SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-98014

MOVIE TINDER

BAKALÁRSKA PRÁCA

2021 **Adam Trebichalský**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-98014

MOVIE TINDER

BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Názov študijného odboru:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Romana Jamrichová

Bratislava 2021 **Adam Trebichalský**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Adam Trebichalský
Bakalárska práca:	Movie tinder
Vedúci záverečnej práce:	Ing. Romana Jamrichová
Miesto a rok predloženia práce:	Bratislava 2021

Bakalárska práca je zameraná na vytvorenie mobilnej aplikácie, ktorej úlohou je dvom používateľom navrhovať spoločné filmové a seriálové tituly, ktoré by ich oboch mohli zaujať. Práca zahŕňa naštudovanie problematiky a potrebných technológií, analýzu problému, jeho návrh a nakoniec implementáciu a otestovanie mobilnej aplikácie. Jadrom teoretickej časti práce je analýza metodík používaných pri odporúčacích systémoch a prehľad typov mobilných aplikácii. Spomenuli sme tak isto nami vybraný framework a technológie, ktoré sme použili na tvorbu samotnej aplikácie spolu s ich výhodami a odôvodnením, prečo sme si ich vybrali. Analytická časť práce obsahuje sumarizáciu existujúcich riešení, pomocou ktorej sme identifikovali potrebné komponenty a interakcie. Softvérový návrh finálneho riešenia je popísaný pomocou UML štandardu. Implementačná časť pozostáva zo zaznamenania postupu pri samotnom vývoji mobilnej aplikácie, pričom sú detailne popísané komplikovanejšie časti riešenia. Na záver sme aplikáciu otestovali, zbilancovali naše výsledky, zhodnotili riešenie a prínos aplikácie.

Kľúčové slová: mobilná aplikácia, odporúčacie systémy, filmy a seriály, react native

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Adam Trebichalský
Bachelor's thesis:	Movie tinder
Supervisor:	Ing. Romana Jamrichová
Place and year of submission:	Bratislava 2021

The bachelor thesis is focused on the creation of a mobile application, which main goal is to recommend to two users a movie or a TV series titles that could both interest them according to their tastes. The work includes also the study of technology, architectural design and implementation of a mobile application. The core of the theoretical part of the thesis consist of the analysis of methodologies used in recommendation systems, an overview of types of modern mobile applications and a description of our chosen framework and technologies for creating the application itself, justifying why we chose them and mentioning the benefits they provide. The design part of the work is devoted to the design of the application itself, which is showed with the help of different types of diagrams. The implementation part consist of showing the procedure in the development of the mobile application itself, while some relevant parts of the solution are listed in the form of images or parts of the code. At the end, we analyze the results, evaluate the solution and the benefits of the application.

Keywords: mobile application, recommendations systems, movies and tv series, react native

Poďakovanie

Obsah

Zoznam obrázkov a tabuliek

Zoznam algoritmov

Zoznam skratiek

UX User experience
UI User interface
NFC Near field communication
HTML HyperText markup language
CSS Cascading Style Sheets
API Application programming interface
OTA Over the air
OS Operation system
UML Unified modeling language
DOM Document object model
PCA -
SVD -

Zoznam výpisov

Úvod

Odporúčacie systémy sú v dnešnej dobe čoraz viac používané v rôznych odvetviach informatiky. Každý kto využíva internet sa s nimi už pravdepodobne stretol či už vedome, alebo nevedome. Najčastejšie sa s nimi bežný človek môže stretnúť pri používaní sociálnych sietí, pozeraní videí na YouTube, pozeraní filmov na streamovacích službách, hľadaní známosti na Tinderi, či nakupovaní na Amazone. Pri všetkých spomenutých službách a ich odporúčacích systémoch ide v širokej podstate o jeden a ten istý cieľ. Zúžiť celú svoju ponuku produktov na tie, o ktoré bude mať spotrebiteľ podľa systému najpravdepodobnejšie záujem.

Vzhľadom nato, že jednou z oblastí, kde sa odporúčacie systémy využívajú vo veľkej miere je aj filmový priemysel, cieľom tejto bakalárskej práce je vytvoriť mobilnú aplikáciu, ktorá bude využívať odporúčací systém nato, aby dvom používateľom na základe ich predchádzajúcich interakcií s filmami a seriálmi v databáze aplikácie, odporučila film, alebo seriál, ktorý bude pre oboch čo najviac relevantný. Používatelia si potom nezávisle od seba môžu v navrhovaných filmoch pomocou jednoduchého swipeovania (z angl. slova swipe) vyberať, či sa im daný film páči, alebo nie, pričom ak nastane medzi nimi zhoda a obaja označia ten istý film, aplikácia im ho zobrazí. Každý používateľ aplikácie bude mať svoju vlastnú filmotéku, teda zoznam filmov ktoré ohodnotil či už pozitívne, alebo negatívne. Aplikácia bude naprogramovaná cez populárny framework React Native.

Motiváciou k tvorbe tejto aplikácie je problém, s ktorým sa stretlo mnoho ľudí, či už pri nekonečnom výbere večerného programu v domácnosti, alebo neúspešnom výbere spoločného filmu medzi kamarátmi. Hlavný prínos aplikácie je najmä skrátenie času pri výbere spoločného filmu, pričom aplikácia nahradí neefektívne ručné prehladávanie databáz ako ČSFD a IMDb.

1 Teoretická časť

1.1 Odporúčacie systémy

Odporúčacie systémy predstavujú súbor softvérových nástrojov a techník, ktoré poskytujú používateľovi odporúčania ohľadom daného súboru položiek. Odporúčania súvisia s rôznymi rozhodovacími procesmi, ako napríklad akú položku kúpiť, akú hudbu počúvať alebo aké online noviny čítať. [1] Dalo by sa teda povedať, že ich hlavným cieľom je odporučiť danému používateľovi jemu relevantné položky. "Položka" je termín, ktorý označuje čo konkrétne daný odporúčací systém odporúča používateľom. Vzhľadom nato, že odporúčania sú väčšinou personalizované, rôzny používatelia dostávajú rozličné odporúčania. Existujú aj nepersonalizované odporúčania. Tie fungujú na jednoduchšom princípe a je ľahšie ich generovať. Typickým príkladom sú rebríčky, ktoré obsahujú výber top 10 najobľúbenjších položiek. Personalizované odporúčania v ich najjednouchšej forme sú zoradené zoznamy položiek.

Keď sa pozrieme na dôležitosť týchto systémov, v určitých odvetviach zohrávajú dôležitú rolu. Väčšina veľkých internetových služieb ako Amazon, YouTube, Netflix, Google, Tripadvisor či IMDb, majú vyvinuté svoje vlastné odporúčacie systémy, ktoré dlhodobo pomahajú zlepšovať ich výsledky, ale aj UX. Tak isto aj mnoho mediálnych firiem už dnes vyvíja odporúčacie systémy, ktoré potom nasadzujú a používajú ako časť služby, ktorú poskytujú svojim odoberateľom. Netflix v roku 2009 ocenil tím programátorov, ktorý ako prvý úspešne zefektívnil ich odporúčací systém o takmer 10%, cenou 1 milión dolárov. To len podčiarkuje ich významnosť a fakt, že svetové firmy do tejto technológie investujú nemalé finančné prostriedky.

1.1.1 Základné funkcie

Existuje viacero dôvodov, prečo môžu poskytovatelia služieb chcieť túto technológiu využiť:

- **Zvýšenie predaja** - najdôležitejšia funkcia pre komerčný odporúčací systém, je samozrejme zvýšenie počtu predaných položiek. Čím efektívnejší je, tým lepšie výsledky firme generuje.
- **Predávať rôznorodý tovar** - ďalšia významná funkcia je umožniť používatelovi vybrať si položky, ktoré by bez odporúčania ťažšie našiel.
- **Zvýšiť spokojnosť používateľa** - dobre fungujúci odporúčaci systém, vie zlepšiť aj celkový UX, čo v konečnom dôsledku zabezpečí, že používateľ bude odporúčania rád

využívať.

- **Zvýšiť vernosť používateľa** - akonáhle používateľ zaregistruje, na základe presných odporúčaní, že ho služba "pozná", nadobudne pocit, že ho považuje za hodnotného zakazníka a bude sa rád vracať.

- **Lepšie rozumieť tomu, čo zákazník chce** - pomocou zbierania rôznych používateľských preferencií a následného analyzovania, vedia firmy lepšie reagovať na trh a zmeny na ňom a tým prispôsobovať svoj sortiment potrebám trhu.

1.1.2 Základné pojmy

Všeobecná klasifikácia dát použitých v odporúčacích systémoch rozlišuje 3 druhy objektov.

