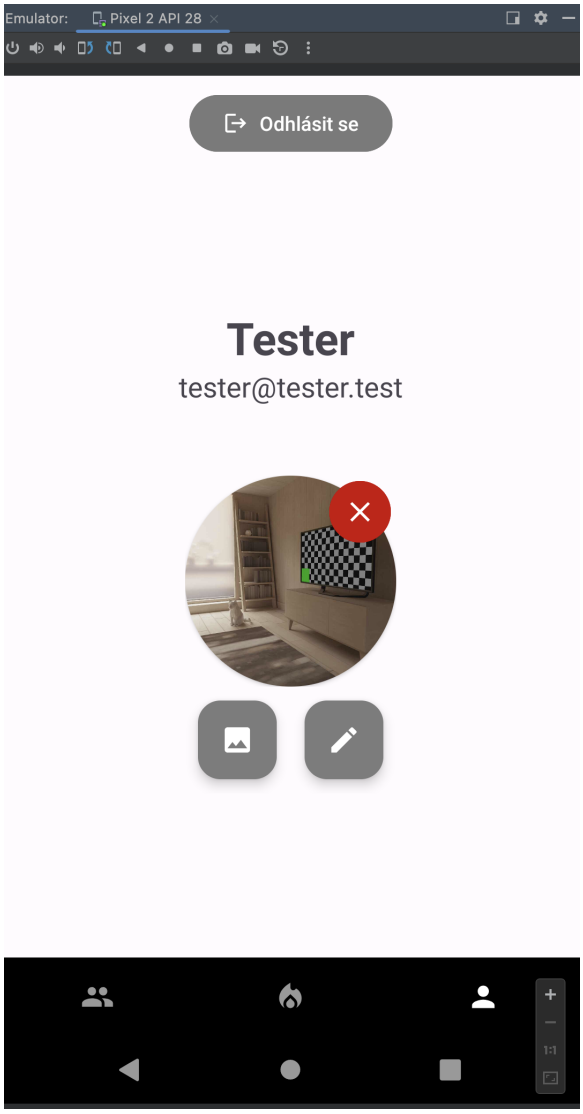
IntelliJ IDEA

IntelliJ IDEA je vývojové prostředí vyvinuto firmou JetBrains, jedná se o IDE (integrované vývojové prostředí), které poskytuje programátorovi řadu nástrojů pro pohodlnější psaní programů. Cílí na jazyky Java a Kotlin (viz. kapitola 2.2). Uživatel si může prostředí a jeho funkce upravovat řadou zásuvných modulů (angl. pluginů) a dodatků, které jsou k dispozici. Jedná se o komerční software.

Android Studio

Android Studio je prostředí přímo určené pro vývoj mobilních aplikací. Vzešlo z prostředí IntelliJ IDEA a bylo představeno firmou Google v roce 2013. Poskytuje přímý přístup ke všem potřebným nástrojům pro mobilní vývoj, které uživateli usnadní celý proces programování. Android Studio nabízí také přímou integraci se systémem Firebase (kapitola 2.4) přímo v aplikaci.

Android Studio také podporuje přímo v prostředí emulovat mobilní telefon pro testování aplikace v reálném čase. Emulátor poskytuje možnost rychlého testování různých variací zařízení (kupříkladu varianty tablet, telefon či různá ostatní chytrá zařízení). Příklad běhu tohoto emulátoru lze vidět na obrázku 2.3.



Obrázek 2.3: Aplikace spuštěná v emulátoru prostředí Android Studio

2.4 Databáze

Při používání aplikace bude v jistém bodě nutno sestavit a následně i komunikovat s databází. Dvě dominantní technologie, které jsou na trhu pro řízení báze dat jsou:

- Relační databáze SQL vychází z dotazovacího jazyka Structured Query Language, který slouží pro operace s daty uloženými v těchto databázích. Data jsou zde uloženy v tabulkách, které obsahují řádky a sloupce.
- NoSQL databáze jsou novější implementací báze dat. Data zde nejsou uložena v tabulce, ale například ve stromové struktuře, či grafu.

Backend jako služba (BaaS)

Termínu „backend jako služba“ (BaaS) rozumíme konceptu, při kterém třetí strana poskytuje plně funkční backendové služby (služby operující s daty) pro aplikace a umožňuje vývojářům snadno přistupovat k databázím, ukládání souborů, autentizaci uživatelů a dalším funkcím. Tento přístup ulehčuje vývojářům práci a umožňuje jim rychleji vyvíjet aplikace bez nutnosti plánování a vytváření vlastního backendu [2].

Firebase od firmy Google poskytuje přesně tyto služby. Není tedy třeba hostovat vlastní backend, čímž se urychlí vývoj aplikace a vývojář se může takřka plně zabývat uživatelským rozhraním aplikace.

Služba Firebase disponuje převážně těmito hlavními produkty[1]:

- Realtime Database
- Cloud Firestore
- Hosting
- Testování
- Autentizace a autorizace uživatelů

Finanční politika Firebase

Služba Firebase staví svoji finanční politiku k roku 2023 na dvou plánech:

- **Spark** plán je bezplatný a poskytuje vývojáři základní funkcionality, které však ve většině případů dostačují potřebám vývoje jednoduché mobilní aplikace. Jedná se o plán, který má nastavené pevné kvóty, které musí být dodrženy.
- **Pay as you grow** (volně přeloženo – plať zároveň jak rosteš) plán funguje na základě toho, jak již název napovídá, že vývojáři je účtován měsíční poplatek za využívání služeb, který je přímo úměrný využití služeb. Tento plán je výhodný pro velké firmy, kterým základní kvóty nestačí. Zároveň je zde zohledňováno využití individuálních služeb, tudíž vývojáři nejsou účtovány služby, které nejsou využívány.

Spark plán pro Firebase Realtime Database	
Úložiště	1 GB
Stažená data	10 GB/měsíc
Aktiniích připojení najednou	100

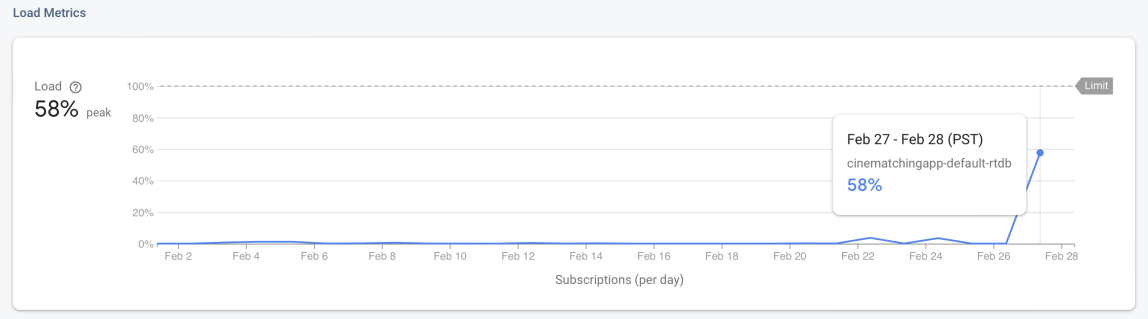
Tabulka 2.1: Tabulka limitů pro Spark plán služby Firebase Realtime Database

Z tabulky 2.1 lze vyčíst, že úložiště i stažená data by pro menší aplikace problém být neměl. Při škálování aplikace by mohl nastat problém v aktuálně připojených zařízení. Pokud by vývojáři tento limit nestačil, v plánu „Pay as you grow“ je možnost obsluhovat až 200 000 zařízení najednou[2].

[1] Seznam produktů získán ze stránek služby Firebase (`https://firebase.google.com/products-build`), aktuální k datu 13. 4. 2023

[2] Data jsou aktuální k datu 2. dubna 2023 a jsou získána z tabulky poskytované na odkaze `https://firebase.google.com/pricing`

Služba Firebase poskytuje také nástroje pro lepší přehled o komunikaci s databází a počtu dat, která byla odeslána, velikost uložených dat, aktivně připojená zařízení či zátěž databáze, jak je možno vidět na obrázku 2.4.



Obrázek 2.4: Graf zátěže Firebase Realtime Databáze udávaný v procentech za měsíc únor

2.5 Grafické uživatelské rozhraní

Grafickým uživatelským rozhraním (GUI) rozumíme soubor nástrojů, kterými uživatel komunikuje s aplikací. Může se jednat o moduly, které uživateli zobrazují informace – kupříkladu text, obrázek, ale i o moduly, které uživatel používá pro komunikaci s aplikací samotnou – tlačítko či textové pole. Tyto nástroje nemají však pouze funkci prezentační, ale i estetickou. Mají tu možnost, že mohou na uživatele zapůsobit a vyvolat v něm dojem o aplikaci jako celku [3].

Při tvorbě GUI je nezbytné myslet na uživatele a působit na něj příjemným dojmem. Pomocí GUI můžeme v uživateli aplikace také vzbudit emoce či pocity. Při samotném tvoření aplikace chceme vždy na uživatele působit kladným dojmem, který zajistí, že se uživatel k aplikaci bude rád vracet a trávit v ní více času.

Při tvorbě GUI je více než vhodné vytvořit si kostru aplikace – wireframe, který je poté testován na uživatelích. Výhodou tvorby takového návrhu je podstatně nižší režie, než při testování hotové aplikace, jednodušší implementace případných doplňků a úprava stávajících vlastností aplikace. Tato maketa by měla co nejkonkrétněji demonstrovat požadované testované vlastnosti aplikace, avšak se zachováním abstrakce, abychom uživatele neovlivnili jak kladně, tak i záporně. Takové ovlivnění může způsobit neobjektivnost návrhu a zkreslení výsledku testu [3].

User experience

Pojmem user experience (uživatelská zkušenost) rozumíme aspekty aplikace, které se pojí s uživatelským pocitem z aplikace co se týče jednoduchosti na použití. UX bez pochyb úzce souvisí s GUI a jistým způsobem z něj i vychází. Při návrhu UX chceme co nejvíce snížit uživatelskou režii pro vykonávání akcí v aplikaci. Snažíme se mít navigaci i jednotlivé komponenty co nejpřehlednější, aby uživatel věděl, kde se právě nachází, přestože se do aplikace dostal poprvé. Také je dobré ctít moderní trendy, aby se uživatel v aplikaci cítil „jako doma“. V případě, že uživatel vyhodnotí aplikaci jako složitou, zanechá to v něm záporný dojem a může dojít i k takovému závěru, že aplikaci využívat nebude vůbec. Z těchto důvodů nesmí být testování UX opomenuto [5].

Kapitola 3

Návrh řešení

Před samotnou implementací aplikace je takřka nezbytné provést návrh, jak by aplikace měla vypadat a co by měla umět. Pro ideální funkcionalitu aplikace je dobré provést průzkum trhu a ideálních uživatelů, aby výsledná aplikace co nejvíce reflektovala jejich potřeby.

3.1 Existující řešení

Existujících aplikací na toto téma je řada. Je však nutno rozdělit aplikace do dvou skupin, a to aplikace, ze kterých vznikla idea takového párování objektů (Badoo, Tinder) a aplikace, které již nabízí obdobnou funkcionalitu, kdy poskytují uživateli na jisté míry možnost výběru filmu na základě společných žánrů, ať už ve dvojici či ve skupině.

- Aplikace Badoo, Tinder – z této skupiny aplikací jsem se inspiroval nápadem pro vyhledání ideálního prvku z pole zúženého obdobnými vlastnostmi.
- Aplikace pro pomoc s výběrem filmů – u aplikací z této skupiny je třeba dále rozebrat uživatelskou přívětivost, funkčnost a výsledný význam aplikace pro uživatele.

Badoo, Tinder

V první kategorii obdobných aplikací jsou aplikace, ze kterých je přejímán jejich koncept s tím, že smysl výsledných aplikací není totožný. Jejich konceptem rozumíme způsob vyhledávání ideálního – v případě těchto aplikací – partnera. Obě aplikace pracují se zájmy, podobnými vlastnostmi, věkem a mnoho dalšími faktory jedinců a na základě toho doporučí uživateli ideálního partnera. Avšak druhá polovička musí tento postoj potvrdit tím, že se jejich zájmy shodnou. Jedná se o obdobný přístup jako v mé aplikaci s tím, že v mé aplikaci se na filmu musí shodnout celá skupina, tj. většinou více stran než jen dvě.

Funkcionalita

Obě tyto aplikace fungují na základě nutnosti vzájemného kladného hodnocení dvou protějšků, aby spolu poté mohli komunikovat. Uživatel postupně prohledává databázi ostatních uživatelů a hodnotí je buď kladně, nebo záporně. K rozhodování má uživatel k dispozici: přibližnou lokaci, fotky, popis, zájmy. Při prvním vstupu do aplikace se uživatel zaregistruje, poté může aplikaci užívat.

Při kladném ohodnocení z obou stran uživatelů nastává tzv. „match“ – shoda, kdy aplikace umožní uživatelům spolu komunikovat prostřednictvím chatu jako tomu je u obdobných komunikačních aplikací.

Aplikace s obdobnou funkcionalitou

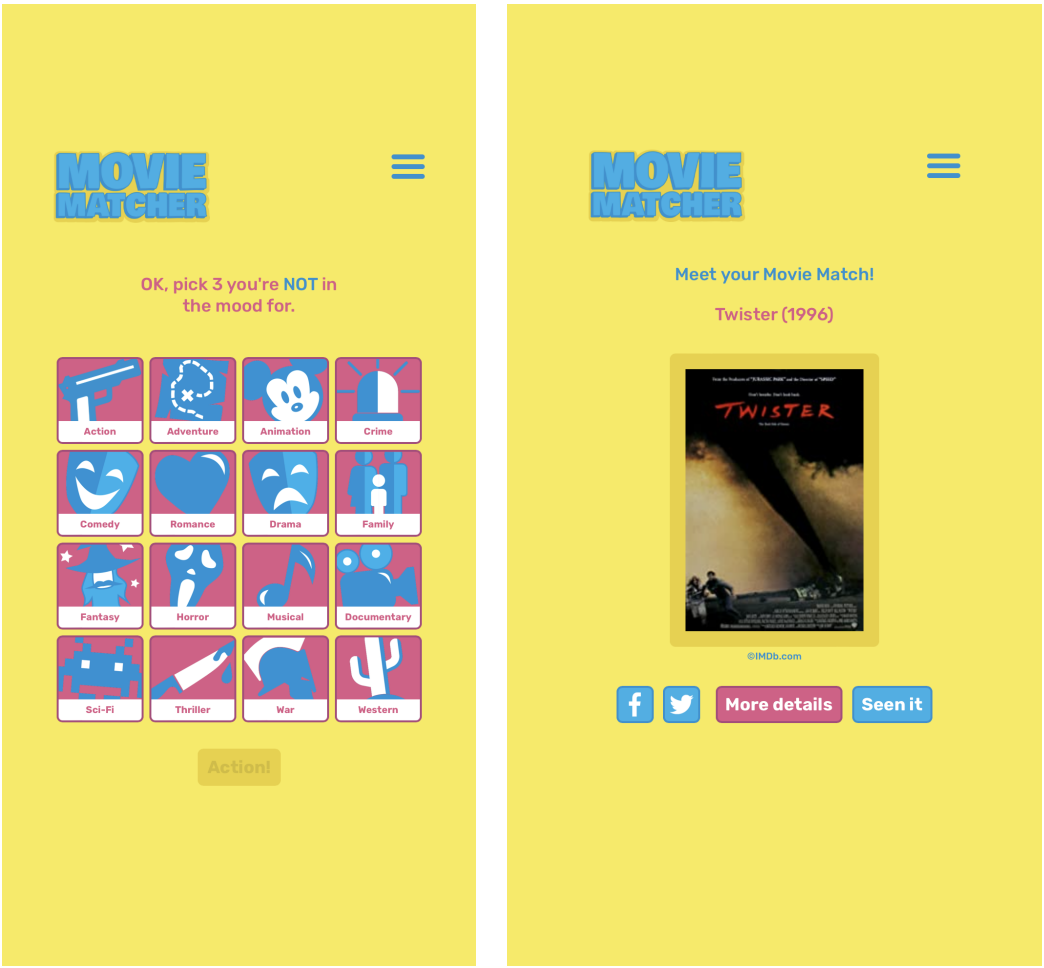
Existuje také řada algoritmů a aplikací, které uživatelům pomáhají s výběrem filmů. Podobné aplikace fungují také na způsobu vytváření skupin, které následně hlasují, aby nalezli film, který by mohli společně shlédnout. To však neznamená, že by pro aplikaci nebylo na trhu místo – jedná se totiž o zahraniční, převážně anglické, aplikace. Mnou vyvíjená aplikace bude mít nad ostatními převážně výhodu **české lokalizace**, tím se bude značně lišit od ostatních a otevřou se jí dveře do zcela nového trhu. Při tvorbě návrhů aplikace jsem se pokoušel vyvarovat chybám, které jsem při zkoumání aplikací našel a doplnil jsem nedostatky těchto aplikací.

Movie Matcher

Jedná se o jednoduchou webovou aplikaci, která má za účel najít film na společné shlédnutí pro dvojici. Uživatel zde může pozvat přítele, který se může připojit přes odkaz pro výběr filmu, nebo může vybírat film s filmovou postavou, která zde zastává roli přítele a vybere žánry adekvátní k postavě.

Co se týče funkcionality, použití aplikace je velmi jednoduché, stačí pouze vybrat 3 záporné a 3 kladné žánry pro obě strany. Nakonec algoritmus vybere nejvhodnější filmy, uživatelé mohou dále generovat filmy znovu, dokud se jim nebudou zamlouvat.

Aplikace má velmi povedené grafické zpracování s vlastními ručně kreslenými ikonami a grafikou, je jednoduchá a intuitivní na použití. Co se týče detailnějšího průzkumu uživatelského rozhraní aplikace, při výsledku hledání jsou tlačítka pro sdílení pro sociální sítě přímo v rovině s tlačítkem na generování nového filmu či pro zobrazení detailu o filmu, to může uživatele zmást a prodloužit tak dobu interakce s aplikací demonstraci uživatelského rozhraní aplikace Movie Matcher lze vidět na obrázku 3.1.



Obrázek 3.1: Pohled výběru žánrů a detailu snímku aplikace Movie Matcher

Date Night

Date Night je aplikace, která je zjednodušenou verzí aplikace Movie Matcher. Aplikace je určená převážně pro páry, které chtějí vybrat ideální film pro společný večer, jak již napovídá název či vzhled stránky.