- **Položky** - sú objekty ktoré sú odporúčané. Môžu byť charakterizované ich zložitosťou a ich hodnotou alebo užitočnosťou. Hodnota može byť kladná, ak je položka pre používateľa užitočná, alebo záporná, ak položka nie je vhodná a používateľ pri jej výbere urobil nesprávne rozhodnutie.

- **Používatelia** - môžu mať veľmi rozdielne ciele a vlastnosti. S cieľom personalizovať odporúčania a interakciu s počítačom, odporúčacie systémy využívajú množstvo informácií o používateľoch. Tieto informácie možu byť štrukturované viacerými spôsobmi a výber, ktoré informácie treba modelovať systémom, záleží od zvolenej techniky.

- **Transakcie** - všeobecne ich označujeme ako zaznamenanú interakciu medzi používateľom a odporúčacím systémom. Sú to dáta podobného typu ako logy, ktoré uchovávajú dôležité informácie vygenerované počas interakcie, ktoré sú užitočne pre odporúčací algoritmus.

Vo všeobecnosti, existujú odporúčacie techniky, ktoré nevyžadujú veľké množstvo vedomostí tj. používajú veľmi jednouché základné dáta, ako napríklad používateľove hodnotenia položiek. Najpopulárnejšou formou dát transakcií sú tak isto hodnotenia, ktoré systém zbiera. Tieto hodnotenia možu byť zbierané explicitne, alebo implicitne. Pri explicitnom hodnotení je používateľ vyzvaný, aby zdieľal svoj názor na danú položku. Hodnotenia môžu byť v rôznych formách, ako napríklad:

- numerické hodnotenie na pevne stanovenej stupnici (napr. 1-5),

- slovné hodnotenia ako veľmí súhlasím, súhlasím, neutrálny, nesúhlasím, veľmi neúhlasím,

- jednoduché binárne hodnotenie (dobré/zlé)

Na ilustráciu predikcie odporúčacieho systému, uvažujme jednoduchý, nepersonalizovaný odporúčací algoritmus, ktorý odporúča najobľúbenejšie piesne. Dôvodom pre použitie tohto prístupu je, že pri absencií presnejších informácií o preferenciách používateľa, populárna pieseň tj. položka, ktorá je obľubená mnohými používateľmi, bude pravdepodobne relevantná aj pre generického používateľa. Teda aspoň v porovnaní s náhodne vybratou piesňou. Preto sa predpokladá, že užitočnosť týchto populárnych piesní, bude pre tohto používateľa primerane vysoká.

Niektoré odporúčacie systémy nemusia nutne odhadovať užitočnosť položiek priamo pred ich odporúčaním, ale môžu namiesto toho použiť určitú formu heuristiky, aby vytvorili hypotézu, že položka je pre používateľa užitočná. Toto je typické napríklad pre systémy založené na znalostiach (knowledge-based systems). Tieto predikcie užitočnosti sú vypočítané špecifickými algoritmami a využívajú rozličné druhy vedomostí, ktoré systém má o používateľoch a položkách. Napríklad systém môže mať funkciu na výpočet užitočnosti, ktorá je boolovská tj. jednoducho iba určuje, či je položka pre používateľa užitočná, alebo nie.

1.1.3 Typy odporúčacích systémov

- **Demografické -** Tento typ systémov odporúča položky na základe demografického profilu používateľa. Sú pri nich generované rôzne odporúčania, pre rôzne demografické oblasti. Mnoho webových stránok používa na vytvorenie personalizácie jednoduché odporúčania pomocou demografikých údajov. Napríklad používatelia sú presmerovaný na konkrétne webové stránky na základe ich jazyka alebo krajiny. Prípadne sa z údajov často využíva aj vek používateľa.

- **Knowledge-based -** Systémy založené na znalostiach, odporúčajú položky na základe konkrétnych znalostí o tom, ako určité vlastnosti položiek zodpovedajú potrebám a preferenciám používateľov a v konečnom dôsledku o tom, ako je položka pre používateľa užitočná.

- **Komunitné -** Odporúča položky na základe preferencií užšej komunity používateľov označovanej ako priatelia daného používateľa. Táto technika sleduje epigram „Povedzte mi, kto sú vaši priatelia a ja vám poviem, kto ste“. Prieskumy naznačujú, že ľudia sa skôr spoliehajú na odporúčania svojich priateľov, ako na odporúčania od podobných, ale anonymných osôb. Toto pozorovanie v kombinácii s popularitou sociálnych sietí vytvára rastúci záujem o komunitné systémy označované aj ako sociálne odporúčacie

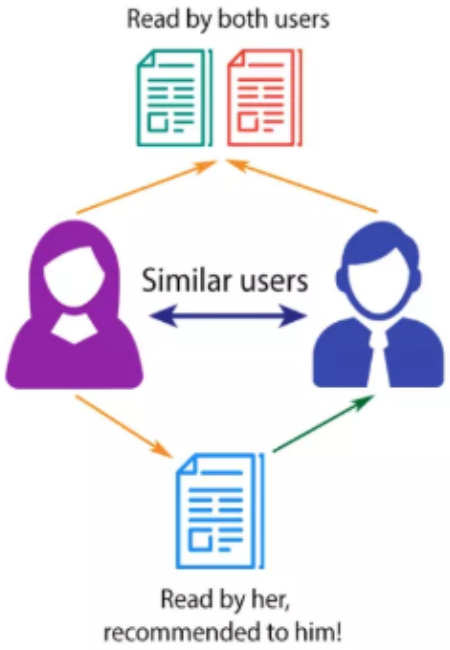
systémy.

1.1.4 Typy odporúčacích techník

Pri odporúčacích systémoch existujú dve hlavné techniky návrhu algoritmu a to tzv. collaborative filtering a content based filtering.

1.1.5 Collaborative filtering

Collaborative filtering je technika, ktorá vytvára odporúčania na základe zbierania preferencií od väčšieho počtu používateľov. Najjednouchšia a pôvodná implementácia tohto prístupu, odporúča používateľovi položky, ktoré v minulosti pozitívne ohodnotili používatelia s podobnými preferenciami (pozri obr. 1). Teda ak máme používateľa u, tak jeho hodnotenie položky i je pravdepodobne podobné, ako hodnotenie tej istej položky i iným používateľom v, pokiaľ u a v hodotili iné položky podobne. Treba poznamenať, že tieto predpovede sú teda špecifické pre daného používateľa, ale vytvárajú sa na základe informácií zhromaždených od veľkého počtu iných používateľov. Collaborative filtering je považovaný za najpopulárnejšiu techniku používanú v odporúčacích systémoch.



Zdroj: `https://towardsdatascience.com/the-remarkable-world-of-recommender-systems-bff4b9cbe6a7`

Obr. 1: Collaborative filtering

Existujú dva prístupy pri používaní tejto techniky:

1. **Memory-based metódy** niekedy označované aj ako neighbourhood-based collaborative filtering metódy, sú metódy v ktorých hodnotenia položiek používateľom, sú predpovedané na základe jeho susedov. Týchto susedov môžeme ďalej definovať dvoma spôsobmi:

   - **User-based**
     Predpovedá hodnotenie používateľa u položky i, použitím hodnotení i od iných používateľov, ktorí sú čo najviac podobní u. Inak povedané snaží sa nájsť iných podobných používateľov (susedov) a odporúčať používateľovi u produkty ktoré sa páčia im.
   - **Item-based**
     Zatiaľ čo pri user-based metóde sa pri predpovedaní hodnotenia spoliehajú na názor rovnako zmýšľajúcich používateľov, item-based prístupy sa zameriavajú na hodnotenie udelené podobným položkám. Túto myšlienku môžeme sformalizovať nasledovne. Predpoveď hodnotenia používateľa u položky i môžeme získať ako vážený priemer hodnotení u iných položiek, ktoré sú čo najviac podobné i.

2. **Model-based methods** využívajú metódy strojového učenia na tvorbu predpovedí, pričom tento problém považujú ako normálny problém strojového učenia. Využívajú sa techniky ako PCA, SVD, faktorizácia matíc, clusterin či neurónové siete.

Collaborative filtering, hlavne typ neighbourhood-based, prináša jeho používaním zaujimavé výhody. Spomeňme pár z nich:

- **Jednoduchosť** - Neighborhood-based metódy sú intuitívne a ich implementácia je realtívne jednoduchá. V ich najjednoduchšej podobe vyžaduje vyladenie iba jeden parameter (počet susedov použitých v predikcii).
- **Účinnosť** - Jednou zo silných stránok, je ich efektívnosť. Na rozdiel od väčšiny model-based systémov nevyžadujú nákladné tréningové fázy, ktoré je potrebné vo veľkých komerčných aplikáciách vykonávať v častých intervaloch. Zatiaľ čo fáza odporúčaní je zvyčajne nákladnejšia ako pre model-based techniky, najbližších susedov je možné vopred vypočítať offline, čo poskytuje takmer okamžité odporúčania. Ukladanie najbližších susedov navyše vyžaduje veľmi málo pamäte, čo robí tento prístup škálovateľným na aplikácie s miliónmi používateľov.

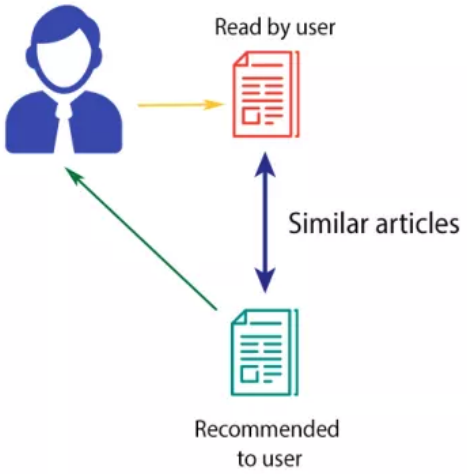
- **Zdôvodnenie odporúčaní** - Takéto metódy tiež poskytujú stručné a intuitívne odôvodnenie vypočítaných predikcií. Napríklad v item-based odporúčaní, zoznam susedových položiek, ako aj hodnotenie, ktoré používateľ týmto položkám dal, sa môžu používateľovi zobraziť ako odôvodnenie odporúčania. To môže pomôcť používateľovi lepšie pochopiť odporúčanie resp. jeho relevantnosť a môže slúžiť ako základ pre interaktívny systém, kde si môžu používatelia vybrať susedov, ktorým by sa mala venovať väčšia dôležitosť.

Hlavnou nevýhodou týchto systémov je tzv. "cold start problem", čo znamená, že je takmer nemožné niečo odporučiť novému používateľovi, alebo odporúčať novú položku používateľom, pokiaľ či už používateľ alebo položka nemájú žiadne interakcie. Navyše mnoho používateľov a položiek má príliš málo interakcií nato, aby algoritmus s nimi vedel na začiatku efektívne pracovať. Riešenia tohto problému bývajú, že novým používateľom sa odporúčaju náhodné položky a nové položky sa odporúčajú náhodným používateľom (tzv. "random strategy"). Často sa používa aj odporúčanie populárnych položiek novým používateľom a odporúčanie nových položiek najviac aktívnym používateľom (tzv. "maximum expectation strategy"). V skorej fáze "života" používateľa alebo položky, sa môže použiť iná metóda ako collaborative filtering.

1.1.6 Content based filtering

Táto technika zahŕňa odporúčanie položiek na základe ich samotných vlastností. Odporúčania sú tu vytvárané na základe predchádzajúcich interakcií jednotlivých používateľov s položkami. Systém pracujúci touto metódou, sa snaží hladať podobnosti medzi položkami, s ktorými mal používateľ v minulosti pozitívnu interakciu (t.j. kúpil si daný produkt, ohodnotil kladne daný film, pridal skladbu do obľúbených atď.). Napríklad ak používateľ pozitívne ohodnotí film zo žánru komédia, systém z toho môže vyčítať, že do jeho preferencií patrí aj tento žáner a teda mu odporúčať komediálne filmy. Princíp fungovania je znázornený aj na obrázku 2.

Nato aby táto technika správne fungovala, si systém musí vytvoriť pri danom používateľovi akýsi model, resp. profil, ktorý reprezentuje používateľove preferencie, ktoré získa na základe vlastností pozitívne hodnotených položiek. Proces tvorby odporúčaní potom pozostáva z hľadania zhody medzi atribútmi profilu a atribútmi prehladávaných položiek. Položky, ktoré su odporúčané používateľovi, sú teda reprezentované, ako súbor vlastností tiež nazývaných aj atribúty. Napríklad pri odporúčaní filmov, vlastnosti, ktoré opisujú film sú herci,režiséry,žánre,hodnotenie filmu atď.



Zdroj: `https://towardsdatascience.com/the-remarkable-world-of-recommender-systems-bff4b9cbe6a7`

Obr. 2: Content based filtering

Na získavanie feedbacku od používateľa sa používajú dva spôsoby:

- **like/dislike** - položky sú jednoducho klasifikované, buď ako relevantné, alebo irelevantné (binárne hodnotenie)
- **rating** - používa sa diskrétna numerická stupnica (numerické hodnotenie)

Výhody content based filteringu

Implementácia content based odporúčaní má niekoľko výhod oproti ostatným technikám.

- **Nezávislosť od používateľov** - Tá je dosiahnutá vďaka tomu, že profil používateľa, na základe ktorého sa hladajú zhody s položkami, je vytvorený len z hodnotení položiek daným používateľom. Naproti tomu, napríklad technika collaborative filtering potrebuje hodnotenia aj od iných používateľov, čo môže byť problém pri aplikáciách, kde nieje dostatočne veľký počet používateľov.
- **Transparentnosť** - Zistiť ako systém funguje, je možné pomocou atribútov, ktoré model zohladňuje pri vytváraní profilu. To vie pomocť pri porozumení, prečo sa konkrétna položka objavila v zozname odporúčaných. Collaborative filtering systémy sa podstatne ťahšie analyzujú, keďže jediné vysvetlenie prečo je položka odporúčaná je, že neznámy používateľ, s podobnými preferenciami, pozitívne hodnotil danú položku.

- **Nová položka** - V content based metóde je možné odporúčať aj novú položku, ktorá nebola ešte hodnotená žiadnym používateľom. Vďaka tomu netrpí tzv. "frist-rater" problémom, ktorý postihuje collaborative filtering. Ten sa pri tvorbe odporúčaní spolieha na používateľské preferencie. To spôsobuje, že pokial nová položka nebude hodnotená dostatočným počtom používateľov, systém nebude schopný ju odporúčať.

Nevýhody content based filteringu

Napriek spomenutým výhodám, použitie tejto techniky prináša aj isté obmedzenia.

- **Nový používateľ** - Nato aby systém dokázal naozaj dobre porozumieť používateľovým preferenciám a vytváral presné odporúčania, je nevyhnutné mať zozbieraných dostatok hodnotení. Preto pri nových používateľoch, ktorí majú zo začiatku málo hodnotení, systém väčšinou neprináša spolahlivé odporúčania.
- **Over-specialization** - Známy problém, kedy systém odporúča položky, ktoré sú počas procesu zhody s používateľovým profilom označené ako úžitočné. Tie však často bývajú podobné, ako položky, ktoré už hodnotil. Tento problém je často označovaný aj ako "serendipity problem", čím sa zdôrazňuje tendencia systému produkovať odporúčania, ktoré sa často podobajú a neobmieňajú. Napríklad ak používateľ pozitívne hodnotil filmy od režiséra Stanleyho Kubricka, v odporúčaniach mu môžu prevažovať filmy tohto typu.

1.2 Mobilné aplikácie

Mobilná aplikácia je softvérová aplikácia vytvorená špecificky pre mobilné zariadenia ako napríklad smartfóny, tablety alebo inteligentné hodinky. Pôvodne boli aplikácie vytvárané výrobcami mobilných operačných systémov, ktorí potrebovali pre používateľov zjednodušiť používanie základných funkcií smartfónu, ako napríklad prezeranie emailov, správ o počasí, prácu s kalendárom, fotenie fotografií atď. Avšak, vďaka rýchlemu vývinu samotných smartfónov a ich operačných systémov, začal rásť dopyt aj po aplikáciách zameraných na iné oblasti. V dnešnej dobe sú najpopulárnejšie rôzne herné aplikácie, navigačné aplikácie, aplikácie na online komunikáciu, hudobné aplikácie a mnohé iné. V posledných rokoch si používanie smartfónu bez spomenutých aplikácií ani nevieme predstaviť a stali sa ich neoddeliteľnou súčasťou. Aplikácie sa väčšinou sťahujú z distribučných platforiem, ktoré sú prevádzkované vlastníkom daného operačného systému, na ktorý je aplikácia určená. Spomeniem dva najväčšie a to App Store patriaci pod operačný systém

iOS a Google Play Store patriaci pod Android. Na oboch platformách vieme nájsť veľké množstvo aplikácií všemožného zamerania, pričom každým dňom pribúdajú ďalšie. Niektoré sú bezplatné, iné spoplatnené tvorcom, pričom zárobok z nej sa delí medzi tvorcu aplikácie a distribučnú platformu.

1.2.1 Typy mobilných aplikácií

Vo všeobecnosti sa mobilné aplikácie delia na 3 základné kategórie. Natívne aplikácie, webové aplikácie a hybridné aplikácie. Dnes však už je pri niektorých frameworkoch ťažké určiť, do ktorej kategórie presne patria, keďže kombinujú vlastnosti viacerých z nich, preto sme spomenuli aj tzv. cross-platform aplikácie.