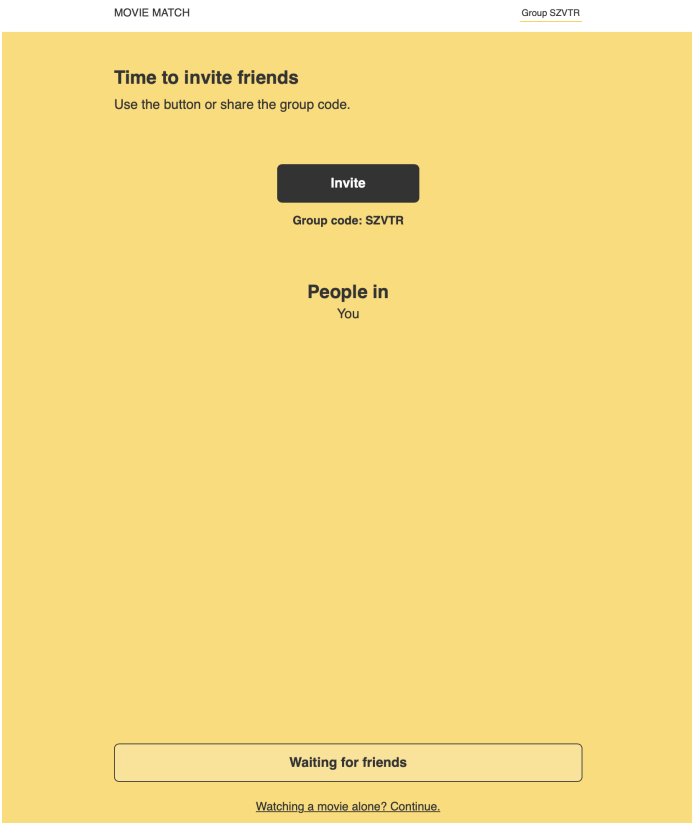
Co se týče funkčnosti, aplikace je velmi zjednodušená, tzn. vše se odehrává na jedné webové stránce, kde uživatel vybere film, který se každé dílčí polovičce páru zamlouvá. Na základě těchto vybraných filmů je uživateli představen výběr filmů, které mají obdobné žánry.

Tím, že aplikace uživateli nabízí několik filmů a ne pouze jeden výsledný film, mohou nastat další neshody při vybírání jednoho konkrétního snímku, jak lze vidět na obrázku 3.2.

Obrázek 3.2: Pohled prezentující výsledné snímky aplikace Date Night

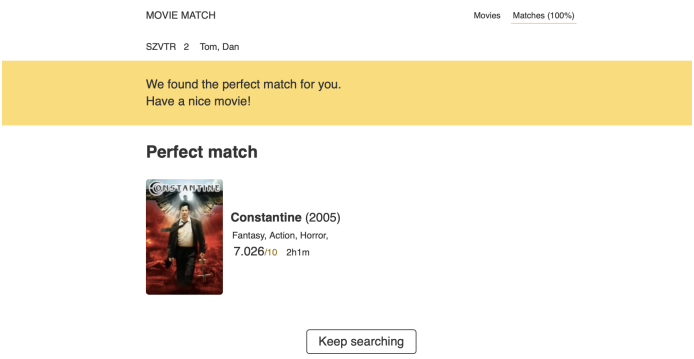
Movie Match

Narozdíl od předešlých dvou obdobných aplikací je aplikace Movie Match velice pokročilá, protože funguje na základě skupin (obrázek 3.3), do kterých se uživatelé mohou přihlašovat pomocí kódu. Při první návštěvě stránky návštěvník zadá své jméno a následně může buďto vytvořit skupinu nebo se připojit do již existující skupiny. Po připojení všech uživatelů do skupiny musí všichni stvrdit, že jsou připraveni stisknutím tlačítka. Následně jsou přemístěni na výběr žánrů, kdy každý z uživatelů vybere žánry, které by rád viděl. Po výběru žánru začíná proces hodnocení filmů. Uživatelé hodnotí film buďto kladně nebo záporně. V okamžiku, kdy se všichni uživatelé skupiny shodnou na stejném filmu, vyskočí zpráva s informací o tomto faktu a členové si poté mohou buďto tento film zobrazit nebo pokračovat v hodnocení. Ve skupině může být více filmů, u kterých nastala shoda.

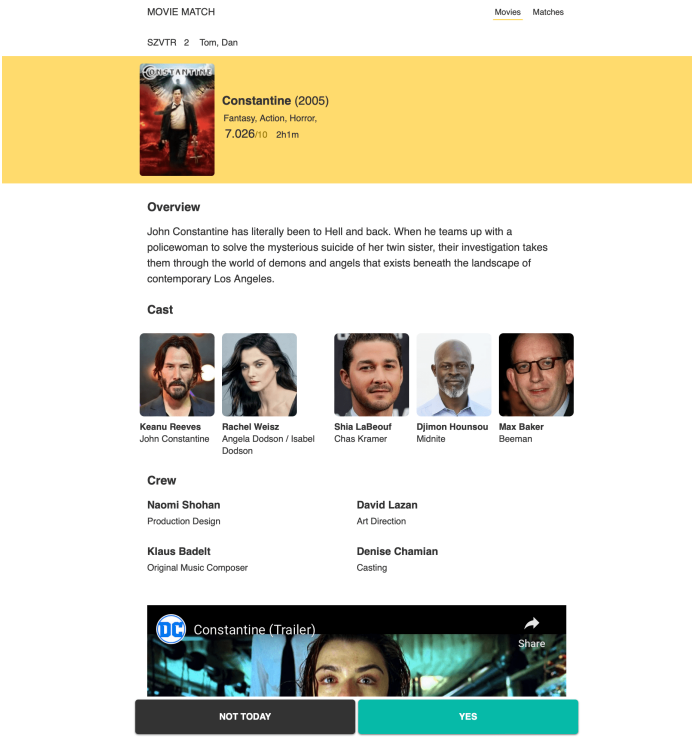


Obrázek 3.3: Pohled reprezentující detailu skupiny aplikace Movie Match

Aplikace funguje velmi dobře, jediný zápor aplikace je méně přívětivé uživatelské prostředí. Některé prvky jsou zde hůře viditelné a celý dojem z uživatelského rozhraní je strohý. Množství pohledů je zahlceno velkým množstvím informací na malém prostoru (viz. obrázek 3.5), či není plně využito místo, jak je tomu na obrázcích 3.3 a 3.4.



Obrázek 3.4: Pohled pro zobrazení výsledku perfektního hodnocení v aplikaci Movie Match



Obrázek 3.5: Pohled zobrazující detailní popis daného filmu v prostředí aplikace Movie Match

3.2 Uživatelský průzkum

Před zahájením šetření uživatelských potřeb je třeba si ujasnit, kdo je ideální uživatel aplikace, jeho potřeby a nároky jak na uživatelské rozhraní aplikace, tak na její funkcionalitu.

Ideální uživatel

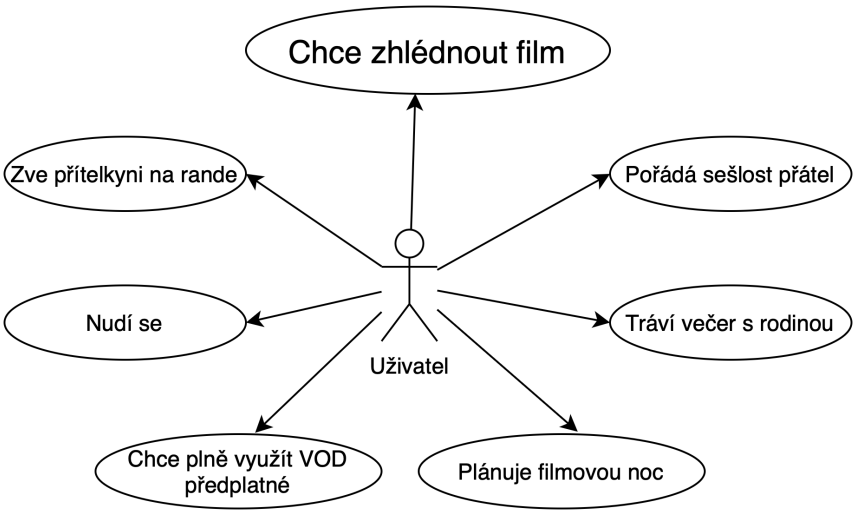
Ideální uživatel mé aplikace by měl být člověk ve věku kolem 20-30 let. Tento člověk je seznámen s moderními trendy mobilních aplikací, protože používá aplikace typu Instagram nebo Messenger. Je společensky aktivní, má středně velkou skupinu blízkých přátel, které si rád zve na návštěvy. Občasně na takových akcích shlédnou i film nebo díl jeho oblíbeného seriálu, avšak někdy této skupině přátel dělá problém se dohodnout, na který film se dívat. Tento problém zkoušeli vyřešit pomocí skupinových zpráv, ale nikdy se neshodli tak, že by byl každý spokojený, protože se ve vláknu zpráv nedalo vyznat.

Potřeby ideálního uživatele jsou jasné. Chce se dívat se skupinou svých přátel na film či seriál, protože se chtějí zabavit během času tráveného spolu.

Diagram případů užití

Před zahájením tvorby samotné aplikace jsem se rozhodl dále vytvořit diagram případů užití (Use Case Diagram – UCD). Pro první UCD (viz. obrázek 3.6) jsem zvolil tématiku směřující k reálným uživatelským potřebám a situacím, ve kterých se uživatel může nacházet, když

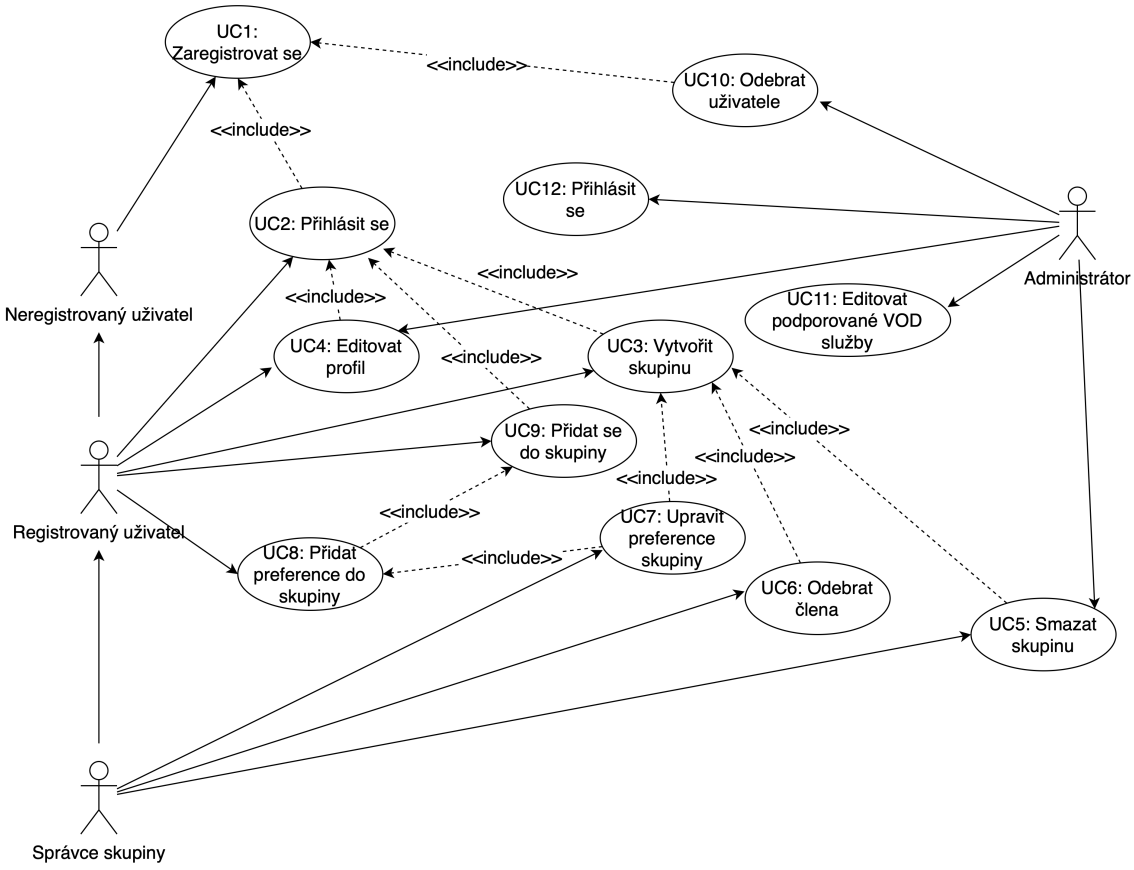
zapne aplikaci pro párování filmů. Můžeme jej chápat jako uživatelsky zaměřený diagram případů užití.



Obrázek 3.6: Uživatelsky zaměřený diagram případů užití

Uživatel je zde v roli, kdy chce zhlédnout film. Avšak jsou zde ještě další faktory, které nám mohou pomoci při navrhování aplikace. Uživatel tedy například může chystat filmový večer s přáteli, vybírat vhodný film pro rodinné sledování, pozvat svou drahou polovičku na schůzku anebo se jen nudí a chce vybrat adekvátní film jen pro sebe.

Po návrhu různých situací, ve kterých chce uživatel párovat filmy jsem sestavil technicky zaměřený diagram případů užití (viz. obrázek 3.7), který zahrnuje všechny možnosti použití aplikace. Pro demonstraci případů užití jsem zvolil 4 aktéry, a to: neregistrovaný uživatel (tento uživatel nebude moci využívat služby aplikace, dokud se nezaregistruje), registrovaný uživatel, který bude mít přístup k celé aplikaci, tj. připojení do skupiny a následné hodnocení vybraných filmů, úprava profilu a účtu. Registrovaný uživatel bude mít také možnost tvorby vlastní skupiny, čímž se stane jejím správcem a bude mít přístup k úpravě této skupiny a jejich členů.



Obrázek 3.7: Technicky zaměřený diagram případů užití reprezentující všechny role návrhu aplikace

Tvorba skupiny

Samotná tvorba skupiny by měla probíhat tak, že registrovaný a následně přihlášený uživatel vyplní formulář, kde uvede název skupiny, maximální počet členů a základní preference skupiny (komedie, akce, rozmezí data vydání, PG rating filmu, apod.). Tyto preference bude moci upravovat pouze správce skupiny. Po vyplnění se skupina uloží v databázi a aplikace vygeneruje zvací kód (unikátní identifikátor), pod kterým se další člen bude moci připojit do skupiny. Tento připojený člen bude nadále moci přidat své preference – tyto preference bude moci upravovat celá skupina – přidávat, smazat, editovat. Správce bude moci správcovství skupiny předat na libovolného uživatele nebo skupinu smazat úplně.

Hodnocení filmů

Hodnocení filmů (*swipování*[1]) bude v mé aplikaci probíhat následujícím způsobem:

1. Členové skupiny zvolí vhodné filtry
2. Správce skupiny započne proces hodnocení snímků

[1]Pojem „swipování“ vychází z aplikace Tinder a popisuje proces, kdy uživatel pomocí přejíždění prstu po displeji doleva či doprava hodnotí ostatní uživatele, v případě mé aplikace bude uživatel hodnotit snímky pomocí tlačítek.

3. Členové skupiny ohodnotí snímky kladně či záporně
4. Správce skupiny ukončí proces hodnocení
5. Výsledky hodnocení se zobrazí členům skupiny

Nejprve členové skupiny zvolí vhodné filtry, na základě kterých se budou snímky vybírat z databáze, dále správce skupiny započne proces hodnocení – stav skupiny se změní na stav „v hodnocení“. V této fázi nebude možné přidávat či odebírat členy. Do databáze se následovně nahrají z API či Firebase filmy, které splňují podmínky skupiny. Každý člen skupiny bude moci přejít do hodnotícího okna a udávat hodnocení konkrétnímu filmu z fronty filmů uložených v DB (líbí +1 bod, nelíbí -1 bod). Pokud některý film dostane počet kladných hodnocení stejný, jako je počet účastníků skupiny, film se vybere jako perfektní shoda a může být prezentován jako výsledek. V případě, že shoda nenastane může správce kdykoliv hodnocení zastavit a jsou mu odprezentovány výsledky (filmy a jejich ohodnocení). Po dokončení hodnocení je skupina navrácena do stavu před hodnocením, filmy jsou z DB uvolněny a je opět možno spravovat preference, uživatele, apod. Zároveň je možné zobrazit nejlepší výsledky posledního hodnocení.

Dotazníkové šetření

Po vytvoření ideálního uživatele a stanovení jeho potřeb může přijít na řadu sběr informací z reálného světa. Tyto informace hrají ve vývoji aplikací velkou roli, protože nám udávají obraz o reálných potřebách uživatelů. Tyto informace jsou často programátory zanedbány a aplikace poté nebývají uživatelsky přívětivé, bývají moc složité nebo minou podstatu reálných uživatelských potřeb.

Tvorba dotazníku

Při mém dotazníkovém šetření jsem volil aplikaci Forms od firmy Google[2]. Apliakce Google Forms nabízí možnost bezplatného vytvoření dotazníků pro sběr informací, její výhodou je jednoduché vytvoření i vyplnění. Zaručuje také anonymní sběr dat, respondenti si tak mohou být jisti, že jejich osobní data nejsou v ohrožení.

Při sestavování dotazníku je třeba dodržovat základní pravidla tvorby, a to:

- Odpovědi by měly pokrývat celé spektrum možností
- Co nejvíce se vyhýbat otevřeným otázkám

Dále je důležité si vytyčit, jaké odpovědi od respondentů požadujeme. V mém případě se jedná o technické informace typu:

- Operační systém uživatelů
- Jaké služby VoD (video na vyžádání) respondenti využívají

Především informace o operačním systému respondentů je pro tvorbu aplikace nezbytně nutná, udává totiž jednoznačně prostředí, pro které bude aplikace nabízena.

Větší procento dotazníku však pokrývají otázky na uživatelské preference v oblasti filmů, které budu využívat při tvorbě UX a doporučování filmů uživatelům, kdy musím tyto získané informace zohlednit, zároveň tak při hledání společných zájmů či žánrů.