- **Natívne aplikácie** - sú vytvorené výlučne pre špecifický mobilný operačný systém tj. napríklad natívne Android aplikácie alebo natívne iOS aplikácie. Kvôli špecifickému zameraniu na jeden operačný systém nie je možné aplikácie kombinovať na rôznych platformách. Napríklad Blackberry aplikácia nie je spustiteľná na Androide, Windows Phone aplikáciu zase nespustíme na iOS. Teda všeobecne povedané, mobilnú aplikáciu nainštalujete a spustíte len na operačnom systéme, pre ktorý je vytvorená. Inštalujú sa priamo do mobilného zariadenia, potrebné dáta sú väčšinou uložené priamo v internom úložišku zariadenia.

  Používané technológie: Na vývoj natívných aplikácií sa používajú viaceré programovacie jazyky podľa toho, pre aký OS majú byť určené. Medzi najpoužívanejšie patria Java a Kotlin pre Android, Swift a Objective-C pre iOS.

  Výhody: Vďaka tomu, že sú zamerané na jednu platformu, vedia byť z hľadiska výkonu rýchlejšie a stabilnejšie. Hardvér zariadenia vedia využívať efektívnejšie. Su schopné priamo pristupovať k všetkým funkciám zariadenia, vďaka čomu vedia využiť širokú ponuku možností, ktoré dané zariadenie ponúka ako napríklad fotoaparát, kontakty zariadenia, bluetooth, NFC či dokonca samotnú polohu zariadenia (GPS). Veľkú obľubu im zabezpečuje aj to, že využívajú natívny UI, čo prináša používateľom lepší UX. Niektoré nevyžadujú na funkčnosť internetové pripojenie.

  Nevýhody: Primárnym problémom je duplicita pri vývoji aplikácie, keďže je potrebné aplikáciu naprogramovať pre viaceré mobilné operačné systémy, čo priamoúmerne zvyšuje cenu nehovoriac o náročnosti údržby a aktualizácie kódu pri každej novej verzii.

Menší komfort spôsobuje aj fakt, že ak používateľ nechce prísť o najnovšiu funkcionalitu a opravu chýb, mal by pri každej aktualizácií aplikáciu preinštalovať resp. si nainštalovať tzv. update.

- **Webové aplikácie** - správajú sa podobne ako natívne aplikácie, najpodstatnejší rozdiel je v tom, že sa k nim pristupuje pomocou webového prehliadača. Sú to v podstate responzívne webové stránky, ktoré sa prispôsobujú svojim vzhľadom zariadeniu na ktorom sú spustené.

  Používané technológie: Webové aplikácie sú vytvorené pomocou HTML, ktorého vzhľad je naštýlovaný pomocou CSS a extra funkčnosť väčšinou zabezpečuje JavaScript.

  Výhody: Keďže na svoje fungovanie využívajú webový prehliadač, v tomto prípade zaniká problém duplicity pri programovaní aplikácie na viaceré operačné systémy. Toto znižuje náročnosť či už na vývoj, alebo cenu. Navyše odpadá potreba sťahovania, inštalácie a aktualizovania čo znamená, že aplikácia nezaberie žiadny priestor v úložisku zariadenia.

  Nevýhody: Hlavná nevýhoda pramení už z názvu - webové aplikácie, z čoho vyplýva, že bez internetového pripojenia ich nieje možné využívať. Webový prehliadač vie tiež ovplyvniť UX. Kým v jednom môže byť k dispozicií plná funkcionalita, môže sa stať, že na druhom už len obmedzená. Programátori sa tomu samozrejme snažia zabrániť a programovať aplikácie tak, aby boli plne funkčné na čo najväčšom množstve najviac používaných prehliadačov. Nevýhodou je tiež značne obmedzený prístup k pokročilejším funkciám zariadenia ako napríklad využívanie gest.

- **Hybridné aplikácie** - sú založené na pincípe miešania prvkov natívnych a webových aplikácií. Jadro aplikácie je napísane pomocou webových technológií (HTML, CSS, JavaScript), pričom je spúšťané z natívnej aplikácie a jej vlastného zabudovaného prehliadača, ktorý je ale pre používateľa neviditeľný. Napríklad aplikácia pre iOS by na zobrazenie používala WKWebView objekt, zatiaľ čo v Androide by na vykonávanie rovnakej funkcie používala WebView objekt. Kód samotný, je potom vložený do kontajnera

natívnej aplikácie s použitím frameworkov ako Apache Cordova (známy aj ako PhoneGap), alebo Ionic. Tieto frameworky navyše majú aj systém pluginov, ktorý umožňuje ľahko prekonať obmedzenia webových aplikácií a rozšíriť funkcionalitu nad rámec prehliadača. Aplikácia tak môže získať plnú kontrolu nad funkciami mobilného zariadenia, čo umožňuje napríklad použiť TouchID pri iOS ako možnosť prihlásenia sa do aplikácie.

Používané technológie: Hybridné aplikácie používajú kombináciu webových technológií a natívnych API. Sú vyvinuté pomocou technológií ako Ionic, Apache Cordova, Swift, HTML5, CSS, a JavaScript.

Výhody: Kombinácia dobrého UX, menšej náročnosti pri vývoji a prijateľná cena, sú často hlavné činiteľe v ktorých hybridné aplikácie predčia konkurenciu. Vďaka tomu, že je z velkej časti použitý rovnaký zdrojový kód, nezávisle od mobilného operančého systému. Na vývoj je potrebných menej vývojárov (napr. nemusia byť zvlášť tímy vývojárov pre iOS a zvlášť pre Android).

Nevýhody: Najvačším mínusom hybridných aplikácií je výkon. Keďže sa načítávajú v spomínanom WebView objekte podobnom webovému prehliadaču, sú vysoko závislé od jeho samotného výkonu keďže je zodpovedný za zobrazovanie UI a beh kódu. Napriek tomu, že aplikácie vedia bežať na viacerých operačných systémoch, je vcelku náročné si túto vlastnosť údržať. Občas vďaka výdavkom na implementáciu cross platformingu, vie cena hybridnej aplikácie dosiahnuť ceny natívnych aplikácií. Všetko však záleží od požiadaviek a od toho, ako veľmi sa chceme priblížiť k danej natívnej aplikácií.

- **Cross platform aplikácie** - sú špeciálnym typom aplikácií, ktoré sú veľmi podobné hybridným, pričom ale ponukajú aj vlastnosti natívnych aplikácií. Na ich vývoj sa používajú špeciálne frameworky, pričom kód je napísaný v bežných programovacích jazykoch ako C# alebo JavaScript. Aplikácie využívajú komponenty daných frameworkov, ktoré sú potom kompilované do natívneho kódu pre konkrétnu platformu.

  Používané technológie: Využívajú sa hlavne frameworky ako Xamarin, React Native a Flutter.

Výhody: Veľmi podobné hybridným aplikáciám. Jeden zdrojový kód kompilovateľný do viacerých OS, natívny UI a prijateľná cena vývoja. Oproti hybridným aplikáciám sú na tom lepšie čo sa týka výkonu, keďže nebežia vo WebView objekte.

Nevýhody: Vývojári sú pri vývoji značne obmedzený ponukou komponentov, ktoré daný framework poskytuje. Z hľadiska výkonu, sú na tom čisto natívne aplikácie stále o niečo lepšie. Pri jednoduchých aplikáciách to však nie je badateľné.

1.2.2 Prečo sme si vybrali React Native?

Pri rozhodovaní o hlavnej technológií na vývoj nášej aplikácie, sme uvažovali nad dvoma možnosťami. Porovnávali sme Android Studio, ktorého programovacím jazykom je Java a vytvorili by sme teda čisto natívnu aplikáciu pre Android a cross platformový framework React Native, ktorý využíva JavaScript. Po zvážení náročnosti jednotlivých riešení, ich výhod a nevýhod, potenciálu do budúcnosti a subjektívnych sympatií, sme vybrali React Native. Detailnejšie sa mu budeme venovať v nasledujúcej kapitole.

1.3 React Native

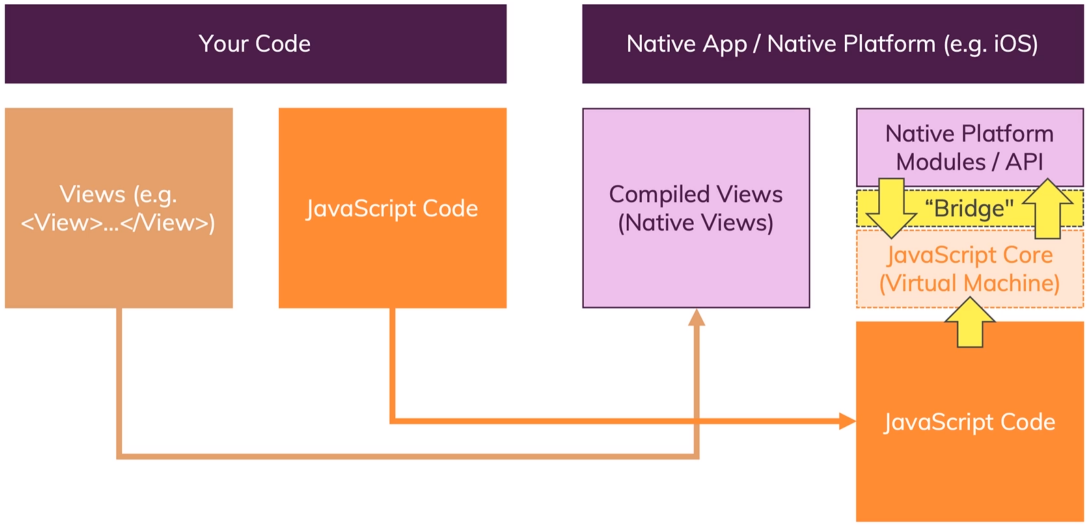
React Native je cross-platformový framework na vývoj mobilných aplikácií, ktorý je postavený na populárnom webovom frameworku React. Rovnako ako React, aj React Native je open-source projekt udržiavaný prevažne vývojármi Facebooku a Instagramu. Potom ako sa v roku 2012 Mark Zuckerberg vyjadril, že používanie veľkého množstva HTML5 v mobilnej aplikácií Facebooku považuje za chybu. Vzhľadom nato, že výsledkom bola nestabilita aplikácie a pomalé načítavanie dát, sa Facebook rozhodol, že ich prioritou bude zlepšiť UX ich mobilnej aplikácie. React na ktorom bol neskôr v roku 2015 postavený React Native, vytvoril Jordan Walke, softvétový inžinier Facebooku. Jeho prvý prototyp, ktorý bol inšpirovaný XHP (rozšírenie PHP ktoré umožňuje vytvárať upravené HTML elementy) sa nazýval FaxJS. Použitý bol prvýkrát v roku 2011 vo Facebookovom news feede a neskôr v roku 2012 na Instagrame. React Native bol ohlásený v roku 2015 na konferencii Facebooku a prvá verzia vyšla 26. marca toho roku. Od vtedy zaznamenal tento framework nárast popularity naprieč celým spektrom vývojárov mobilných aplikácií a je čoraz častejšie používaný. Sú pomocou neho vytvorené viaceré známe mobilné aplikácie ako Facebook, Instagram, Skype, Airbnb, Bloomberg, SoundCloud a mnohé iné.

1.3.1 Princíp fungovania

Pri vytváraní aplikácií pomocou React Native, je všetka logika, volania API a management stavov v JavaScripte. V porovnaní s inými hybridnými frameworkami, aplikácia vytvorená v React Native nebeží vo WebView, ale používa tzv. komponenty ktorých logika a štýlovanie je napísané priamo v kóde, ale vyrendrované sú konkrétne do natívnej podoby podľa platformy. Vďaka tomu je zaručené vysoké percento recyklovania kódu (cca. medzi 85 - 99 %), pričom UI je prispôsobené konkrétnej platfrome. V prípade, ak je to potrebné, React Native umožňuje písať kód, logiku a štýlovanie špecificky pre konkrétnu platformu.

1.3.2 Bridge

React Native aplikácia je zložená z dvoch častí, JavaScript kódu a natívneho kódu. Z technologického hľadiska je zaujimavé, že natívny kód je napísaný diametrálne odlišnými jazykmi. Pri iOS sa napríklad používa Objective-C alebo Swift, pri Androide Java či Kotlin. Bridge je koncept, ktorý umožňuje a zabezpečuje komunikáciu medzi týmito dvoma časťami. Je jedným z najdôležitejších princípov, bez ktorého by si natívny kód a JavaScript kód nevedeli medzi sebou vymieňať žiadne informácie.



Zdroj: `https://www.udemy.com/course/react-native-the-practical-guide/`

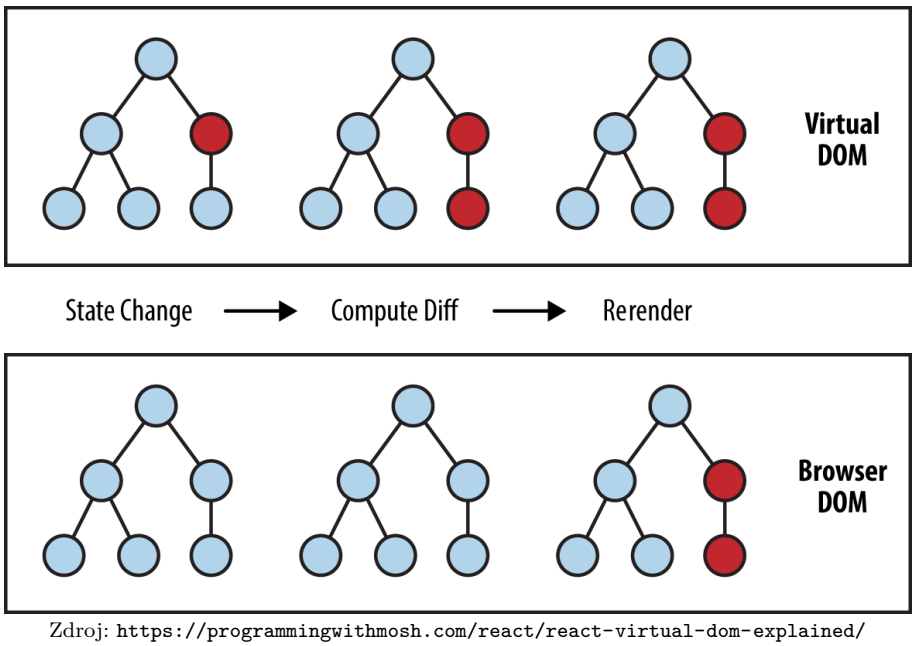
Obr. 3: Ukážka fungovania React Native aplikácie

1.3.3 Virtual DOM (Document Object Model)

Ďalším dôležitým koncepotom v React Native, je tzv. Virtual DOM. Predtým, ako si vysvetlíme pracovanie s Virtual DOM v React Native, sa nato najprv pozrieme z pohľadu Reactu. DOM je vo všeobecnosti skratka pre Document Object Model (inak nazývaný aj

real DOM), čo je programové rozhranie pre HTML a XML dokumenty. DOM reprezentuje dokument ako skupinu uzlov (tzv. "nodes") a objektov. To umožňuje jazykom ako napríklad JavaScript modifikovať dané uzly a tým aj celý dokument. DOM je reprezentovaný ako dátovy strom, vďaka čomu je každá zmena rýchla. Avšak po tejto zmene, zmenené elementy a ich potomkovia musia byť nanovo vyrendrované aby nastal aj priamo update UI danej aplikácie. Proces rendrovania je to, čo spôsobuje nezanedbateľné spomalenie výkonu, ktoré je navyše priamo úmerné so zväčšujúcim sa počtom UI komponentov, ktoré treba nanovo vyrendrovať.

Tu prichádza na scénu Virtual DOM a dosahuje podstatne lepšie výkonostné výsledky ako real DOM. Virtual DOM je iba virtuálne znázornenie DOM. Vždy, keď sa zmení stav aplikácie, namiesto real DOM sa aktualizuje virtual DOM. Keď je do UI pridaný nový element, vytvorí sa virtual DOM (reprezentovaný ako strom). Akonáhle sa zmení stav ktoréhokoľvek elementu, vytvorí sa nový virtual DOM, prebehne proces porovnania (nazývaný "diffing") aktuálnej verzie a prechádzajúcej verzie vritual DOM. Potom sa vypočíta najlepší možný spôsob ako tieto zmeny premietnuť do real DOM. Akonáhle React vie, ktoré elementy vo virtual DOM boli zmenené, zaktualizuje len dané elementy v real DOM. Vďaka tomu je výkon neporovnateľne lepší v porovnaní s priamou manipuláciou s DOM.



Zdroj: `https://programmingwithmosh.com/react/react-virtual-dom-explained/`

Obr. 4: Virtual DOM vs. real DOM

Na obrázku 4 vidíme červené kruhy, ktoré znázorňujú zmenené uzly. Tieto uzly reprezentujú konkrétne UI elementy, ktorých stav sa zmenil. Následne je vypočítaný rozdiel

medzi aktuálnou a predchádzajúcou verziou virtual DOM stromu, pričom celý podstrom rodiča je nanovo prerendrovaný a tým poskytne aktualizáciu UI. Aktualizácie real DOM sa posielajú hromadne, namiesto odosielania aktualizácií pre každú jednu zmenu stavu.