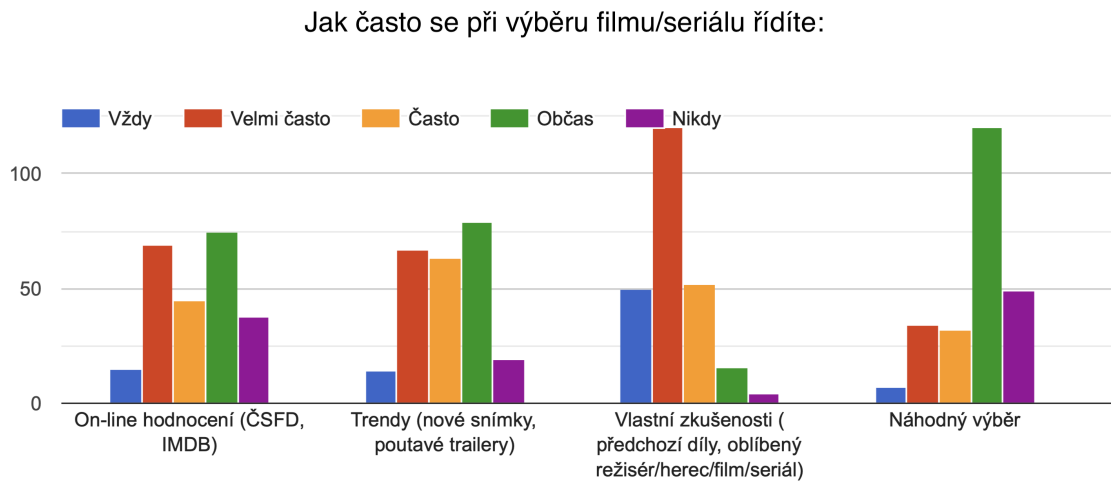
[2]Dotazník je dostupný ke dni 3. dubna 2023 na odkaze https://forms.gle/bVRzszcG7CWTJA24A

Sběr a interpretace dat

Na dotazník odpovědělo celkem **240** respondentů, z nich 82,1 % žen a 17,5 % mužů. Všichni z respondentů jsou ve věku od 16 let výše s tím, že 56,3 % z nich je ve věku 16–30 let. Toto rozmezí věku se dá považovat za mou primární cílovou skupinu, což potvrzuje užitečnost výsledků formuláře pro tvorbu mé aplikace.

V rámci průzkumu VoD služeb bylo zjištěno, že 58,3 % z dotázaných využívá službu Netflix, 22,1 % Disney+ a 21,7 % HBO GO. 30,8 % dotázaných VoD služby nepoužívá, to však neznamená, že služby mé aplikace nemohou využívat – stačí, když VoD službu vlastní alespoň 1 člověk ze skupiny. Zároveň se zjistilo, že 62,9 % respondentů využívá mobilní telefon s operačním systémem Android.

V případě dotazu na preference při výběru filmu či seriálu, na který se potenciální uživatel chce podívat, z něj vzešly následující výsledky:



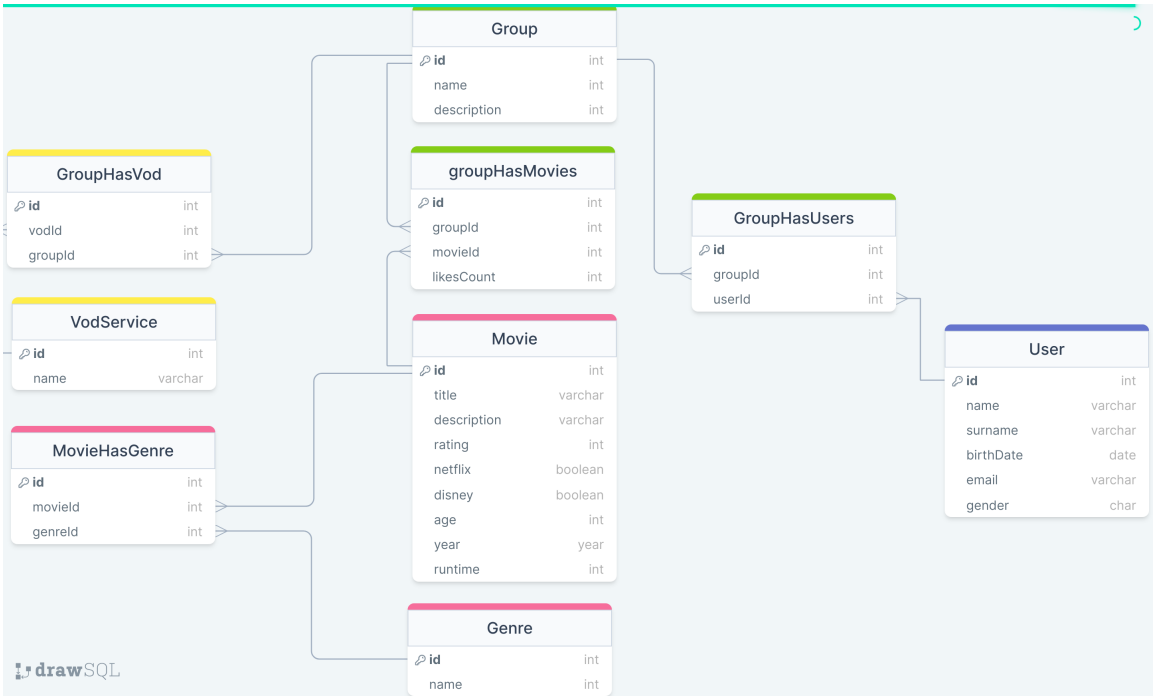
Obrázek 3.8: Výsledek odpovědí na jednu z otázek formuláře, která zjišťuje, čím se dotazovaní při výběru filmu nejvíce řídí

Z obrázku 3.8 lze vyčíst mimo jiné podstatné informace i to, že více jak polovina dotazovaných se velmi často řídí při výběru filmu vlastní zkušeností – ať už je to na základě oblíbeného herce, či předchozích dílů.

Tyto znalosti lze bohatě využít při nabízení uživateli potřebných informací, na základě kterých se může rozhodnout, zda by se na film rád podíval, či ne.

3.3 Ukládání dat

První myšlenka pro databázi byla ve formě SQL databáze, jelikož s SQL databázemi nabývám základních zkušeností pro vytvoření potřebné databáze. Počáteční návrh relačního diagramu entit pro SQL databázi lze vidět na obrázku 3.9.

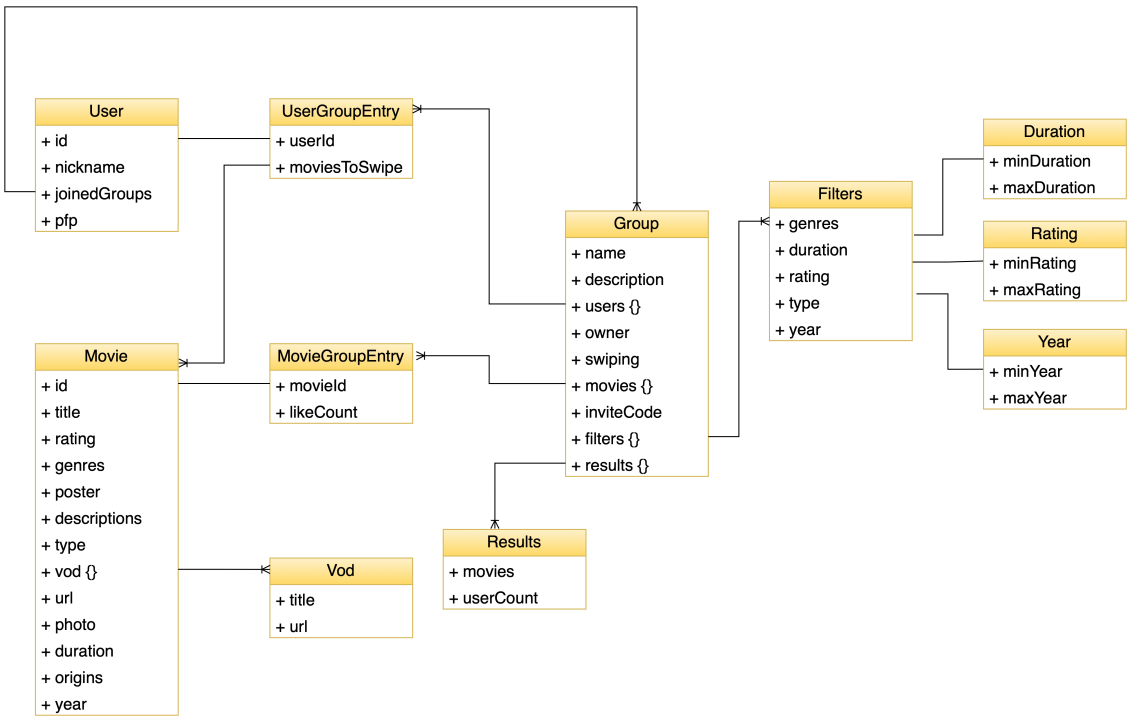


Obrázek 3.9: Návrh relačního diagramu entit pro SQL databáze

Po zkoumání výhod a nevýhod SQL databází a jejich alternativě ve formě NoSQL databází jsem se nakonec rozhodl použít právě technologii NoSQL databáze. Jako systém pro správu dat jsem se rozhodl používat Firebase od firmy Google (backend jako služba, viz. kapitola 2.4) – tato databáze má výhodu v tom, že jde přímo integrovat s Android Studiem, ve kterém budu aplikaci programovat. Zároveň lze se službami Firebase komunikovat takovým způsobem, že nebude potřeba vývoje separátního API, ale logika může být prováděna přímo v aplikaci ve vrstvě aplikační logiky.

Návrh databáze

Pro účely mé aplikace jsem sestavil také finální diagram popisující strukturu databáze, abych si ujistil, které objekty je nutno ve svém programu vytvořil a jak by mezi sebou měli komunikovat. Diagramovou reprezentaci sestavené databáze lze vidět na obrázku 3.10 (každá entita znázorněna v diagramu vlastní implicitní klíč, který ji jednoznačně označuje v rámci každé úrovně JSON stromu popisující databázi).



Obrázek 3.10: Diagram popisující strukturu NoSQL databáze a vzájemné komunikace entit

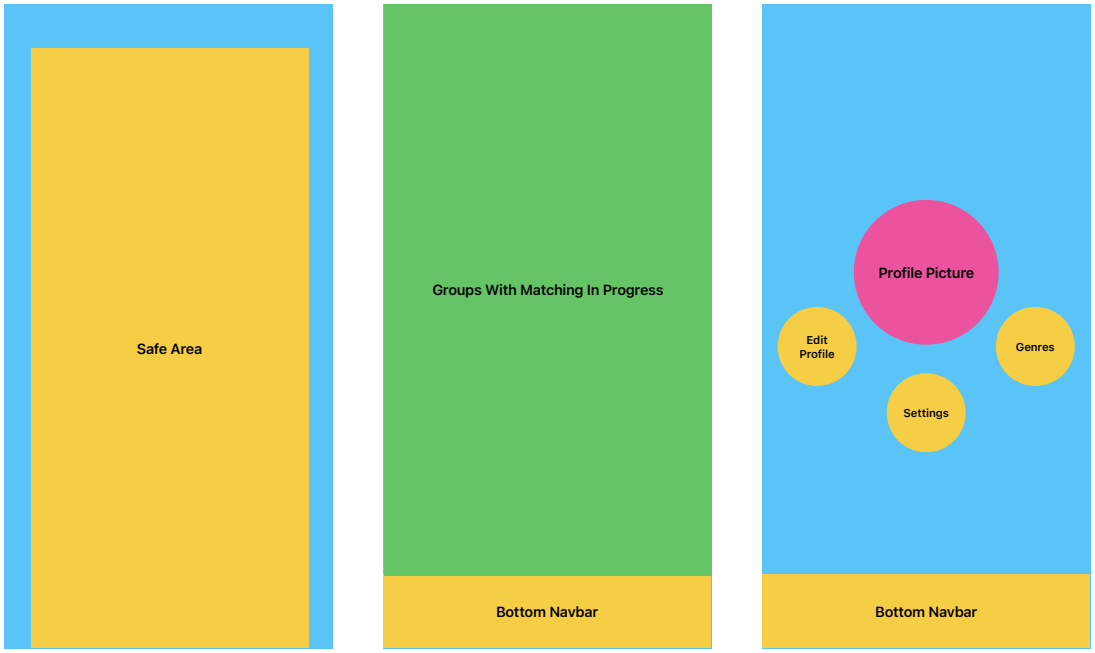
3.4 Uživatelské rozhraní

Poslední krok před vlastní implementací aplikace je dobré ujasnit si, jak bude výsledná aplikace rozložena, co všechno má uživateli zobrazit a jakým způsobem tyto informace mají být řazeny.

Kostra aplikace

Pro ucelení těchto návrhů je vhodné vytvořit základní abstraktní kostru aplikace (angl. wireframe). Tato kostra obsahuje pouze zjednodušené abstrakce finálních komponent výsledného uživatelského rozhraní. Tyto návrhy můžeme využít pro testování, zda se uživatel v aplikaci dokáže vyznat a vše je rozmístěno dle jeho očekávání. Vzhledem k tomu, že se jedná o Android aplikaci jsou návrhy kostry důležité – je totiž potřeba respektovat všechny možné tvary, rozměry a proporce, které jsou právě u platformy Android velice rozmanité, narozdíl od platformy mobilních telefonů Apple, které si drží své konvenční charakteristické tvary.

Tvorbu kostry jsem prováděl v aplikaci Apple Freeform, která nabízí tvorbu kompozic jednoduchých prvků včetně jejich popisů a vlastností.

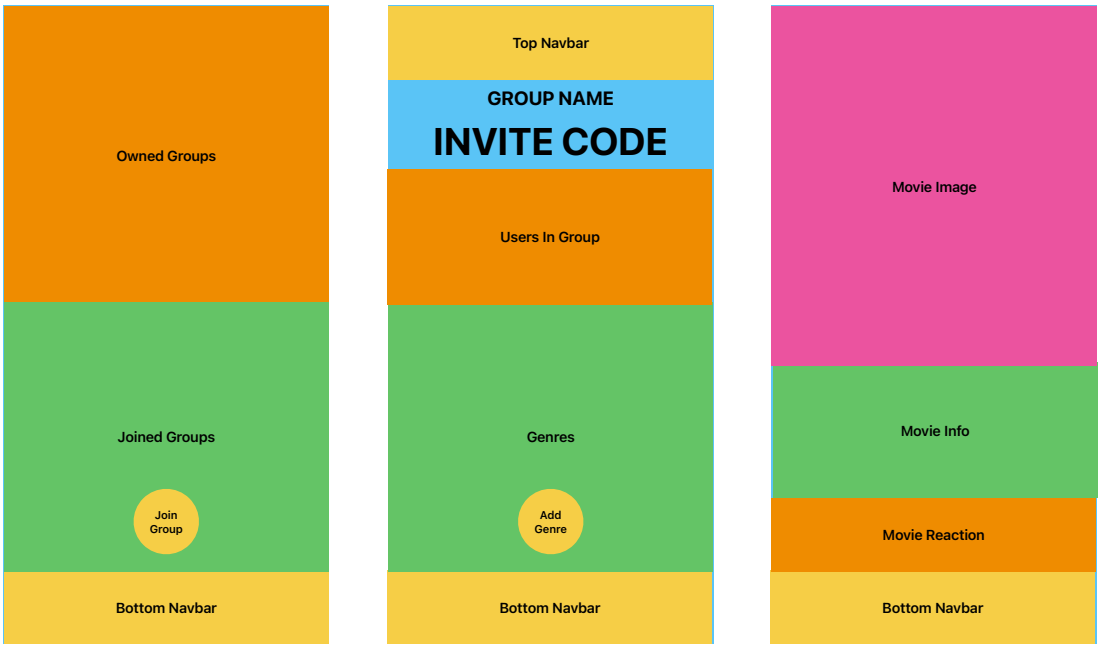


Obrázek 3.11: Návrh pro základní kostru aplikace, seznam aktivních skupin a uživatelský profil (zleva doprava)

První trojice návrhů (obrázek 3.11) reprezentuje jednodušší pohledy, ze kterých však vychází pohledy pokročilejší. Ze všeho nejdříve jsem si definoval pracovní oblast, kde se budou nacházet všechny komponenty uživatelského rozhraní. Jsou zde definovány boční odsazení, které berou v potaz fakt, že telefon uživatele bude mít potenciálně zahnuté okraje či ovládací prvky na kraji displeje (viz. Samsung „quick action“ boční panel). Mimo boční odsazení je zde implementováno i horní odsazení pro telefony s vykrojeným displejem (kamera v displeji). Následně jsem také navrhnul pohled, který zobrazuje všechny skupiny, které mají aktivní režim hodnocení pro rychlý přístup. V poslední řadě jsem v této skupině musel navrhnout profil uživatele, který slouží uživateli pro přístup k nastavení jeho profilu (jméno, profilová fotka, apod.), nastavení aplikace a potenciálně i upravení jeho oblíbených žánrů.

Navigace aplikací

V těchto návrzích je zároveň užit komponent navigační lišty, která je umístěna ve spodní části aplikace. Její úkol je zjednodušit uživateli průchod aplikací. Umístění navigace ve spodní části obrazovky je z důvodu jednoduché přístupnosti a malému úsilí uživatele při interakci.



Obrázek 3.12: Návrh kostry pro seznam skupin, detail skupiny a detail filmu (zleva doprava)

U složitějších pohledů se části obrazovky dělí na více částí z důvodu objemu obsahu, který je uživateli nutno zobrazit (např. u detailu skupiny). U seznamu skupin (obrázek 3.12 vlevo) jsou uživateli zvlášť zobrazeny skupiny, které sám vytvořil a ty, do kterých se připojil. Uživatel má k dispozici také tlačítko pro připojení se do skupiny pomocí kódu. Toto tlačítko zobrazí vyskakovací okno, kam uživatel bude moci zadat kód skupiny. Po zadání správného kódu bude uživatel přihlášen do této skupiny a přesměrován na její pohled. Pohled skupiny obsahuje základní informace o skupině, jako jsou název, kód pro pozvání uživatele, seznam připojených uživatelů a seznam žánrů, které byly vybrány členy skupiny. Každý člen bude moci přidat žánr do skupiny pomocí tlačítka a vybráním konkrétního žánru z vyskakovacího dialogu. V tomto pohledu je také nový komponent, kterým je horní navigační panel. Tento panel bude obsahovat tlačítko pro navrácení se zpět na seznam skupin, popřípadě i tlačítko pro vstup do nastavení skupiny pro jejího majitele (administrátora). V pohledu pro detail filmu při hodnocení jsou zobrazeny pouze základní informace o daném snímku zároveň s panelem pro uživatelovu interakci s filmem. Více informací o filmu lze zobrazit přes náležité tlačítko, které rozšíří oblast informací o snímku.