Spôsob akým virtual DOM využíva samotný React Native je vo veľkej miere podobný. Hlavnou odlišnosťou je, že sa namiesto webových komponentov rendrujú natívne komponenty, tým pádom sa nepoužívajú webové technológie.

1.3.4 Základné komponenty

Kľúčovým prvkom každej React Native aplikácie sú komponenty. Každý komponent má inú úlohu, no dokopy tvoria celok - samotnú aplikáciu. Z hľadiska koncepcie môžeme povedať, že komponenty sú podobné JavaScript funkciám. Vedia príjímať vstupy (nazývane "props"), s ktorými potom umožňujú pracovať vo vnútri komponentu a vracajú elementy ktoré popisujú čo sa má zobraziť na obrazovke. V React Native existujú dva hlavné typy komponentov, ktoré tvoria aplikáciu. Sú štruktúrované rovnako, ako v bežnej webovej aplikácii vytvorenej pomocou Reactu.

- **Class komponenty**
  Sú to triedy rozširujúce základnú triedu z Reactu s názvom Component. Majú prístup k lifecycle metódam Reactu ako napríklad render či state/props funkcionalite od rodičovskej triedy. V súčasnosti sú menej používané kvôli tomu, že sú komplikovanejšie ako functional komponenty. Aj keď stále existujú prípady, v ktorých je ich syntax potrebná, vo všeobecnosti sa pri vytváraní nového komponentu viac používa syntax functional kompononetu. Vo výpise 1 máme úvedený aj jednoduchý príklad class komponentu.

```
import React, { Component } from 'react';
import { Text } from 'react-native';

class Cat extends Component {
  render() {
    return (
      <Text>This is class component!</Text>
    );
  }
}

export default Cat;
```

Listing 1: Príklad class komponentu

- **Functional komponenty**
  Sú najjednoduchší spôsob vytvorenia komponentu. Ich deklarácia je v podstate rov-

naká ako pri obyčajnej JavaScript funkcii. V minulosti sa za ich nevýhodu oproti class komponentom považovalo to, že neumožňovali správu stavov (states) a nemali prístup k lifecycle metódam ktoré React Native poskytuje. To sa zmenilo až po vydaní verzie 16.8.0 v roku 2019, v ktorej vývojári pridali Hooks, čím tento problém zanikol. Odvtedy sa ich popularita ešte zvýšila a stali sa novým štandartom. Na výpise 2 môžeme vidieť, že je jednoduchšie ich udržiavať "light weight" a kód je pri nich čitateľnejší, ako pri class komponentoch.

```
import React from 'react';
import { Text } from 'react-native';

const Cat = () => {
  return (
    <Text>This is functional component!</Text>
  );
}

export default Cat;
```

Listing 2: Príklad function komponentu

React Native navyše poskytuje aj sadu predpripravených hotových natívnych komponentov, ktoré sa dajú veľmi jednoducho použiť pri programovaní aplikácie. Patria medzi ne napríklad View, Text, Button, Image, či TextInput.

1.3.5 Props a state

Props a State sú dva typy dát, pomocou ktorých vieme pracovať s komponentami.

- **Props**

  Props (z anglického slova "properties") je obdoba argumentov pri klasických funkciách v JavaScripte alebo atribútov pri HTML. Komponenty prijímajú props od rodičovského komponentu. Ich dôležitou vlastnosťou je, že sú tzv. "immutable" tj. nemenné vo vnútri komponentu. V Reacte a v React Native smerujú dáta jedným smerom - od rodičovských komponentov k detským. Celý koncept props je založený natom, že si viete vytvoriť jeden komponent, ktorý je možné použiť na viacerých miestach v aplikácii. Rodičovské komponenty potom vedia zavolať váš vytvorený komponent, pričom na rôznych miestach vie mať rozličné props. Props v zásade pomáhajú písať znovu použiteľný kód.

- **State**

  State pracuje odlišne v porovnaní s props. Je to interná vlastnosť komponentu, ktorá

pomáha v rámci komponentu sledovať určité informácie. Použitím "setState" sa daný komponent aj jeho detské komponenty nanovo vyrendrujú, vďaka čomu sa nemusí programátor zaoberať implementáciou event handlerov ako v iných jazykoch. State sa používa v situáciách, keď sa menia dáta v rámci komponentu. Dobrým príkladom je napríklad interakcia používateľa s komponentom, pri klikntí na tlačidlo alebo zaškrtnutí checkboxu. Napríklad pri vypĺňaní formuláru má každý z textových inputov svoj vlastný state. Ak vyplníte daný input, automaticky sa mení jeho state, čo spôsobuje prerendrovanie celého komponentu a všetkých jeho detskych komponentov.

1.3.6 Hooks

Hooks boli predstavené vo verzii 16.8.0 v roku 2019. Ich hlavnou úlohou je umožniť používať state a lifecycle metódy vo functional komponentoch, čo predtým bolo možné iba vytvorením class komponentov (v nich Hooks nie sú použiteľné). Ich uvedenie výrazne zvýhodnilo používanie functional komponentov oproti class komponentom. Tak isto ako pri komponentoch, aj tu React Native umožňuje využiť predpripravené Hooks priamo od vývojárov. Najčastejšie používané sú useEffect a useState. V prípade potreby je možné si vytvoriť aj vlastný hook tak, aby spĺňal požadovanú funkcionalitu.

1.4 Expo alebo React Native CLI ?

Predtým ako sa programátor pustí do vývoja aplikácie pomocou React Native, ho čaká rozhodnutie či použiť pomocný framework Expo, alebo vstavanú funkcionalitu React Native CLI. Nato aby sme mohli vytvorenú aplikáciu vôbec spustiť, potrebujeme jednu z týchto technológii.

1.4.1 Expo

Framework používaný pri vytváraní React Native aplikácií. Je to balík nástrojov a služieb vytvorených pre React Native, ktoré pomáhajú pri vývoji aplikácie.

Výhody

- Nevyžaduje vedomosť programovania v natívnom kóde.
- Nepoužíva Xcode alebo Android Studio.
- Najrýchlejší a najjednoduchší spôsob, ako vytvoriť natívne React Native aplikácie.
- Uvolňuje OTA updates.

- Vstavaný prístup k natívnym APIs.

Nevýhody

- Ďalšia vrstva abstrakcie.
- Neumožňuje zasahovať do natívneho kódu.
- Niesu k dispozicií všetky iOS a Android APIs.

1.4.2 React Native CLI

Vstavaný nástroj v React Native, ktorý pomáha spustiť aplikáciu.

Výhody

- Umožňuje zasahovať do natívneho kódu.
- Vieme pomocou neho pridávať natívne moduly (Objective C/Java).

Nevýhody

- Vývoj iOS aplikácie nie je možný na inom OS ako MacOS.
- Na vytvorenie projektu sa vyžaduje Android Studio alebo XCode.
- Nastavenie projektu (vrátane konfigurácie) je pomerne komplikované a nepohodlné.
- Neposkytuje niektoré JavaScript APIs napr. notifikácie, je potrebné ich ručne doinštalovať.

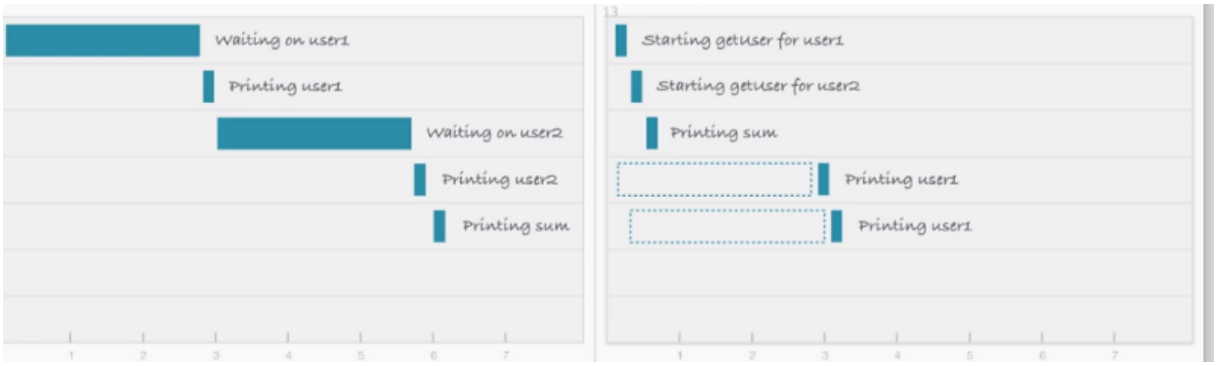
1.4.3 Odôvodnenie výberu

V našom prípade sme si vybrali pracovať s Expom, keďže práca s ním je jednoduchšia a priamočiarejšia a vzhľadom na typ a vlastnosti aplikácie, nám jeho funkcionalita plne postačuje.