Testování kostry aplikace

Testování kostry návrhu probíhalo metodou pozorování potenciálního uživatele, který měl za úkol určit, o který pohled se jedná a popsat jejich komplexnost, popřípadě určit, jak rychle se v tomto návrhu orientuje.

Potenciálnímu uživateli byly postupně předvedeny pohledy, které byl schopen bez obtíží jednoznačně identifikovat. Co se týče navigace v aplikaci uživatel neměl výtky, navigace byla podle jeho slov jednoduchá, přehledná a vše bylo na svém místě. V oblasti rozložení komponentů si byl uživatel téměř jistý, vše zapadalo do návrhu dle jeho zkušeností z ostatních aplikací a obecným konvencím co se týče androidových aplikací obecně. Jediné výhrady,

které se týkají rozložení prvků, měl uživatel k pohledu, který reprezentuje zobrazení všech skupin – zde uživatel upozornil na absenci tlačítka pro vytvoření nové skupiny, v pohledu s detailem filmu uživatel varoval před příliš velkým prostorem pro obrázek filmu, v jeho případě se nejedná o rozhodující element.

Testování návrhu kostry mi pomohlo k lepšímu pochopení detailnějších uživatelských potřeb a zajistilo mi fakt, že následná šablona aplikace bude rozložena správně. Kladné i záporné poznatky z testování se zajisté promítnou v maketě aplikace.

Maketa aplikace

Po tvorbě kostry je dobré také vytvořit maketu aplikace, která bude blíže finální verzi. Moje verze makety obsahuje již konkrétnější informace s celkovým rozpoložením. První návrh této makety je pouze v neutrálních barvách z důvodu, že pořád netestujeme uživatelský dojem z aplikace, nýbrž její rozpoložení a uživatelskou přívětivost, co se týče jednoduchosti navigace a pohodlné orientace.

Maketu aplikace jsem tvořil v programu Figma (nástroj Figma přiblížen v kapitole 2.1). Z makety jsem sestavil interaktivní návrh aplikace, kterým může uživatel dynamicky procházet. Snímky z této makety lze vidět na obrázku 3.13.

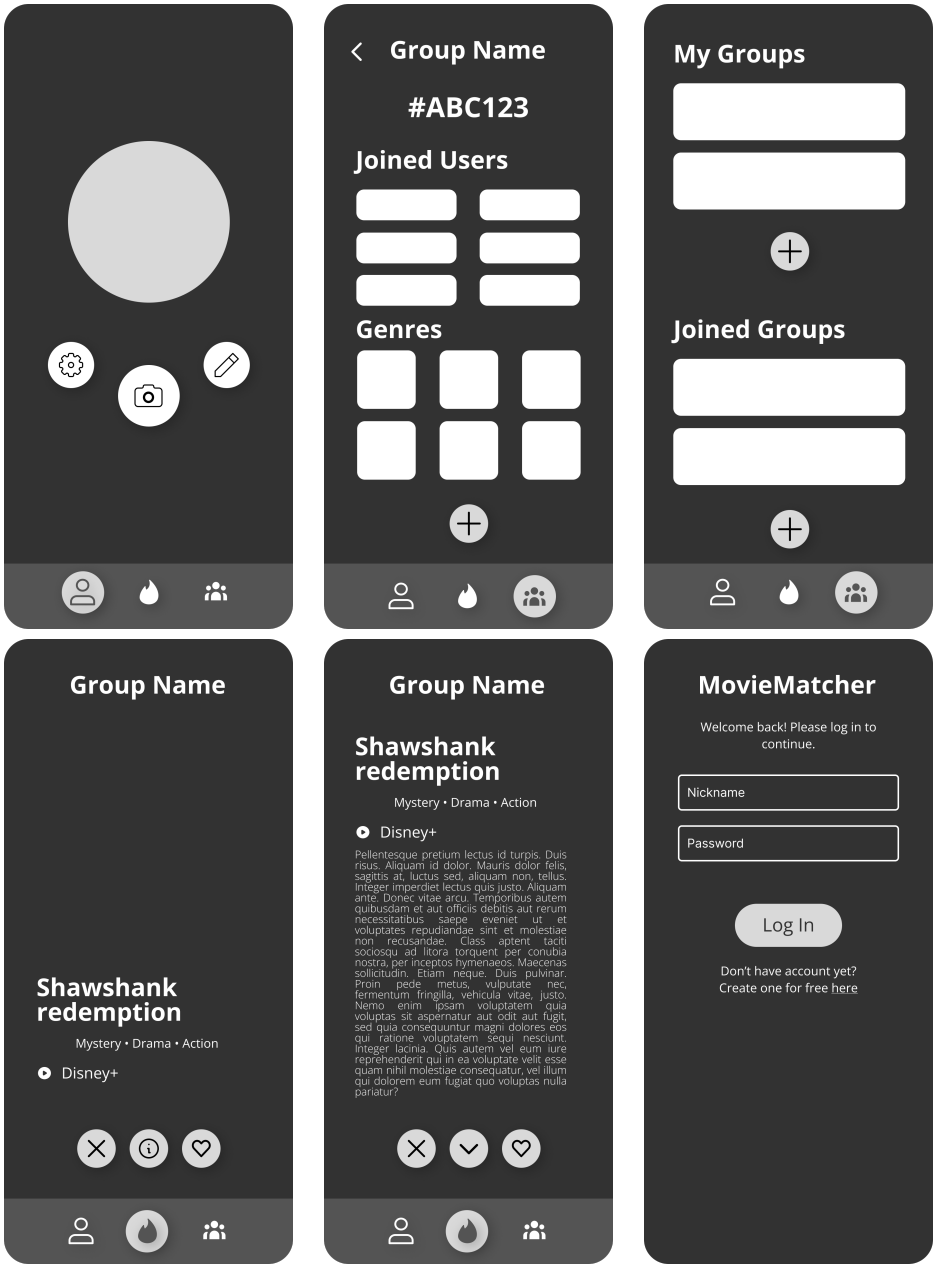
Při tvorbě makety aplikace je také nutno vzít v úvahu, že se jedná o mobilní aplikaci, tím pádem je nutno dodržovat základní konvence pro tvorbu takových aplikací (více v kapitole 4.2). Mezi základní konvence při takovém návrhu jsou:

- **Dostatečně velké interaktivní prvky** – tímto rozumíme tlačítka, karty, obrázky a další prvky, se kterými může uživatel, převážně dotekem, interagovat (kupříkladu spodní navigace). Tyto prvky by měly být dostatečně velké, aby uživatel měl velký prostor pro možnou chybovou odchylku doteku, která může vznikat v závislosti na velikosti prstu, nepřesností při navigaci prstu na displej a podobné aspekty, kvůli kterým taková odchylka vzniká. Tyto prvky by měly být zároveň zasazeny do kontextu návrhu tak, aby uživatel intuitivně jednoznačně mohl rozpoznat interaktivní prvek od prvku statického (využíváme tak barevné odlišení, konvence usazování takových prvků či nápovědu).
- **Dostatečně velký text** – v tomto smyslu musíme brát v potaz, že uživatel může naši aplikaci využívat v různých situacích – jedná se o mobilní zařízení, takže je jasné, že uživatel nebude vždy aplikaci využívat v ideálních podmínkách (světelné podmínky, vzdálenost uživatele od displeje), tomu musí adekvátně odpovídat i zvolená typografie pro projekt.

Testování makety

Testování navržené makety aplikace probíhalo opět stylem pozorování zástupce cílové skupiny aplikace, který prováděl nyní již interaktivní průchod aplikací.

Testování samotné makety prozradilo, že kostra aplikace byla navrhnuta dobře. Uživatel se v aplikaci pohyboval s přehledem a pohotově nacházel potřebné komponenty pro tuto navigaci. Přirozeně interagoval se spodním i horním navigačním panelem. Po pár úpravách co do pozicování elementů jsem se mohl vrhnout na implementaci konkrétní aplikace.



Obrázek 3.13: Pohledy znázorňující maketu aplikace v nástroji Figma

Kapitola 4

Implementace

Po vyhodnocení dotazníku a hotovém návrhu aplikace se lze posunout na stěžejní část, a to na samotnou implementaci návrhu.

4.1 Platforma

Jak již vyplynulo z dotazníku potenciálních uživatelů v kapitole 3.2, větší část respondentů využívá mobilní zařízení s operačním systémem Android, což také potvrzuje tvrzení grafu 2.1. Z tohoto důvodu jsem rozhodl, že aplikaci budu vyvíjet právě na tuto platformu. Zároveň z důvodu optimalizace, lepšího výkonu a možného budoucího rozšíření v Android ekosystému internetu věcí jsem zvolil, že potřeby mé aplikace bude více splňovat programování aplikace nativně.

4.2 Použité technologie

V průběhu implementace aplikace jsem využíval mnohé aplikace a knihovny (`Firebase SDK`, `Material Design 2`, `Material Design 3`, apod.). Pro vývoj mobilních aplikací existuje velká řada programů a knihoven, které umožňují programování aplikací na tato zařízení, před zahájením implementace je důležité vybrat právě ty programy, které jsou vhodné přímo pro mé požadavky.

Programovací jazyk

Z důvodu, že aplikace cílí na zařízení s operačním systémem Android je možnost výběru ze dvou hlavních jazyků, ve kterých lze programovat takovéto mobilní aplikace – Kotlin a Java.

Po průzkumu kladů a záporů obou jazyků jsem dospěl k závěru, že jazyk **Kotlin** se jeví pro účely mé aplikace jako lepší volba především z důvodu jeho efektivnosti a lépe vypadajícímu kódu, který přidává na srozumitelnosti a přehlednosti.

Zároveň jsem určil verze Android, s kterými je aplikace kompatibilní. Aplikace je kompatibilní s Android zařízeními, které mají verzi Android 6 až verzi Android 13. Díky tomuto rozpětí aplikace podporuje 98,2 % Android zařízení[1].

[1] Data aktuální k dubnu roku 2023, dostupné z odkazu `https://apilevels.com`

Prostředí

Jakožto prostředí pro vývoj aplikace jsem zvolil Android Studio (viz. kapitola 2.3) od firmy JetBrains z důvodu jeho přehlednosti, jednoduchosti použití a faktu, že je stavěný přímo na vývoj mobilních aplikací, tzn. má přímo zabudovaný kompilátor, emulátor a ladící nástroje, které usnadní celý průběh implementace a testování aplikace.

Rozdíly v návrhu a implementaci

Nejvíce znatelný rozdíl mezi návrhem aplikace a vlastní implementací je absence administrátorské role, která je popsána v diagramu 3.7. Tato role ve finální implementaci chybí z důvodu, že vzhledem k využití služby Firebase má administrátor aplikace k dispozici interaktivní administrační panel, ve kterém může provádět všechny akce zmíněné v diagramu případů užití. Tento panel je uživatelsky velmi přívětivý a dokáže se v něm zorientovat i technicky méně znalý člověk. Přidání takové role je v případě potřeby možno implementovat v dalších verzích aplikace, viz. kapitola 5.3.

Další znatelnou změnou je logika okolo hodnocení filmů. V kapitole 3.2 bylo demonstrováno, že film může být hodnocen kladně či záporně (+1 bod nebo -1 bod). Ve finální implementaci jsou filmy hodnoceny buďto kladně, nebo vůbec. Vzhledem k tomu, že se hledá ideální shoda, můžeme záporná hodnocení zanedbat a pracovat jen s nezápornými (+1 bod +0 bodu). V případě, kdy by se hledala nejlepší i nejhorší shoda, bylo by nutné zachovat i záporné hodnocení.

Při začátku hodnocení filmů se mimo změnu stavu skupiny do objektu dané skupiny v databázi uloží seznam snímků, které jsou získány filtrací a seznam uživatelů se snímky, které konkrétní uživatelé ještě nehodnotili.

Snímky v aplikaci se také filtrují pouze podle roku jejich vydání, jejich hodnocení na internetu a žánrů. Je tomu tak z důvodu, že ač je vzorek dat velký (500 snímků zastupující různé kategorie jak žánrů, hodnocení, tak i let vydání), tak není dostatečně velký na to, aby se dalo filtrovat pomocí konkrétnějších prvků (například podle dostupných služeb videa na vyžádání).

Ostatní rozdíly nejsou markantní, jedná se kupříkladu o posunutí prvku na jiný pohled či odlišnou implementaci se stejným výsledkem.

Řízení báze dat

Pro řízení báze dat jsem zvolil platformu **Firebase**, která nabízí především hostování NoSQL databáze, který v mém projektu využívám. NoSQL databázi jsem zvolil z důvodu její flexibility a snadné reakci na změny v databázi ve formě naslouchačů (angl. *listenerů*), které naslouchají změny v databázi a za pomocí zpětného volání (angl. *callback*) metody na ně lze v programu jednoduše a rychle reagovat v reálném čase za běhu aplikace.

Konkrétní služba, kterou jsem využil pro komunikaci s databází je **Realtime Database** z nabídky služeb Firebase. Nabízí komunikaci mezi koncovým zařízením uživatele a databází v reálném čase. Realtime Database naslouchá na zvoleném uzlu databáze na probíhající změny, které detekuje a aplikace na ně reaguje. Tato funkcionalita je velmi užitečná například v případě, kdy více uživatelů upravuje naráz atributy jedné skupiny (společné žánry), chceme totiž, aby se na výběru žánrů mohli podílet všichni uživatelé, nejen majitel. Další případ použití, který je ideální pro Realtime Database je naslouchání na „match“ skupiny, tj. okamžik, kdy všichni uživatelé kladně ohodnotí jeden daný film. V tomto případě chceme, aby služba naslouchala na každou změnu ve filmech, ze kterých uživatelé vybírají a

při změně aplikace zkontrolovala, zda nenastala situace, kdy by počet kladných hodnocení filmu byl stejný s počtem uživatelů ve skupině.

Zároveň také nabízí služby pro autentizaci a autorizaci uživatelů, která je zabudována v systému Firebase. Pro autentizaci a autorizaci uživatelů v mé aplikaci je tato služba velmi přínosná z důvodu přímé integrace s hostovanou NoSQL databází, bezpečnosti a převážně spolehlivosti. Další služby, které Firebase poskytuje lze najít v kapitole 2.4.

Pro komunikaci s databází jsem využíval nástrojů Firebase SDK, které poskytuje prostředí pro komunikaci s Firebase přímo pro Kotlin. Pro připojení k databázi jsem po nastavení samotného rozhraní využíval postupu, kdy jsem nejprve databázi definoval pomocí odkazu, aby bylo jasné, na kterou databázi se připojuji[2]. Samotná instance databáze ukazuje na kořenový prvek, z této instance lze přistupovat k zanořeným položkám pomocí unikátních klíčů, který každý prvek vlastní.

Po instanci databáze lze vytvořit více referencí (reference na databázi s uživateli, reference na databázi s filmy, apod.). Ke každé z těchto referencí lze přidat naslouchač[3] (angl. *Listener*), který naslouchá na každou změnu v dané referenci a při takové změně informuje program o této skutečnosti. Tento proces naslouchání je řešen ve speciálním objektu, který obsahuje funkce pro naslouchání různých částí databáze. Při jejich změně je volající této funkce informován o změně pomocí rozhraní (angl. *interface*), jak lze vidět v ukázce kódu 4.1, který informuje objekt `groupController` o požadavku kontinuálního naslouchání skupiny s kódem `mCode`.

```
groupController.listenGroupWithCode(object: GroupsController.GroupStatus{
    override fun groupsListening(...) {
        ...
        // Zmena ve skupine (nacist skupinu znovu)
    }
    override fun foundKey(key: String) {
        mKey = key // Nove ziskany klic
    }
    override fun handleDatabaseError() {
        activity!!.finish()
    }
}, mCode!!)
```

Výpis 4.1: Funkce pro naslouchání skupiny s určitým kódem

Grafické uživatelské rozhraní

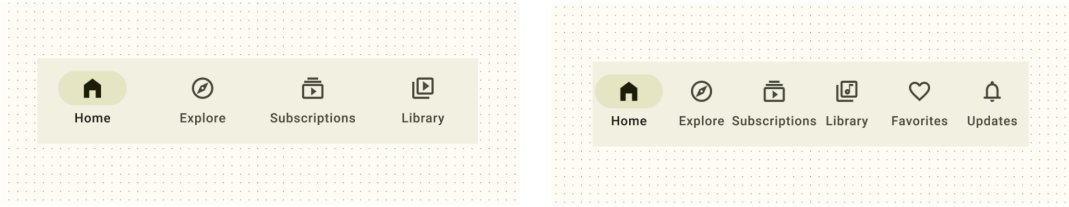
Pro tvorbu GUI prvků jsem využíval standardní elementy, které poskytuje knihovna **Material Design** [4], která se specializuje na poskytování standardizovaných prvků pro jednotný pocit z používání různých aplikací. Material Design krom těchto komponent (entit) poskytuje podrobnou dokumentaci, která jasně udává, jak by se komponent měl, či neměl používat pro přehlednost, srozumitelnost a jednoduchost výsledného prostředí.

V mé práci bylo využíváno elementů jak z Material Design 2, tak z nové verze Material Design 3 a byly co nejvíce dodržovány standardy jejich používání.

[2] Rozumíme tím jedinečný HTTP odkaz směřující na konkrétní databázi, v mém případě se jedná o řetězec https://cinematchingapp-default-rtdb.europe-west1.firebasedatabase.app

[3] Lze přidat naslouchač, který kontroluje změny neustále, nebo který pouze načte data jednou a dále již nenaslouchá

Kupříkladu na obrázku 4.1 lze vidět, jak vypadá dle konvencí Material Design 3 správné využití prvku pro spodní navigaci v aplikaci. Levé (správné) využití obsahuje pouze 4 základní prvky, je tedy přehledné a uživatel jednoduše vybere správnou záložku. V případě pravého (špatného) využití tohoto menu lze vidět, že obsahuje šest prvků, které jsou velmi natěsno naskládány u sebe, což způsobuje sníženou přehlednost a celkově špatný dojem z nabídky, nemluvě o faktu, že je zde větší pravděpodobnost situace, kdy uživatel omylem zvolí jinou položku, než zamýšlel, kvůli malým rozměrům každé položky spodní navigace stránky.