1.5 Node.js

Node.js je open-source, cross-platformové, back-endové runtime prostredie JavaScriptu, ktoré beží na Chrome V8 JavaScript engine a vykonáva kód JavaScript mimo webového prehliadača. Tento engine prekladá JavaScriptový kód do rýchlejšieho strojového kódu. Strojový kód je nízkoúrovňový kód, ktorý počítač dokáže prečítať bez potreby ďalšej interpretácie. Vznik node.js bol logickým krokom po tom, ako vývojári JavaScriptu rozšírili použiteľnosť jazyka z čisto skriptovacieho, ktorý bolo možné používať a spúšťať iba v prehliadači, na jazyk ktorý umožňuje vytvoriť samostatnú aplikáciu spustiteľnú aj mimo prehliadača.

Nesporným prínosom node.js je, že využíva tzv. event-driven non-blocking I/O model, ktorý ho robí efektívnym. I/O alebo aj input/output, môže byť čokoľvek od čítania resp. zapisovania lokálnych súborov, až po vytváranie HTTP requestov na API. Non - blocking znamená, že napríklad pri requeste na vytiahnutie údajov dvoch používateľov z databázy, daný request neblokuje vykonávanie ďalšieho kódu počas čakania na odpoveď. Na obrázku 5 nižšie možeme vidieť porovnanie blocking (vľavo) a non blocking modelu (vpravo) na spomenutom príklade s databázou.



Zdroj: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

Obr. 5: Porovnanie blocking a non-blocking modelu

1.5.1 Yarn

Je package manager pre JavaScript, ktorý pomáha riadiť dodatočne doinštalované knižnice a rôzne iné dependencies v našom projekte. Keďže v súčasnej dobe sa pri programovaní využíva veľké množstvo knižníc a dependencies, aby sa zjednodušila práca programátora a nemusel istú funkcionalitu programovať nanovo, je odporúčané package manager používať.

Node.js prichádza s predinštalovaným package managerom npm. Napriek tomu sme

sa rozhodli použiť yarn, ktorý bol vytvorený Facebookom, tak isto ako React Native a preto práca s ním bola konzistentnejšia a bez väčších problémov na rozdiel od npm.

1.5.2 Axios

Axios je knižnica pre JavaScript, ktorá sa používa na vytváranie HTTP requestov z node.js alebo XMLHttpRequests z prehliadača. Podporuje aj ES6 promise API, pričom ešte viac uľahčuje celý proces okolo requestov tým, že ešte viac zlepšila dobre fungujúcu fetch() funckiu z JS a vylepšila error handling.

1.6 TMDb API

The Movie Database (TMDb), je databáza filmov a TV seriálov, ktorá bola vytvorená v roku 2008 úzkou komunitou ľudí, ktorá sa neskôr postupne rozrástla. Dnes je TMDb používaná vyše 400 000 developermi a firmami. Úspech im prinisela hlavne ich dostupná API, ktorá je zadarmo a ponúka širokú škálu queries pomocou ktorých je možné získať podrobné informácie o filmových tituloch. Ich veľkou prednosťou je aj množstvo dostupných titulných plagátov k filmom vo vysokej kvalite. Jedinou jej nevýhodou je, že popularitou zatiaľ nedosahuje úroveň najväčšej databázy na svete IMDB a pri niektorých filmoch niesú ich hodnotenia kredibilné.

1.7 Firebase

Firebase je platforma vyvinutá spoločnosťou Google, ktorá pomáha vytvárať, zlepšovať a rozširovať funkcionalitu aplikácií. Obsahuje sadu nástrojov , ktoré pokrývajú veľkú časť služieb, ktoré by si vývojári museli inak sami naprogramovať. Patria sem napríklad analytika, autentifikácia používateľov, databázy, konfigurácia, ukladanie súborov, doručovanie push notifikácií a mnoho iného. Služby sú uložené v cloude a sú škálovateľné s malým resp. minimálnym úsilím zo strany vývojára.

1.7.1 Firebase authentication

Poskytuje backendové služby, jednoducho použiteľné SDK a predpripravené UI knižnice na autentifikáciu používateľov v aplikácii. Podporuje autentifikáciu pomocou hesla, telefónneho čísla, Googlu, Facebooku a Twitteru. Je úzko integrovaná s ostatnými službami Firebase a využíva štandardy ako OAuth 2.0 a OpenID Connect, takže ju možno ľahko integrovať do vlastného backendu. Okrem autentifikácie umožňuje aj registráciu a

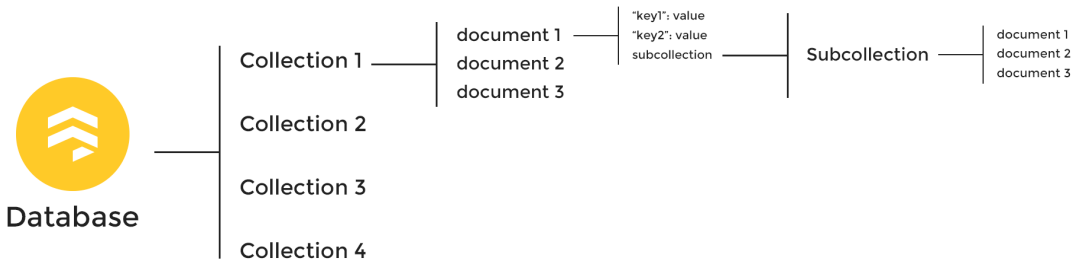
správu zaregistrovaných používateľov. V našej aplikácií sme si zvolili použiť autentifikáciu pomocou hesla.

1.7.2 Cloud firestore

Cloud Firestore je škálovateľná NoSQL cloudová databáza, pre vývoj mobilných a webových aplikácii. Udržuje dáta synchronizované medzi klientskými aplikáciami prostredníctvom listenerov v reálnom čase. Ponúka bezproblémovú integráciu s ostatnými produktami Firebase a Google Cloud vrátane cloudových funkcií.

Vo firestore dátovom modeli (pozri obr. 6), sa dáta ukladajú do tzv. dokumentov, ktoré obsahujú polia (fields) namapované na konkrétne hodnoty. Tieto dokumenty sú ukladané do kolekcií, ktoré slúžia ako kontainer pre viacero dokumentov. Dokumenty podporujú mnoho rôznych dátových typov, od jednoduchých ako sú napríklad reťazce alebo čísla, až po komplexnejšie ako mapy, polia, alebo komplexné vnorené objekty. Vytvoriť komplexnú databázu umožňuje vytváranie subkolekcií v rámci dokumentu (tj. kolekcie osbahujú dokumenty, ktoré obsahujú kolekcie a tie obsahujú ďalšie dokumenty atď.). [2]

Friebase ponúka aj druhý typ databázy, Firebase Realtime Database, čo je tiež NoSQL cloudová databáza. Avšak, oproti firestore ponúka jednoduchší dátový model, v ktorom sa dáta ukladajú iba vo formáte JSON, čo je vhodné iba pri jednoduchšej dátovej štruktúre. Taktiež nepodporuje používanie zložitejších queries. Firestore bol teda v našom prípade lepšou voľbou v oboch prípadoch.



Zdroj: https://waelyasmina.com/firebase-cloud-firestore-tutorial-web/

Obr. 6: Znázornenie dátového modelu Cloud Firestore

2 Analytická a návrhová časť

2.1 Analýza problému

V dnešnej dobe, kedy ponuka produktov v jednotlivých odvetviach je enormná, je častým problémom pre používateľov vyfiltrovať len tie produkty, o ktoré majú reálne záujem a vyhovujú ich kritériám. To podmienilo vznik odporúčacích systémov (kapitola 1.1), ktoré sa dnes vo veľkej miere využívajú.

Jedným z prípadov, pri ktorom sa s daným problémom často stretávame, je aj výber filmového titulu. S príchodom streamovacích služieb ako Netflix, Amazon Prime, Hulu či HBO GO, vďaka ktorým sú dostupné milióny titulov, je veľmi náročné vybrať si tie, ktoré budú vyhovovať osobným preferenciám jednotlivca. Väčšina spoločností to vyriešila zavedením odporúčacích systémov a používaním rôznych rebríčkov popularity. To však vyriešilo iba situáciu, ak sa človek rozhodne pozerať film sám. Bežne sa však stáva, že ľudia chcú pozerať film vo dvojici a pri spoločnom výbere filmu strávia neprimerane veľa času prezeraním rôznych filmových databáz a rebríčkov filmov.