Obrázek 4.1: Správné použití spodního menu (vlevo), nesprávné použití (vpravo) [4]

Po nastudování dokumentace pro elementy, které jsem chtěl používat jsem je důsledně implementoval tak, aby co nejvíce splňovaly implementační podmínky. Tento fakt lze vidět kupříkladu na vlastní implementaci navigačního panelu v aplikaci samotné (obrázek 4.2).

Obrázek 4.2: Vlastní implementace spodní navigace, která ctí normy Material Design 3

Verzování, záloha, pracovní postup

Pro verzování projektu jsem využíval nástroj Git, konkrétně službu GitHub, která poskytuje všechny potřebné služby pro verzování, zálohování a další akce, nástroje, které jsou pro aplikaci tohoto rázu důležité, některé až nezbytné.

Implementace dílčí funkcionality při tvorbě projektu vždy odrážela název konkrétní Git větve, která se při dokončení implementace konkrétní části spojila do větve `main`, která zaštiťována produkční vrstvu aplikace. Toto spojení bylo vždy realizováno pomocí tzv. *pull-requestů*.

Ve službě GitHub jsem zároveň nastavil kontinuální integraci (angl. *CI Pipeline*) přímo pro Kotlin (potažmo i Java) Android aplikace, která se vždy při *pull-requestu* do větve `main` pokusila v uzavřeném prostředí (kontejneru) nainstalovat potřebné nástroje (především nástroje `Gradle` a `JDK 11` pro kompilaci programu). Následně se v kontejneru pokusila sestavit a přeložit celý program a spustit automatické testy, aby se odhalily případné kompilační chyby, kdy je tuto verzi nepřípustné publikovat. V tomto případě je nutno kód prozkoumat, případné chyby najít, eliminovat a pokusit se o tento automatizovaný překlad znovu. Tento proces se případně opakuje, dokud se vyskytují chyby v kódu. Pro předcházení takových chyb slouží kompilátor, který je přímo v Android studiu. Kód jsem kompiloval průběžně zpravidla po přidávání či úpravě funkcí, logických částí, objektů nebo změn ve struktuře kódu. V případě, kdy program projde kontinuální integrací bez chyb (situace, kdy

se program úspěšně zkompiloval bez výskytu varování či chyb) je aplikace hodna vydání do produkce a tím pádem je možno ji spojit s hlavní větví.

4.3 Implementace aplikační logiky

Po velmi detailním návrhu aplikace i technologií jsem výslednou aplikaci programoval s tím, že jsem se snažil co nejvíce využít výše zmíněné technologie tak, aby výsledná implementace byla co nejbližší návrhu.

Vycházel jsem jak z grafických předloh pro uživatelské rozhraní, tak například z diagramu databáze z obrázku 3.10 tím způsobem, že jsem modely využívané v aplikační logice mapoval takřka 1:1 k jejich vyobrazení v tomto diagramu.

Autentizace

Autentizace pro aplikaci MovieMatcher je zaštiťována službou Firebase, převážně z důvodu přímé integrace jak v Kotlinu, tak v Android Studiu, kdy je pracovní postup přímo stavěný na účely implementace autorizace a autentizace uživatelů. Autentizační služby Firebase zároveň poskytují možnosti přihlášení pomocí různých služeb (Google účet, Facebook, apod.). V případě mé aplikace je však využíváno pouze přihlašování pomocí e-mailu a hesla. Úryvek kódu implementující registraci uživatele je možno vidět ve výpisu 4.2.

Při registraci je uživatel požádán pomocí vstupních boxů o poskytnutí jeho uživatelského jména, emailové adresy a kombinace dvou hesel, kdy se obě tato hesla musí shodovat pro úspěšnou registraci.

Podmínky registrace

Nároky na uživatelské jméno nejsou striktní – uživatelské jméno může být libovolný řetězec delší než 3 znaky a zároveň kratší než 20 znaků s tím, že může obsahovat alfanumerické, tečku a podtržítko (jméno musí začínat i končit alfanumerickým znakem nebo číslem). Uživatelské jméno nemusí být unikátní (narozdíl od e-mailové adresy, pomocí které se přihlašuje a registruje) z důvodu, že se podle něj v databázi nikdy neindexuje – indexuje se pomocí unikátního identifikátoru (UID) uživatele.

E-mailová adresa musí mít správný formát – musí se jednat o reálnou e-mailovou adresu, tj. adresa ve tvaru formálně popsaném regulárním výrazem. Tato adresa musí být zároveň unikátní pro každého uživatele.

Co se týče hesla, musí se jednat o řetězec alfanumerických znaků a množiny vybraných ASCII znaků. V případě, že se tento řetězec shoduje s řetězcem poskytnutým uživatelem v poli pro zopakování hesla, je dané heslo považováno za validní.

Heslo zároveň musí z bezpečnostních důvodů obsahovat:

- 6–30 znaků
- Alespoň jedno velké písmeno
- Alespoň jedno číslo

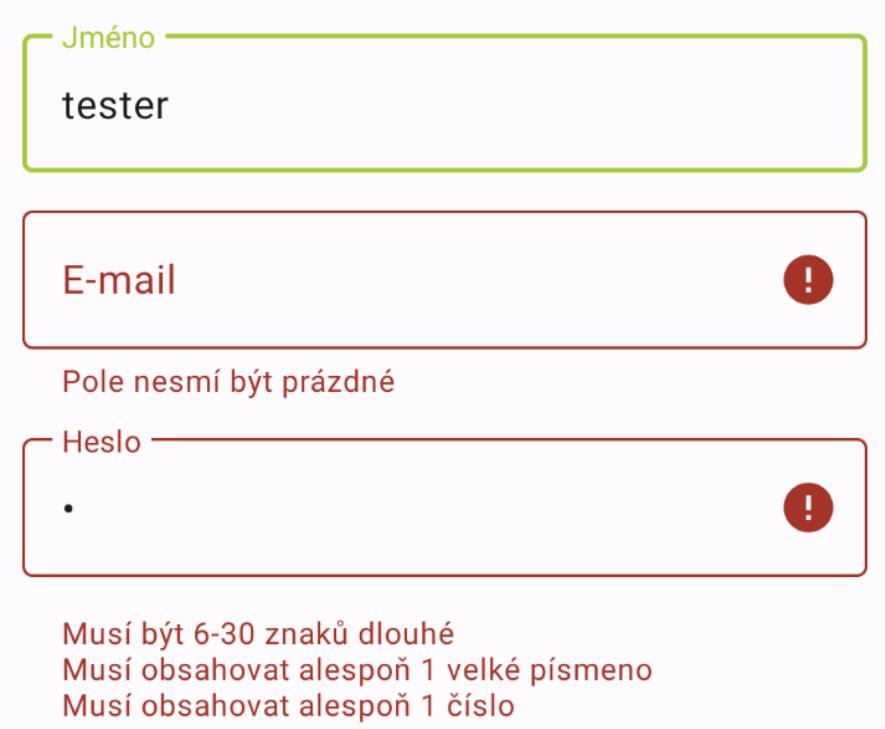
```
auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            val user = auth.currentUser
            Utils.showShortToast(...)
            try {
                userController.createUser(User(user!!.uid, username))
            } catch (e: Exception){
                Utils.showShortToast(...)
            }
        } else {
            ...
        }
    }
```

Výpis 4.2: Registrace uživatele pomocí Firebase service

Kontrola vstupů

Kontrola uživatelem poskytnutých vstupů při registraci probíhá jak na straně aplikační logiky, tak na straně Firebase. Při potvrzení registračního formuláře jsou údaje validovány prvním kontrolním blokem v aplikační logice. Jedná se o kontrolu validního uživatelského jména a shody hesel.

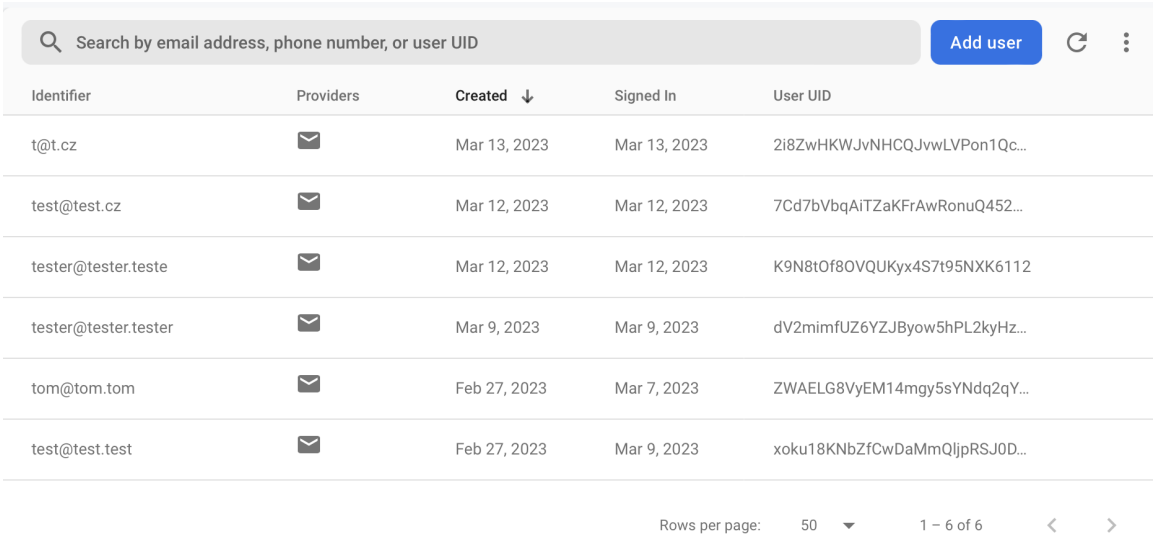
V případě, že nějaký z údajů nesplňuje podmínky, je uživatel o této skutečnosti adekvátně informován a má možnost tyto chyby opravit. Při validní kontrole údajů je poslán požadavek na autorizační služby Firebase, která při úspěšné validaci údajů provede další logiku potřebnou pro úspěšné zařazení nového uživatele do databáze, v opačném případě přijde z Firebase odpověď s atributem `task.isSuccessful` nastaveným na `false`, podle čeho lze poznat, že registrace uživatele proběhla neúspěšně. Důvod neúspěšnosti registrace lze zjistit pomocí dalšího atributu, který přijde s odpovědí – `task.exception` – který jednoznačně určuje důvod, kvůli kterému registrace neproběhla úspěšně. Na základě těchto informací je možné v aplikační logice zachytit takovou událost a adekvátně na ni zareagovat a informovat uživatele o neúspěšném stavu této transakce s popsáním problému, který nastal. Uživatel má opět možnost údaje změnit a po změně transakci opakovat. Příklad dynamické kontroly vstupů je možno vidět na obrázku 4.3.



Obrázek 4.3: Příklad různých stavů dynamických textových polí

V rámci procesu registrace nového uživatele se vytvoří nový záznam v autentizační databázi Firebase (viz. obrázek 4.4), který obsahuje pouze základní údaje důležité pro autentizaci (e-mail, pod kterým byl účet založen, služba, pod kterou byl účet založen (e-mail a heslo, Facebook, Google, Apple ID, apod.), informace o časech, kdy se uživatel registroval, naposledy přihlásil a velmi důležité User UID, což je unikátní identifikátor uživatele v rámci systému). Kromě záznamu do této tabulky se vytvoří záznam v Realtime Databázi, který vytváří nový objekt (uzel) ve stromu uživatelů. Tento uzel slouží pro uchovávání dalších údajů o uživateli, které nelze uložit v autentizační databázi Firebase. Pro můj případ užití postačí pouze uživatelské jméno pro lepší identifikaci uživatele v rámci skupiny, seznam připojených skupin, který slouží pro optimalizaci vyhledávání skupin, ve kterých je uživatel připojen a profilový obrázek převedený do base64[4] kódování pro jeho snadné uložení ve formě textového řetězce.

[4]Base64 kódování spočívá v převádění každých 3 původních bajtů na 4 kódované znaky ASCII



Obrázek 4.4: Autentizační tabulka v prostředí Firebase

Autorizace

Autorizace je velmi úzce spojená s autentizací. Pro autorizaci jsem ve své aplikaci využíval dva hlavní postupy:

- Autorizace na straně **aplikace** byla využívána převážně v případech zobrazení skupiny. Zde byla autorizace potřebná pro zajištění, že jenom uživatel smí měnit určité kritické aspekty skupiny (mimo filtrů). Jedná se například o změnu jména skupiny, popisu skupiny či odstraněni uživatele ze skupiny. Tyto případy jsou řešeny v aplikační logice pomocí kontrol, kupříkladu zda se UID aktuálního uživatele rovná s UID majitele skupiny. Pokud ano, uživatel je oprávněn tyto údaje změnit, v opačném případě mu nejsou vůbec prezentovány (obyčejný uživatel připojený do skupiny nevidí tlačítko pro odstranění ostatních uživatelů, kdežto vlastník skupiny ano).
- Autorizace na straně služby **Firebase** je využívána pro případy, kdy není přípustné, aby určitá skupina uživatelů přistupovala do databáze k datům, která jsou kritická. Tato opatření jsou nastavována přímo ve Firebase administraci pomocí pravidel, které jsou zapsány v JSON formátu a spravovány pomocí interaktivního editoru (lze vidět ve výpisu 4.3), který mimo jiné kontroluje i validitu těchto pravidel. V mém případě jsem nastavil základní pravidla:
  - Čtení i zápis do databáze je oprávněn pouze přihlášeným uživatelům. Pokud přijde na databázi požadavek s prázdným UID uživatele, není obsloužen a do aplikační logiky je navrácena chyba, která značí, že dotaz nebyl proveden, jelikož nebyly splněny požadavky pro autorizaci.
  - Z databáze filmů se smí pouze číst, nelze do ní zapisovat. Úprava databáze filmů je možná v interaktivním prostředí Firebase, popřípadě lze v další verzi aplikace integrovat administrátorský panel, kde by bylo možné filmy upravovat.

```
{
  "rules": {
    ".read": "auth.uid != null",
    ".write": "auth.uid != null",

    "groups": {
      ".indexOn": ["inviteCode"]
    },
    "movies": {
      ".indexOn": ["id"],
      ".write": false
    },
    "users": {
      ".indexOn": ["id"]
    }
  }
}
```

Výpis 4.3: Soubor pravidel pro databázi

Při každé další změně aplikace je také volána funkce, která kontroluje, zda je uživatel stále přihlášen a je tedy oprávněn užívat aplikaci. V případě, kdy uživatel přihlášen není, je automaticky převeden na přihlašovací obrazovku, ve které je pro další užívání aplikace nucen se opět přihlásit. Tato situace může nastat, když aplikace není po delší dobu využívána a sezení (angl. *session*) uživatele je přerušeno.

4.4 Filtrace výběru filmů

Filtrace filmů či seriálů z databáze Firebase probíhá přímo v aplikační logice, kdy se berou v potaz filtry, které sestavují členové skupiny. Konkrétními filtry jsou žánry, rok vydání a hodnocení filmu. Filtry jsem zároveň rozdělil do dvou kategorií, a to:

- **Ořezové filtry** jsou filtry, které při nesplnění podmínky, kdy se filtr skupiny nerovná s atributem filmu, tento film zahodí a dále již není zpracován. Tímto tedy rozumíme situaci, kdy například skupina s filtrem, který obsahuje rozmezí let 2000 až 2023 při filtraci zahodí film, který je z roku 1990. Analogicky je tomu tak při filtrování hodnocení filmu.
- **Doplňkový filtr** je druh filtru, který filmy neořezává, ale spíše doplňuje již vybranou skupinu filmů tak, aby ve výsledné skupině byly právě ty filmy, které nejlépe splňují požadavky skupiny. Do této kategorie spadá filtrace žánrů skupiny, kdy se při postupném načítání filmů z databáze kontroluje, jak moc jsou žánry skupiny shodné s žánry daného filmu. Na základě toho se film buďto zařadí do skupiny vyfiltrovaných filmů, nebo se zahodí.

Filtry jsem takto rozdělil z toho důvodu, že v libovolném rozmezí jak let, tak hodnocení se téměř vždy najde určitý počet filmů, avšak ne vždy se všechny žánry skupiny budou shodovat se žánry, které obsahuje položka v seznamu filmů. Tímto faktorem by bylo způsobeno, že filtry skupiny by musely být velmi obecné, aby se vůbec nějaké filmy do skupiny načetly.

Samotná filtrace

Programové řešení filtrace filmů spočívá ve smyčce, ve které se načítají filmy z databáze filmů. Před načítáním dat z databáze se vytvoří prázdný seznam, který reprezentuje výsledné vyfiltrované filmy. Při načítání se pro snížení objemu získaných dat využívá paginace[5] – filmy se tedy načítají po částech (po 50 filmech najednou). Tato část filmů se následně zpracuje a přepíše se další skupinou dat pro filtraci. Toto načítání probíhá až do načtení všech dat z databáze filmů. Při načtení jednoho výřezu dat z databáze se prochází každý dílčí prvek (film). Nejprve je kontrolováno, zda se rok vydání a filmu shoduje s filtry, které jsou obsaženy v objektu skupiny. V případě, že se nerovnají, film je zahozen a pokračuje se zpracováním dalšího prvku ze skupiny dat z databáze. V případě, že se rok i hodnocení shoduje, je vyhodnocen průnik žánrů filmu se skupinou žánrů skupiny. Následně je vypočítána velikost tohoto průniku.

Posledním krokem při filtraci je kontrola, zda je seznam vyfiltrovaných filmů ve skupině plný. V případě, kdy tento seznam není plný, je film do tohoto seznamu zařazen zároveň s informací o počtu shodných žánrů. V případě, kdy je seznam plný se zkontroluje, zda se v této skupině nachází film, který má menší počet shodných žánrů – pokud ano, je tento film nahrazen novým filmem, jinak je tento nový film zahozen a program pokračuje načtením další položky získaných dat z databáze. Zjednodušený úryvek filtračního kódu lze vidět ve výpisu 4.4.

Následně jsou filtrované filmy seřazeny podle počtu společných žánrů z důvodu, aby uživatel viděl nejprve ty nejrelevantnější výsledky.

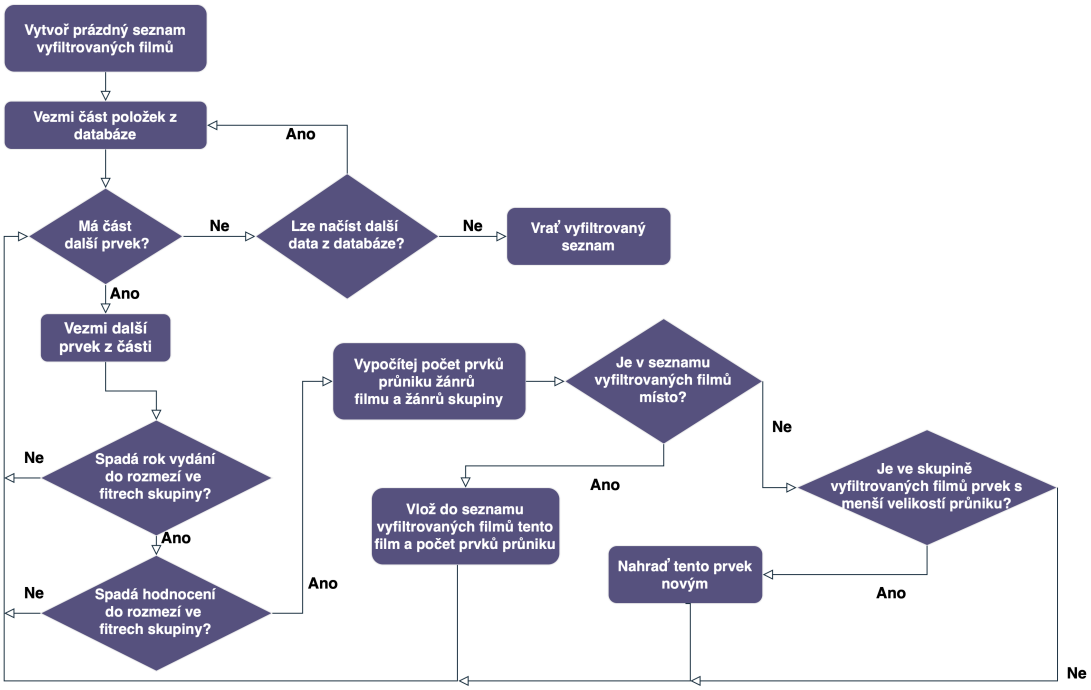
```
val intersect = filters.genres!!.intersect(tmpMovie.genres!!.toSet())
if(adding && filters.genres!!.any { it in tmpMovie.genres!! }){
    if(filteredMovies.size >= 30){
        if(filteredMovies.any { it.commonGenres!! < intersect.size }){
            var entryToRemove: MovieFilterEntry? = null
            for(filterEntry in filteredMovies){
                if(filterEntry.commonGenres!! < intersect.size){
                    entryToRemove = filterEntry
                }
            }
            if(entryToRemove != null){
                filteredMovies.remove(entryToRemove)
                filteredMovies.add(MovieFilterEntry(...))
            }
        }
    } else
        filteredMovies.add(MovieFilterEntry(tmpMovie, intersect.size))
}
```

Výpis 4.4: Proces filtrace na základě průniku žánrů

Tyto výsledné filmy (jejich identifikátory a počet kladných hodnocení) jsou poté zapsány do objektu skupiny jako atribut `movies`, zároveň jsou identifikátory těchto filmů připsány ke každému uživateli skupiny jako `movies_to_swipe` atribut. Tento atribut slouží k tomu, aby se po hodnocení filmu uživatelem odebral a tím pádem se zamezilo, aby uživatel hodnotil

[5] Načítání dat postupně po jednotlivých vzorcích

více filmů najednou, protože v okamžiku, kdy bude pokračovat v hodnocení, kdy mezitím projde jiné pohledy, je tomuto uživatele servírován seznam filmů, které ještě nehodnotil.



Obrázek 4.5: Vývojový diagram popisující kompletní proces filtrace filmů před zahájením hodnocení

Nabízení filmů uživateli

Po zahájení hodnocení má uživatel možnost hodnotit filmy kladně, či záporně (neutrálně). Při zahájení tohoto hodnocení je seznam filmů, které uživatel prozatím nehodnotil seřazen podle hodnocení ostatních filmů ve skupině, aby se mu nejprve zobrazovaly ty filmy, které byly v okamžik začátku hodnocení nejlépe hodnocené.

4.5 Implementace grafického uživatelského rozhraní

Při tvorbě grafického uživatelského rozhraní (GUI) jsem dbal převážně na interaktivitu prostředí – dynamické elementy, které uživateli usnadní průchod celou aplikací. Dobrý příklad takového elementu je například textové pole s následnou dynamickou validací vstupu poskytnutým uživatelem (demonstrace lze nalézt v sekci 4.3).

Při kontrole vstupu poskytnutých uživatelem vycházím z možnosti zachytit okamžik, kdy bylo vstupní pole změněno a na tuto skutečnost následně reagovat akcí – v mém případě validací. K zachytávaní okamžiku, kdy bylo pole změněno slouží událost zpětného volání `addTextChangedListener`. Při každé změně vstupního pole je volána adekvátní validační funkce, která nejprve zkontroluje validitu vstupu na základě regulárního výrazu, poté v případě správného vstupu ukončí svou funkci a uživatel není informován o výskytu chyby. V případě, kdy validace vstupu neuspěje, je nejprve vyhodnocena příčina chyby a uživatel je o této příčině informován.

Zároveň je také adekvátně zabarveno dané špatně vyplněné vstupní pole, aby se uživatel mohl jednoduše a rychle zorientovat a jednoznačně nalézt chybu, která se vyskytla při zadávání. Kódovou implementaci takového řešení lze vidět ve výpisu 4.5.

```
val emailEdit: EditText = findViewById(R.id.emailEditText)
emailEdit.addTextChangedListener {
    email = it.toString()
    validateEmail()
}
```

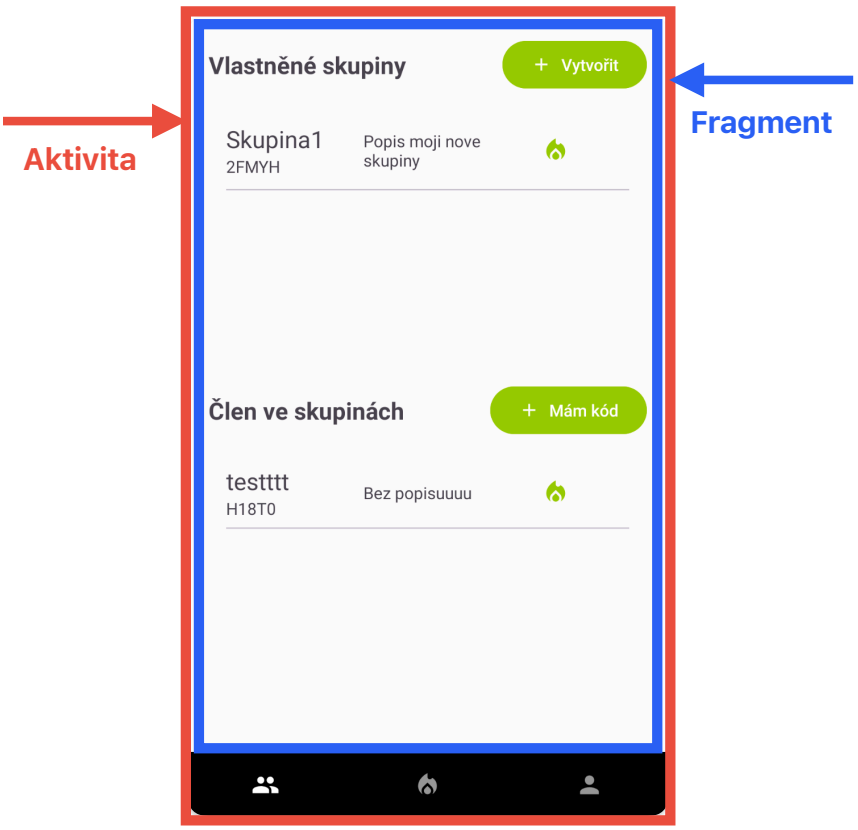
Výpis 4.5: Dynamická kontrola emailu, při které je po každé změně provedena kontrola vstupu

U všech pohledů jsem zároveň vycházel z makety aplikace co se týče jak obsahu zobrazeného na pohledu, tak jeho následného rozležení.

GUI se skládá ze základních stavebních prvků tvořících uživatelské rozhraní na platformě Android a to jsou aktivity a fragmenty.

- **Aktivity** jsou zpravidla pohledy zabírající celou obrazovku, kdy vždy na jeden pohled náleží právě jedna aktivita [7]. V mém případě se jedná například o aktivitu pro přihlášení uživatele, pro registraci uživatele, nebo aktivitu zobrazující hlavní pohled aplikace (viz. obrázek 4.6). Aktivity dále mohou obsahovat kromě základních dílčích komponent, jako jsou například textová pole a obrázky, také složitější komponenty složené z více dílčích prvků.

- Složené komponenty mohou být právě například **fragmenty**. Kontext fragmentů se může dynamicky měnit bez nutnosti změny hlavní aktivity [7]. Dobrý příklad správného využití těchto fragmentů je vidět na pohledu hlavního menu, kdy spodní navigací lze přepínat kontext tak, že se mění obsah fragmentů, nicméně pořád se jedná o tu stejnou aktivitu s tím stejným spodním navigačním panelem. Transakce fragmentů zajišťuje `Fragment Manager`. Tento objekt lze využít pro komunikaci s fragmenty, jejich přepínání a úpravy.

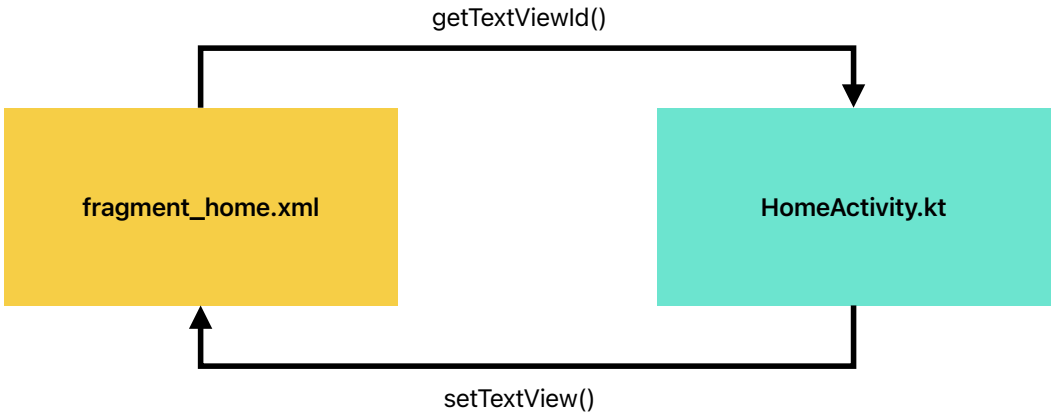


Obrázek 4.6: Demonstrace fragmentu a aktivity v rámci hlavního pohledu aplikace

Na obrázku 4.6 lze také vidět přímou implementaci obou těchto základních stavebních bloků – aktivity i fragmentu. Aktivita zabírá celou obrazovku a obsahuje pouze spodní navigaci a zobrazovací pole, které se nahrazuje příslušným fragmentem. Při inicializaci tohoto pohledu je jako první fragment načten fragment uživatelského profilu, aby se uživatel mohl rychle zorientovat a nebyly mu hned nabízeny skupiny.

Propojení rozhraní a aplikační logiky

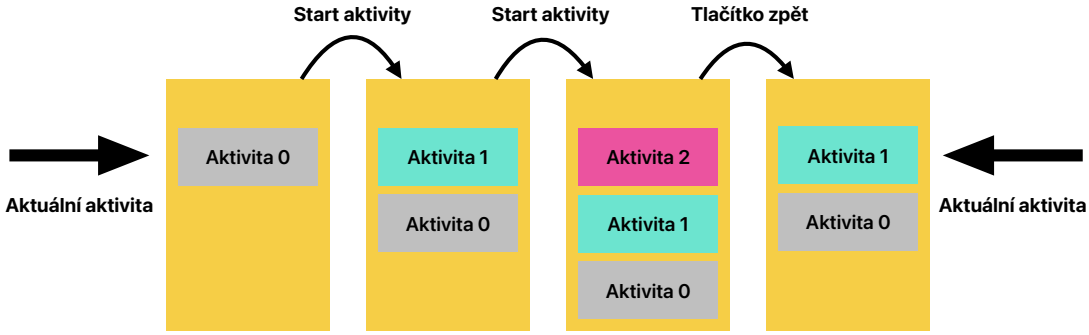
Propojení mezi uživatelským rozhraním (angl. *frontend*) a aplikační logikou (angl. *backend*) je realizováno pomocí hledání identifikátorů, který jednoznačně popisuje danou komponentu v aplikaci pomocí dvojice: pohled (angl. *view*) a identifikátor komponenty (element ID) a spojuje tak vrstvu uživatelského rozhraní a logiky. Je nutné, aby byl tento identifikátor jednoznačný v kontextu pohledu, ideálně však i v kontextu celé aplikace. Tato dvojice poté jednoznačně určuje element z tabulky komponent a lze mu tak následně přidělovat atributy přímo z kódu aplikační logiky. Toto propojení je nezbytné pro veškerou interaktivitu uživatelského prostředí, například při validaci uživatelských vstupů. Diagram reprezentující takové propojení je možno vidět na obrázku 4.7.



Obrázek 4.7: Diagramová vizualizace komunikace aplikační logiky s uživatelským rozhraním

Přepínání aktivit

Pro přepínání kontextu mezi aktivitami slouží tzv. záměr (angl. *Intent*) [7]. Přes tento objekt lze měnit momentální kontext aktivit pomocí volání funkce `startActivity(Intent)`. Jako parametr funkce je nutno vložit instanci aktivity, kterou lze inicializovat pomocí záměru a momentálního kontextu. Při zavolání funkce `startActivity(...)` je spuštěna nová aktivita do popředí a „překryje“ tak původní aktivity, avšak tuto aktivitu nepozastaví. Překrytá původní aktivita může stále vykonávat svoji logiku – například naslouchat na změny v databázi Firebase. Nově spuštěná aktivita je zařazena do tzv. zásobníku aktivit (angl. *Activity Stack*, jeho reprezentaci lze vidět na obrázku 4.8), který jasně definuje spuštěné aktivity a jejich pořadí. K původní aktivitě se lze vrátit pomocí zavolání funkce `finish()`, která ukončí momentálně prováděnou aktivitu a odstraní ji ze zásobníku aktivit – pokud je aktivita v popředí, zruší ji, pokud je hlouběji v zásobníku aktivit, jednoduše ji vymaže a zásobník posune. Místo funkce `finish()` lze také implicitně použít systémové tlačítko zpět, které volá v základním nastavení stejnou funkci. Funkcionalitu tohoto tlačítka lze přenastavit v kódu a změnit tak jeho chování. V mém případě však takové skutečnosti není potřeba a každá část aplikace počítá s jeho funkčností a využívá ji. Pokud jsem v kódu chtěl začít novou aktivitu a zároveň ukončit předcházející aktivitu (například po úspěšném přihlášení a následném přesměrování na profil uživatele) jsem zavolal novou aktivitu a následně ihned předcházející (*login* – přihlašovací) aktivitu ukončil pomocí funkce `finish()`.



Obrázek 4.8: Vizualizace zásobníku aktivit při jejich průběžném přidávání a ukončování

Případné zachycení, kdy aktivita již není v popředí či situace, kdy se aktivita do popředí vrátí se mohou zachytit pomocí zpětných volání `onPause()` a `onResume()`. V mé implementaci je využíván převážně zpětné volání `onResume()`, kdy se kontroluje, zda je uživatel stále přihlášen v aplikaci a pokud není, je navrácen na přihlašovací pohled, kde se musí opětovně přihlásit pro pokračování průchodu aplikací.

Při práci s objektem `Intent` lze také mezi aktivitami předávat argumenty, které lze následně využívat v nově inicializované aktivitě. Funkce pro takovéto předávání parametrů jsou například `Intent.putExtra(<název>, <hodnota>)`. Lze tak předávat základní datové typy (`Int`, `Char`, `String`, apod.) anebo například serializovaný objekt. V mém případě jsem využíval převážně předávání hodnot datového typu `String` a `Long`. `String` pro předávání klíčů uživatele a skupin, `Long` převážně pro předávání identifikátorů filmů, apod. Tyto argumenty se musí přidat do záměru ještě před zavoláním samotné funkce `startActivity()`. Přistupovat k nim poté lze pomocí funkce `getStringExtra(<název>)` pro datový typ řetězce znaků nebo `getIntExtra(<název>)` pro datový typy reprezentující celé číslo. Analogicky lze pak získat i další podporované datové typy. Při předávání je nutno zajistit, aby názvy položek byly unikátní a šlo k nim tak jednoznačně přistupovat. Všechny tyto hodnoty jsou do nové aktivity předány v balíčku (angl. *Bundle*) [7]. Pomocí výše uvedených funkcí pracujeme právě s uvedeným balíčkem.

Hodnoty v aplikaci dále alternativně předávám pomocí globálního objektu, který uchovává jak konstanty pro celou aplikaci (například konstanta určující pole obsahující všechny podporované žánry aplikace – toto pole se využívá při nabízení uživateli všech možných žánrů, které lze do skupiny přidat), tak i proměnné, které lze dynamicky měnit za běhu aplikace. Toto řešení předávání parametrů není ideální a je využíváno pouze v krajních případech (například když více pohledů využívá stejnou hodnotu). Tato metoda je náchylná k neoprávněnému nahlédnutí z důvodu, že se nejedná o privátní hodnoty. Touto metodou tedy nejsou posílány citlivé informace jako jsou hesla, klíče a podobné citlivé či chráněné údaje.

Přepínání fragmentů

Jak bylo výše již zmíněno, přepínání kontextu fragmentů zajišťuje `Fragment Manager`. S tímto objektem lze komunikovat a měnit tak momentální stav fragmentu, který by se měl zobrazit. Fragmenty lze jak nahrazovat, tak i přidávat do zásobníku fragmentů (angl. *Fragment Stack*) obdobně, jako tomu bylo u aktivit.

V případě, kdy je potřeba změnit fragment, musí se provést transakce. Tato transakce se volá nad `parentFragmentManager`, což je správce fragmentů nadřazeného fragmentu (protože nadřazený fragment je ten, který je potřeba změnit).

Dále je zapotřebí stanovit, který fragment má být inicializován. Následně musíme stanovit, zda chceme nynější fragment nahradit, či překrýt novým. Pokud chceme nahradit stávající fragment, voláme nad transakcích funkci `replace(fragment)`, jinak voláme funkci `addToBackStack()`. V případě, kdy chceme tento nový fragment zavřít, stačí zavolat funkci `popBackStack()`, která tento fragment vyřadí ze zásobníku fragmentů a nahradí ho posledním fragmentem na jeho vrcholu.

Obdobně jako u aktivit funguje i způsob vkládání fragmentů „na sebe“. Při vložení nového fragmentu je starý fragment překryt novým. U těchto fragmentů je tedy důležité, aby měly nastaveny pozadí, jinak hrozí, že fragment bude prosvítat do starého fragmentu. Zároveň je nutné nastavit, aby nový fragment byl interaktivní, jinak by uživatel manipuloval s fragmentem v pozadí namísto nového fragmentu. Oba tyto atributy lze nastavit přímo v implementaci GUI v XML souboru tohoto fragmentu.

Předávání dat mezi fragmenty jsem implementoval narodil od aktivit, kdy je možno předávat data pomocí záměru tak, že jsem předával data pomocí tzv. doprovodného objektu (angl. *companion object*), který obsahuje funkci `newInstance(<args>)`, která může vytvořit novou instanci fragmentu s danými parametry, při přepnutí na fragment skupiny je takto předán kód skupiny pro jednoznačné načtení právě této skupiny z databáze.

Čištění po přepnutí aktivit/fragmentů

Po pozastavení funkčnosti fragmentů či aktivit je nutno vyčistit (pozastavit) možné prostředky, které tato třída může obsahovat. Převážně se takto čistí posluchače databáze, aby se zbytečně neaktualizovaly pohledy, které se nevyužívaly a optimalizovalo se tak využití a zátěž databáze.

Prezentace skupin uživateli

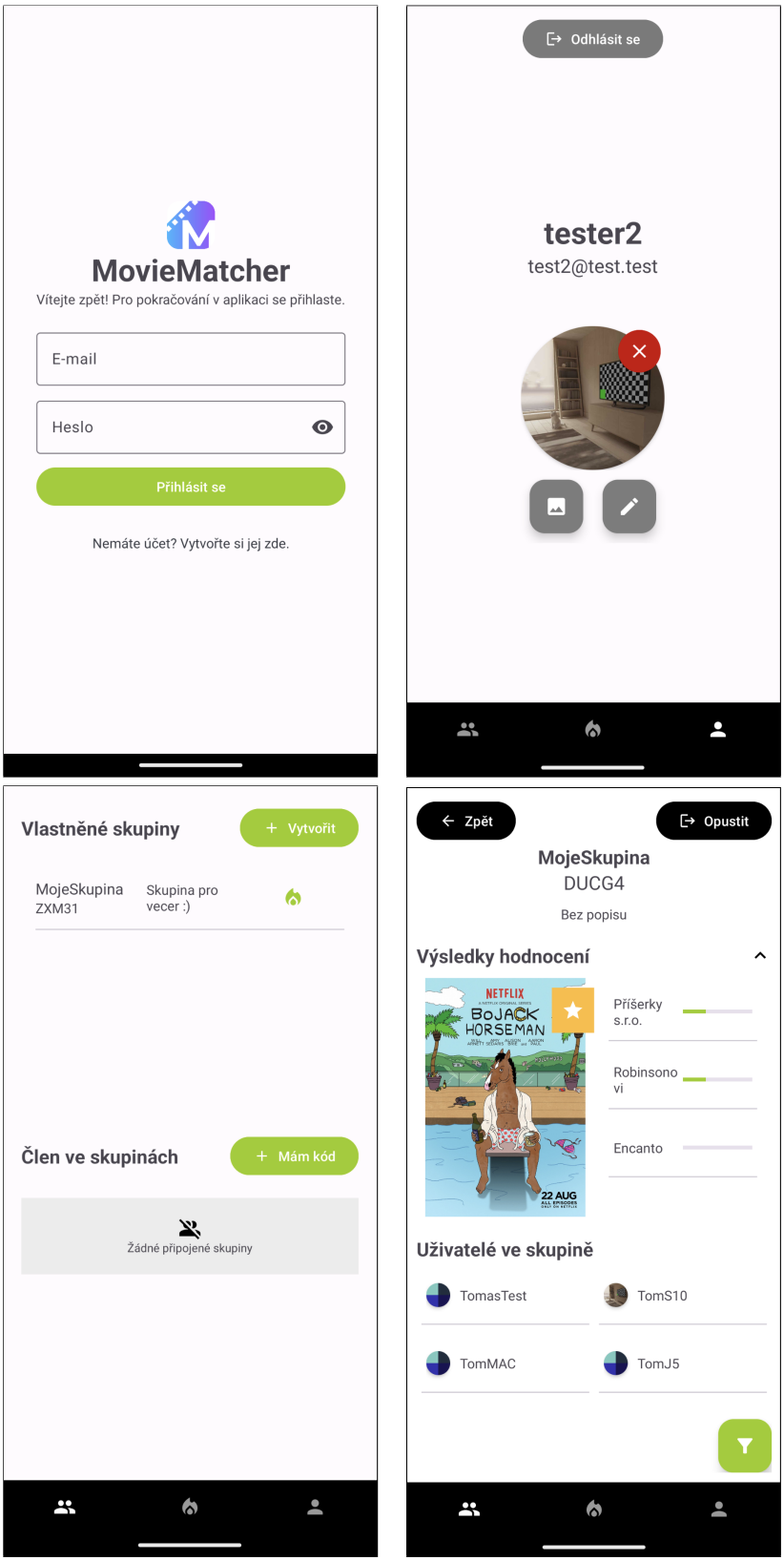
Aby uživatel aplikace mohl co nejsnadněji a nejpřehledněji získat informace o skupinách, ve kterých je připojen, či spravovat své vlastní skupiny, rozhodl jsem se při samotné implementaci nabízet seznam skupin uživateli ve dvou různých pohledech. Prvním pohledem je seznam všech vlastněných/připojených skupin. Tento seznam je určen pro kompletní přehled o všech skupinách. Uživatel zde má možnost zároveň vytvořit skupinu nebo se do již existující skupiny připojit.

Pro rychlý přístup ke skupinám, které jsou momentálně ve stavu hodnocení má uživatel k dispozici zvláštní pohled, který mu nabízí pouze tyto skupiny.

Výsledný vzhled aplikace

Celkový výsledný vzhled aplikace velmi dobře popisují následující snímky obrazovky pořízené přímo z běhu aplikace.

Na horním snímku obrazovky z obrázku 4.9 vlevo lze vidět přihlašovací pohled, na kterém se uživatel musí přihlásit (popřípadě registrovat), aby mu byl umožněn vstup do aplikace. Vpravo se nachází pohled pro správu uživatelského profilu. Tento pohled slouží uživateli pro změnu profilového obrázku (levé tlačítko pod profilovým obrázkem), kdy tento obrázek lze popřípadě smazat příslušným červeným tlačítkem s křížkem, a změnu uživatelského jména, pro kterou slouží pravé tlačítko pod profilovým obrázkem.



Obrázek 4.9: Snímky obrazovky demonstrující pohled pro přihlášení, profil uživatele, pohled vlastněných a připojených skupin a detail skupiny (zleva doprava)

Na spodním snímku obrazovky z obrázku 4.9 vlevo je možno vidět pohled, který uživateli prezentuje skupiny, které uživatel vlastní a ve kterých je připojen. V případě, že uživatel momentálně není připojen v žádné skupině, je mu tato informace sdělena pomocí karty, kterou lze vidět ve spodní polovině tohoto obrázku. V případě, že uživatel žádné skupiny nevlastní, je tato karta zobrazena i v sekci vlastněných skupin. Více o prezentaci skupin uživateli je popsáno v kapitole 4.5. Skupiny jsou zde zobrazovány jako seznam karet, kdy na každé kartě lze najít název skupiny, kód pro připojení a její popis. V případě, že skupina je momentálně ve stavu hodnocení, je tato informace zobrazena v podobě ikony, kterou lze najít v pravé části karty dané skupiny. Posledním obrázkem v této kolekci snímků obrazovky je pohled, který vyobrazuje detail skupiny, do které je uživatel přihlášen. Tento pohled zobrazuje mimo základní údaje o skupině (název, kód, popis) i informace o výsledcích posledního hodnocení skupiny, proběhlo-li již nějaké. V případě, že ve skupině hodnocení ještě neproběhlo, tato informace zobrazena není a objeví se až po prvním ukončeném hodnocení skupiny. Dále je zde zobrazen seznam uživatelů, kteří jsou ve skupině připojeni – konkrétně jejich profilový obrázek a uživatelské jméno. Při kliknutí na profilový obrázek jej lze zvětšit. Jako poslední element je na tomto pohledu k disposici plovoucí akční tlačítko, které při interakci zobrazí filtry skupiny, které lze upravovat. Pohled prezentující filtry skupiny lze vidět na obrázku 4.10.

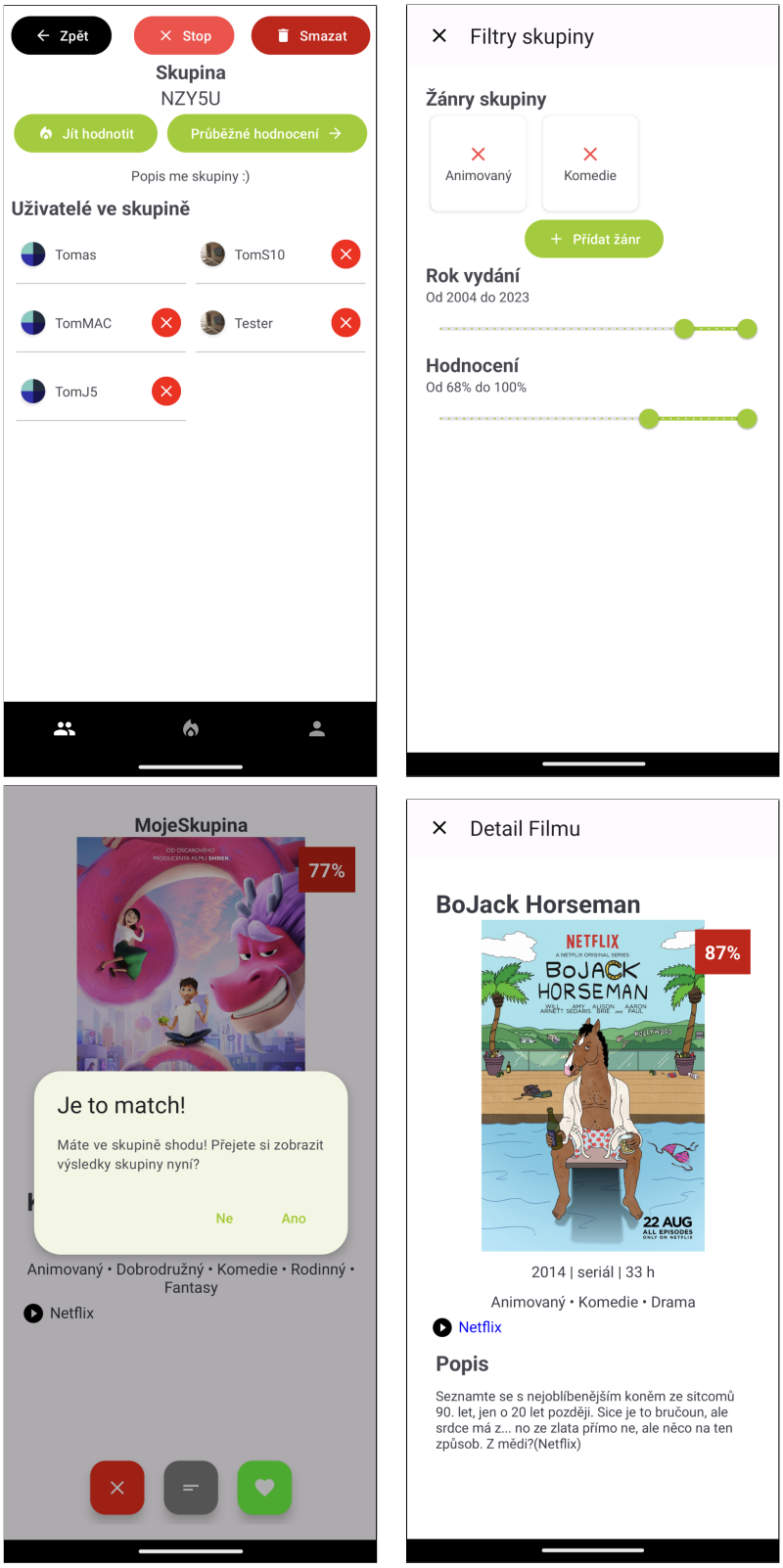
Na horním snímku z obrázku 4.10 vlevo lze vidět pohled detailu skupiny, který se zobrazuje pouze majiteli skupiny, značně se tak liší od pohledu člena skupiny. Zobrazuje více možností, mezi které patří možnost začít či ukončit proces hodnocení filmů pomocí tlačítka, které lze vidět nahoře uprostřed v tomto pohledu. Toto tlačítko se dynamicky mění na základě stavu hodnocení skupiny. Dále má majitel skupiny možnost tuto skupinu smazat[6] či odstranit některého z jejich účastníků. Vzhledem k tomu, že je zobrazená skupina ve stavu hodnocení jsou také zobrazena tlačítka, kterými se lze dostat do hodnotícího pohledu, či do pohledu, který zobrazuje průběžná hodnocení všech filmů ve skupině (tyto tlačítka jsou zobrazeny jak účastníkovi, tak majiteli skupiny).

Na pravém horním obrázku jsou vyobrazeny filtry skupiny, které lze měnit a ovlivňovat tak filmy, které jsou jejím uživatelům nabízeny. Lze zde přidávat či odebírat žánry, upravovat rok vydání či hodnocení snímku na internetu.

Ve spodní řadě vpravo lze pak vidět snímek, který zobrazuje obrazovku pro hodnocení filmů. Jsou zde udány základní údaje o daném snímku, pomocí spodního červeného a zeleného tlačítka lze snímek hodnotit. Pomocí šedého tlačítka mezi nimi lze přepínat mezi obrázkem filmu a jeho popisem.

Na pravém spodním obrázku lze vidět detail filmu, který si uživatel může zobrazit buďto z pohledu průběžného hodnocení filmů, či z výsledků posledního hodnocení filmu, které lze vidět na obrázku 4.9. Mimo informace o filmu je zde možnost po kliknutí na modře zvýrazněnou VoD službu tuto službu s daným filmem ihned otevřít ve webovém prohlížeči.

[6]Skupinu lze smazat pouze v případě, že v ní je připojen pouze majitel, je tedy nutno nejdříve odstranit všechny její účastníky



Obrázek 4.10: Snímky obrazovky popisující pohled detailu skupiny, pohled pro nastavení jejích filtrů, pohledy prezentující shodu a detail filmu (zleva doprava)

Kapitola 5

Optimalizace a testování

Jeden z posledních kroků při vývoji samotné aplikace bylo zjistit, zda je plně funkční a zda splňuje potřeby uživatele, které jsou popsány v kapitole 3.2. Zároveň je tak důležité aplikaci, pokud to jde, optimalizovat, aby fungovala efektivně na různých zařízeních.

5.1 Optimalizace

Přestože aplikace funguje jak by měla, je dobré, aby byla co nejvíce optimalizovaná. Tato skutečnost je takřka nezbytná vzhledem k tomu, že aplikace by měla ve výsledku být nachystaná na publikování a představení veřejnosti. Chceme tedy pokrýt co největší spektrum zařízení a může se stát, že se aplikace dostane na zařízení, které disponuje horším výkonem než očekáváme nebo bude používána v hraniční situaci (např. velmi slabé připojení k síti). Snažil jsem se tedy, aby kód aplikace byl co nejefektivnější a výsledná prezentace aplikace co nejpřívětivější koncovému uživateli.

UI optimalizace

Po implementaci aplikace a patřičném testování jsem provedl četné optimalizační kroky především pro ulehčení načítání UI (více o uživatelském rozhraní v kapitole 2.5) například při změně dat v aplikační logice a následné nutnosti znovu-načtení pohledu. Kupříkladu lze tuto optimalizaci demonstrovat na pohledu, kdy uživatel zobrazuje detail skupiny. V tomto případě je při inicializaci pohledu nastaven také naslouchač (viz kapitola 4.2), který naslouchá na změny a při změně nějakého aspektu skupiny celý pohled znovu načte. Z hlediska funkčnosti splňuje mé požadavky, načte správná data, zhodnotí je a na základě výsledku pohled patřičně upraví. Problém však nastává v okamžiku, kdy jsou do pohledu zavedeny listy prvků (`Recycler View` komponenta). Tato komponenta způsobuje úkaz, kdy při každém znovu-načtení skupiny tyto listy viditelně „probliknou“, a to i při změně banálních prvků skupiny, které nemají nic společného s těmito pohledy, jako například název skupiny, popis skupiny, atd. Pro řešení tohoto problému jsem provedl dekompozici programové logiky pro načítání dat do skupiny na funkční celky, které jsou vzájemně nezávislé, tyto celky se poté načítají postupně pouze v případě, kdy se nějaký z prvků dané skupiny změnil. Příklad dekompozice zobrazovaných prvků je seznam uživatelů, tlačítka pro interakci se skupinou, filtry skupiny.

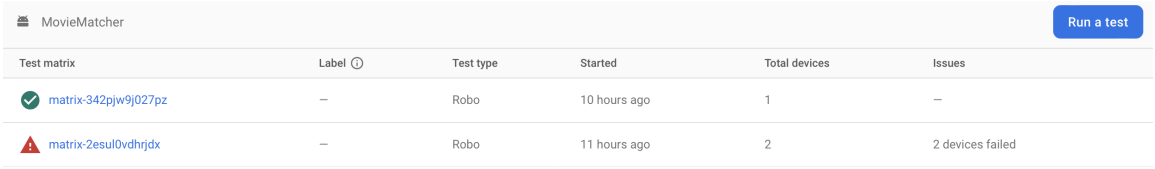
5.2 Testování

Jako hlavní testovací nástroj jsem během tvorby projektu využíval emulátor Android zařízení implementovaný v Android Studiu. Díky faktu, že se jednalo o emulátor zařízení jsem měl k dispozici mimo samotné testovací zařízení i výpisový terminál – *Logcat*, ve které jsem měl k dispozici průběžné výpisy ze zařízení. Mohl jsem takto identifikovat problémy, které nebyly očividné z chování samotného zařízení (například přetečení paměti, alokace zbytečných bloků dat, apod.). Do konzole jsem zároveň vypisoval potřebné průběžné informace o stavu aplikace.

Pro testování vlastně implementovaných funkcí jsem průběžně také tvořil testovací skripty, které danou část kódu otestovaly před sestavením programu přímo v editoru.

Testování implementace

Pro testování implementace aplikace jsem využil mimo jiných výše zmíněných postupů také službu **Test Lab** z platformy Firebase, která zaštituje většinu databázových operací v aplikaci. Služba Test Lab poskytuje možnosti automatizovaného testování implementace aplikace na více zařízeních zároveň. Aplikaci lze testovat jak na Android, tak iOS zařízeních. V mém případě však využívám pouze možností testování na Android platformě. V rámci testování je k dispozici také, mimo základní informace o běhu aplikace (např. zda aplikace prošla testem bez pádu), velká řada doplňujících informací jak o právě testovaných stavech, tak o celkovém dojmu z aplikace – nejde tedy pouze o povrchní testování daných případů, jde i o testování UX a UI aplikace.



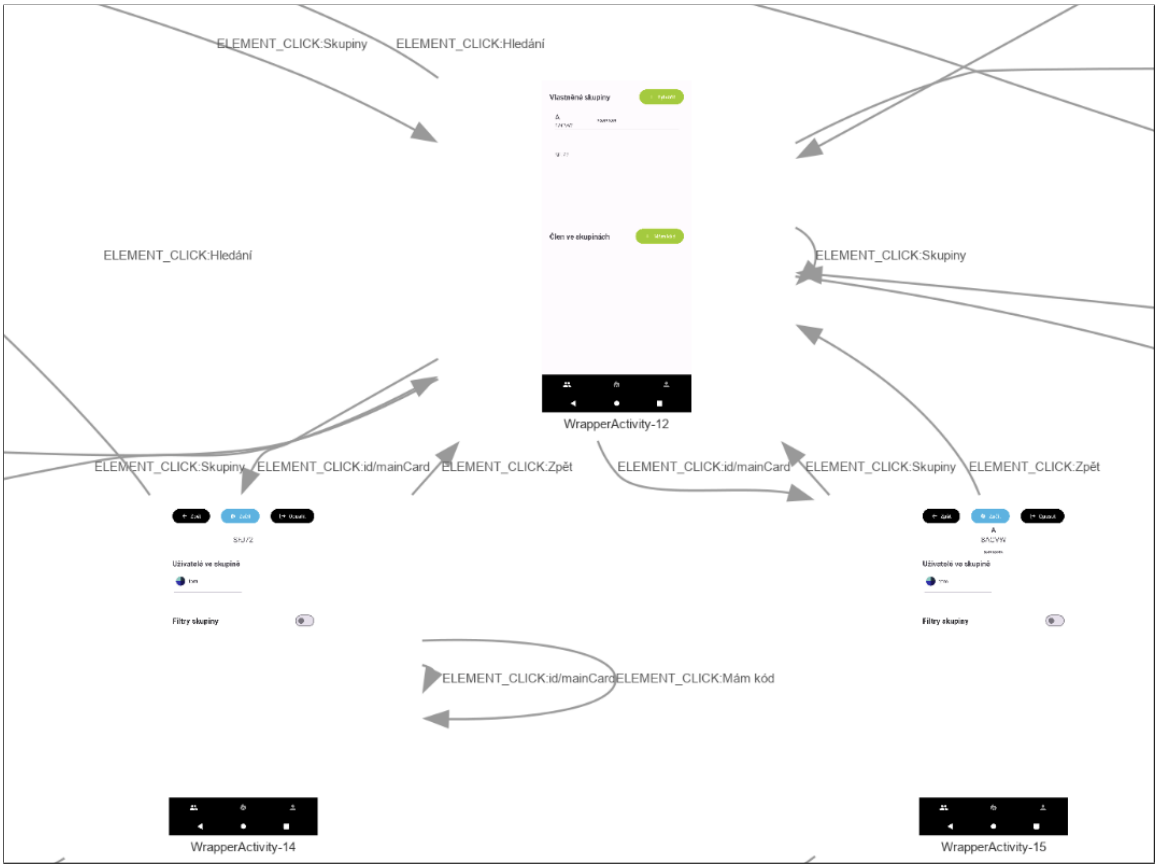
Obrázek 5.1: Výsledky dvou robotestů

Před zahájením testování lze poskytnout vlastní *Robo Skript*, který udává, jaké akce by se měly v aplikaci na zařízení provézt. Jedná se o soubor ve formátu JSON, který obsahuje záchytné body pro orientaci a řízení testů. Jako takovýto skript lze použít předdefinovaný soubor, nebo lze přímo v Android Studiu nahrát svůj vlastní skript. Proces nahrávání vlastního skriptu spočívá v zapnutí aplikace s programem, který zachytává události v aplikaci. Po spuštění tohoto emulátoru se lze pohybovat aplikací, interagovat s tlačítky, vyplňovat textové vstupy a přepínat kontexty. Každá tato akce je zachycena a po následném ukončení aplikace převedena do výše zmíněného JSON souboru, který lze poté při nastavování testů nahrát jako vlastní skript. Výsledky dvou takovýchto testů lze vidět na obrázku 5.1.

V případě využití autentizace v aplikaci lze navést testovací skript i na přihlašovací formulář, konkrétně na pole uživatelské jméno a heslo, které se k autentizaci využívají.

Další možnost použití je namísto předdefinovaného průvodu aplikací tento skript neposkytovat, tím se provede test v režimu *deep crawler*. V tomto režimu testovací aplikace po spuštění začne procházet všechny možnosti, kterými může uživatel s aplikací interagovat. Aplikace automaticky testuje všechny případy, které v aplikaci mohou nastat. Tento průchod aplikací je poté ukončen po dosažení časového limitu, který byl v mém případě 5–10 minut.

Po průchodu testem dostaneme výstupní zprávu o průběhu zároveň se snímky obrazovky přímo z aplikace, které ukazují, kterými částmi aplikace test procházel. Zároveň je tak k dispozici, video, které zachytává celý průběh testu a *crawl graf*, což je graf, který jednoznačně udává, kudy v aplikaci test postupoval s informacemi o interakcích s prvky aplikace. Obsahuje snímky obrazovky z aplikace, které jsou propojeny přechody označenými názvem prvku, se kterým proběhla interakce pro dosažení dalšího snímku. Výřez z tohoto grafu (zhruba 1/10 výsledného obsahu samotného grafu) lze vidět na obrázku 5.2.



Obrázek 5.2: Výřez z crawl grafu poskytnutého službou Firebase Test Lab

Lokální testování databáze

Před nasazením aplikace na hostovanou Firebase je dobré otestovat funkcionalitu komunikace s databází a správný přístup k datům lokálně, převážně z důvodu, aby se zbytečně nezatěžovala reálná databáze při případných požadavcích/nahrávání velkých dat způsobených například možným zacyklením v programu. Zátěž databáze je pro testovací účely kritická z důvodu omezení způsobené měsíčními limity na hostovanou Firebase firmou Google.

Pro lokální testování služby Firebase slouží rozhraní `Firebase-CLI`, které poskytuje takřka totožné služby jako tomu je u hostované verze. Po spuštění služby přes terminál se nejprve provede proces nastavení/stáhnutí všech požadovaných služeb (v mém případě Realtime Database a Firebase Authentication), poté služba spustí patřičné služby na daných portech pro přístup z lokální sítě. V tento moment lze s lokální Firebase komunikovat stejně tak, jako s hostovanou verzí. Pro přístup se využívá HTTP odkaz, který obsahuje IP adresu lokální sítě (v případě mobilních zařízení adresa `10.0.2.2`), port služby, kterou chce program

využívat (`8080` — Firebase Authentication, `9000` — Firebase Realtime Database) a následně název projektu, ke kterému je přistupováno, tedy `http://10.0.2.2:8080/cinematchingapp` pro autentizaci projektu „cinematchingapp“ – názvu mého projektu.

Firebase-CLI zároveň poskytuje webové rozhraní, které je velmi podobné s webovým rozhraním hostované Firebase. Lze tak jednoduše a intuitivně komunikovat s databází, importovat JSON soubory a jednoduše tak populovat databázi, spravovat uživatele a sledovat změny v databázi v reálném čase. Zároveň tak lze jednoduše spravovat autorizované uživatele.

Uživatelské testování

Jako jeden z posledních kroků při testování aplikace je nutno provést testování aplikace na reálných uživatelích. Přestože během testování prošly robotesty i přes to, že aplikace splňovala svoji funkcionalitu v testech z kapitoly 5.2, nemusí aplikace splňovat účely veřejnosti, popřípadě může být implementovaná odlišným způsobem, než uživatel očekává.

Zároveň je nutno provádět tyto testy ve větší skupině uživatelů. Během testů byla aplikace totiž testována na 1-4 zařízeních – skupina, která byla kombinací emulátorů v Android Studiu a fyzických Android zařízení. V případě uživatelského testování byly testy prováděny na skupině 6 uživatelů, kdy každý uživatel využíval své vlastní fyzické zařízeni.

Uživatelské testování probíhalo v jedné místnosti s konstantním připojením k internetu. Samotné kroky, ve kterých testování se skupinou uživatelů probíhalo byly následující:

1. **Nastavení atmosféry**, ve které se testy mají odehrávat. Všem členům testovací skupiny jsem vysvětlil, jakým směrem se mají testy ubírat. Toto zasazení skupiny do kontextu probíhalo slovně. Vysvětlil jsem skupině, že jsou v roli skupiny přátel, která má za úkol „dohodnout se na jednom snímku, který si pustí večer na společné akci“. Při výběru uživatelé mají samozřejmě k dispozici testovanou aplikaci. Uživatelé byli také poučeni, že by měli co nejméně komunikovat, aby nejlépe využily hlavní vlastnosti aplikace, a to tu, že poskytuje uživatelům možnost dohodnout se na filmu téměř bez jakékoliv komunikace.

2. **Pozorování průběhu** testu probíhalo tím způsobem, že jsem pozorované subjekty nechal provádět daný test a postupně jsem si poznačoval poznatky získané tímto testováním. Zároveň jsem zkoumal, jak snadno se uživatelům aplikace používá, zda nemají problémy s přístupností, popřípadě navigací v aplikaci, obdobně jako u testování makety aplikace v kapitole 3.4. Někteří uživatelé se rychle zaregistrovali a začali upravovat filtry skupiny, zatímco zbylá část skupiny pečlivě vybírala uživatelské jméno, heslo a profilovou fotku.

3. **Vyhodnocení** testu probíhalo způsobem, kdy jsem agregoval všechny poznatky z testování a vyhodnotil z nich závěr. Celý test trval uživatelům přibližně 20 minut s tím, že se nejprve museli zaregistrovat, poté sestavit skupinu o všech zúčastněných členech, nastavit společné filtry a následně projít seznam doporučených filmů, dokud nedosáhli společné shody. V průběhu testování jsem narazil na tyto hlavní poznatky:

    - Uživatelé hned u registrace ocenili interaktivitu vstupních textových polí.
    - Uživatelé se pohotově orientovali v rozhraní aplikace.
    - Testeři několikrát využívali možnost upravení popisu a názvu skupin. Tato implementovaná funkce ušetřila značné množství času, které by jinak bylo plýtváno na opakované mazání a tvoření skupin.

Zpětná vazba testerů

Po dokončení testu jsem zároveň požádal zúčastněné o poskytnutí zpětné vazby k aplikaci z důvodu získání informací o tom, zda je aplikace přívětivá i z pohledu uživatele a působí na něj pozitivním dojmem – je totiž možné, že pohled uživatele aplikace (v mém případě testerů) se může lišit od pohledu pozorovatele. Získání této zpětné vazby probíhalo formou krátkých rozhovorů se zúčastněnými testery aplikace.

Zpětná vazba ukázala, že největším záporem aplikace je absence vlastní grafiky, popřípadě ilustrací a pozadí – zkrátka aplikace v nynějším stavu působí velmi sterilně a neosobně. Tento fakt nemá na jednoduchost používání ani funkčnosti velký vliv, ale je nutno s ním pracovat v pozdějších iteracích aplikace, viz. kapitola 5.3. Část testerů aplikace by zároveň ocenila, kdyby byla při shodě ve skupině o této skutečnosti informována například pomocí notifikace, která by se spustila pokaždé, když by taková událost nastala.

5.3 Možná vylepšení

Ačkoliv je aplikace ve stavu, kdy by mohla být považována za hotovou a připravenou k jejímu vydání, časem by bylo dobré aplikaci udržovat a postupně vylepšovat – mluvíme tak o tzv. „verzi 2“. Verze 2 je teoretická verze, která zahrnuje vylepšenou funkcionalitu, popřípadě vyřešené stávající problémy.

Design

Jak bylo v kapitole 5.2 zmíněno, jeden z hlavních nedostatků aplikace spočívá v nedostatečně osobním a poutavém designu. Tento fakt lze napravit vylepšením stávajícího rozhraní o další grafické prvky, popřípadě upravit stávající grafické prvky tak, aby poutaly více pozornosti.

Funkcionalita

Co se týče funkcionality následující teoretické verze aplikace, měla by disponovat hlavní vlastností, jejíž absence byla vytknuta ve zpětné vazbě testerů z kapitoly 5.2, a to notifikací při výskytu vzájemné shody v nějaké ze skupin, ve kterých je uživatel připojen. Uživatelé by zde měli také možnost notifikace pro dílčí skupinu buďto zapnout či vypnout a vyhnout se tak nevyžádanému zasílání notifikací, které je podle většiny uživatelů nevhodné.

Kapitola 6

Závěr

Hlavní premise aplikace byla usnadnit uživatelům vybrat ideální film pro společné shlédnutí a ušetřit tak čas strávený zbytečným domlouváním a vybíráním snímků. Aplikace tento účel splňuje a jak bylo zmíněno i ve zpětné vazbě testerů, opravdu je uživatelům v tomto směru nápomocná.

Při tvorbě aplikace byly taktéž využívány moderní technologie pro tvorbu mobilních aplikací a zároveň byly využity správné metodiky pro samotnou tvorbu aplikace z hlediska stálé komunikace s uživatelem, která byla udržována od samotného návrhu kostry aplikace (kapitola 3.4) až po finální uživatelské testování výsledné aplikace (kapitola 5.2). Uživatelské zpětné vazby u každé části se odrážely na postupném dalším vývoji tak, aby vyřešily co nejvíce uživatelských nesrovnalostí/poznámek. Taktéž jsem při samotném vývoji zabředl do většiny odvětví takového procesu, ať už je to návrh, samotné programování aplikace, tak i design (kupříkladu návrh loga aplikace).

Aplikace také ctí a odráží trendy moderního designu mobilních aplikací jak v oblasti rozpoložení, tak v oblasti stylu samotných prvků. Tímto se aplikace stává uživatelsky více přívětivá a můžeme ji považovat za nachystanou pro komerční užívání veřejností z důvodu, že uživatel se v aplikaci dokáže pohotově orientovat a bezproblémově ji tedy naplno využívat, pokud má uživatel již nějaké předchozí zkušenosti s podobnými moderními aplikacemi.

Literatura

[1] Arshad, W. *Managing State in Flutter Pragmatically*. Packt Publishing Ltd., 2021. ISBN 978-1-80107-077-5.

[2] Cloudflare. *What is BaaS? | Backend-as-a-Service vs. serverless* [online]. Cloudflare [cit. 2023-13-04]. Dostupné z: https://www.cloudflare.com/en-gb/learning/serverless/glossary/backend-as-a-service-baas/.

[3] Cohen, R. a Wang, T. *Beginner's Guide to Kotlin Programming*. Apress Berkeley, 2014. ISBN 978-1-4842-0383-5.

[4] Google. *Material Design 3* [online]. 2013 [cit. 2023-04-02]. Dostupné z: https://m3.material.io.

[5] Hartson, R. a Pyla, P. S. *UX Book - Process and Guidelines for Ensuring a Quality User Experience*. Elsevier, 2012. ISBN 9780123852410.

[6] Hunt, J. *Beginner's Guide to Kotlin Programming*. Springer Cham, 2021. ISBN 978-3-030-80892-1.

[7] Lacko Luboslav. *Vývoj aplikací pro Android*. Albatros Media, a.s., 2015. ISBN 978-80-251-4347-6.

[8] Mawlood Yunis, A.-R. *Android for Java Programmers*. Springer Cham, 2022. ISBN 978-3-030-87459-9.

[9] McAllister, W. a Fritz, S. J. *Programming Fundamentals Using Java: A Game Application Approach*. David Pallai, 2015. ISBN 978-1-938549-76-2.

[10] StatCounter. *Mobile Operating System Market Share Worldwide* [online]. StatCounter, duben 2023 [cit. 2023-04-02]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[11] Ujbányai, M. *Programujeme pro Android*. Grada Publishing, a.s., 2012. ISBN 978-80-247-3995-3.

Příloha A

Obsah přiloženého paměťového média

Součástí výsledné práce je také přiložená SD karta obsahující následující adresáře:

- **src/** – adresář obsahující zdrojové soubory výsledné aplikace
- **doc/** – adresář obsahující vygenerovanou dokumentaci k aplikaci
- **bin/** – adresář obsahující zkompilovaný instalační balíček aplikace pro platformu Android
- **tex/** – adresář obsahující zdrojové soubory technické zprávy
- **BP.pdf** – text technické zprávy ve formátu PDF
- **BP.mp4** – video demonstrující finální aplikaci s popisem její funkcionality
- **README.md** – soubor obsahující manuál pro instalaci a spuštění