To nám vnuklo myšlienku, že by sa tento proces výberu dal zjednodušiť moderným riešením pomocou mobilnej aplikácie, založenej na odporúčacích systémoch.

2.2 Existujúce riešenia problému

Na začiatku práce sme analyzovali už existujúce riešenia. Je dôležité poznamenať, že v čase zadania práce nebola dostupná aplikácia, ktorá by obsahovala podobnú funkcionalitu a bola by dostupná na oba spomínané operačné systémy.

2.2.1 Android OS

Na OS Android bola dostupná len 1 aplikácia vytvorená v roku 2019, avšak už dlhšie nepodporovaná vývojárom. Aplikácia s názvom MovieMatch má však viacero nevýhod oproti riešeniu, ktoré sme zvolili my. Rozdiel je napríklad v tom, že nie je možné v nej hodnotiť filmy swipeovaním bez toho, aby používateľ bol súčasťou skupiny. Po nainštalovaní a spustení, vygeneruje aplikácia kód, ktorý treba poslať iným používateľom, aby sa mohli pripojiť do jeho skupiny. Tak isto sa používateľ vie pripojiť do inej skupiny pomocou kódu danej skupiny. Toto riešenie spôsobuje, že používateľ je závislý od iných používateľov a bez nich je aplikácia nepoužiteľná. Veľkou slabinou je aj spôsob implementácie ako webovej aplikácie (otvára sa v prehliadači). To vzhľadom na štandardy vývoja mobilných aplikácií, nie je dostatočne moderné riešenie a UI oproti natívnym a hybridným (resp. cross platformovým) aplikáciám pôsobí zastaralo. Detailnému porovnaniu typov mobilných aplikácií sme sa venovali v kapitole 1.2.

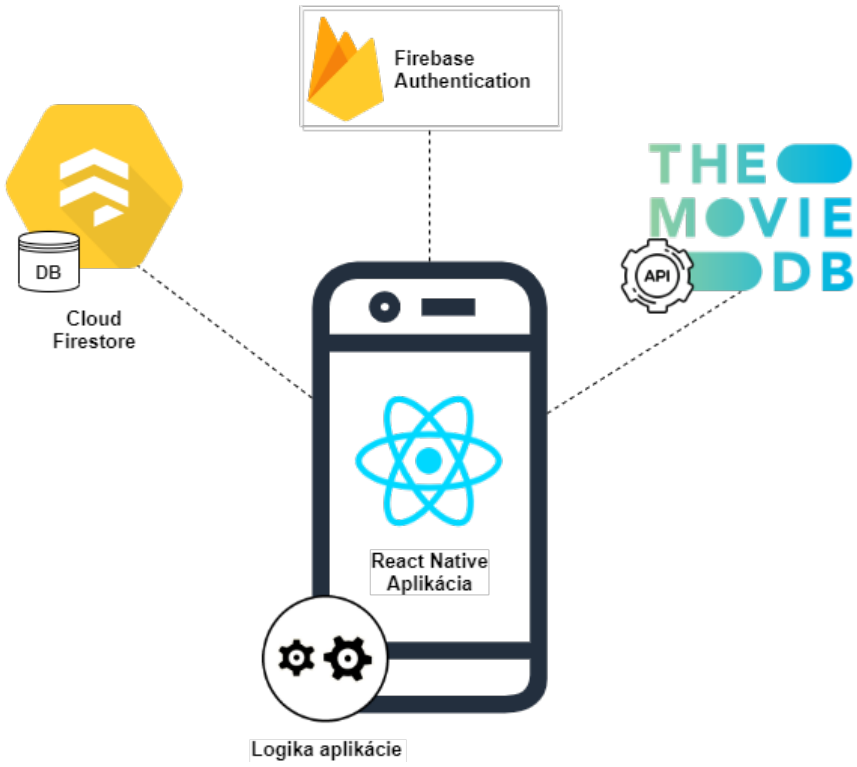
2.2.2 iOS

Na iOS v čase zadania práce podobná aplikácia nebola k dispozicií. To sa neskôr zmenilo, keď v novembri 2020 vyšla aplikácia Movie Match. Funkcionalitou a vzhľadom je na tom omnoho lepšie ako spomínaná Android aplikácia. Ako nevýhodu však považujeme, že ak sa používateľ chce spojiť s iným používateľom, musí to riešiť len zadaním jeho emailu, na ktorý mu aplikácia pošle pozvánku vo forme emailu. Toto riešenie je nepraktické, najmä ak je už používateľ v aplikácií zaregistrovaný. Intuitívnejším riešením by bolo mať takéto pozvánky zobrazené priamo v aplikácií.

2.3 Návrh riešenia

Cieľom tejto bakalárskej práce je navrhnúť a implementovať mobilnú aplikáciu, ktorá bude slúžiť ako pomocník pri spoločnom výbere filmu dvoch používateľov. Na základe ich osobného profilu preferencií jednotlivých atribútov filmov, sa im vygenerujú odporúčané filmy, ktoré budú môcť obaja hodnotiť a následne si pozrieť zhody. Používateľ bude vedieť hodnotiť filmy aj samostatne bez nutnosti byť prepojený s iným používateľom. Hodnotenie filmov bude vo forme binárneho hodnotenia(like/dislike) a implementované pomocou známeho swipeovania doprava a doľava. Celý tento princíp obsahuje prvky content based filteringu, čo je jedna z metód odporúčacích systémov (kapitola 1.1.6).

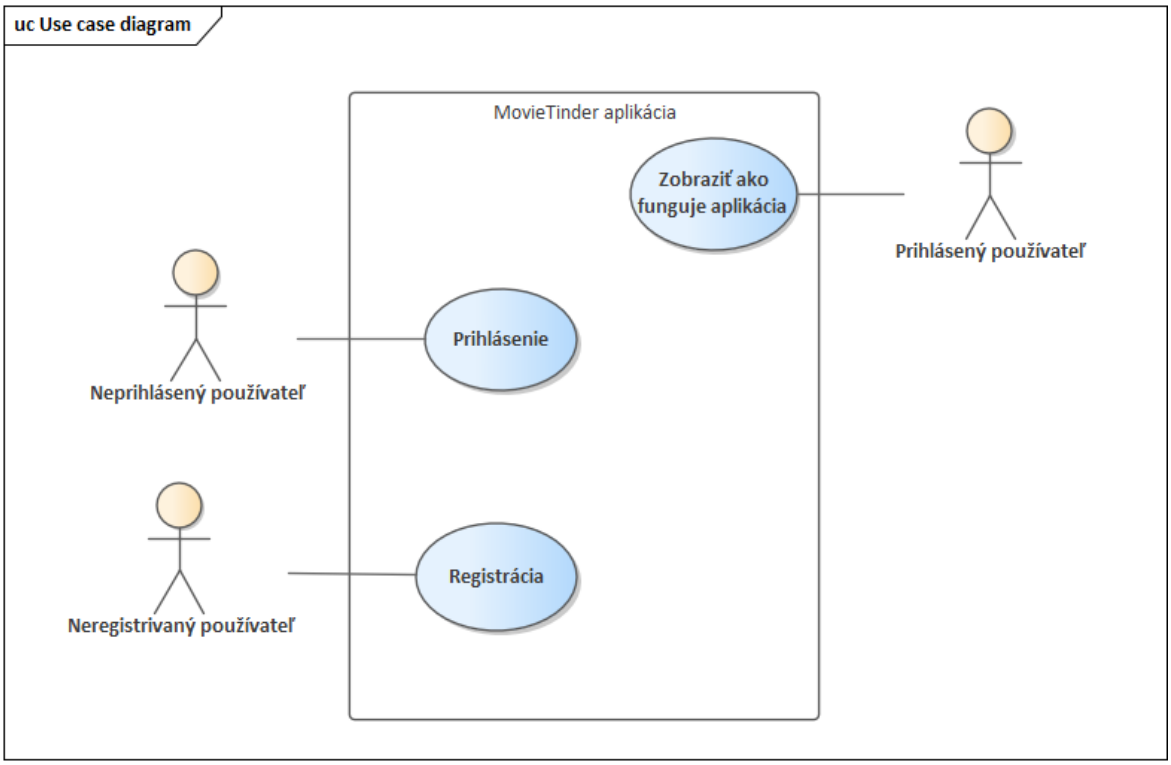
Všetka logika aplikácie prebieha na klienstkej strane aplikácie, keďže komplexnosť aplikácie si nevyžadovala rozdelenie na frontend a backend. V rámci implementácie, sme sa rozhodli, že by bolo vhodné, aby fungovala pre oba populárne operačné systémy (Android aj iOS), vďaka čomu používatelia nebudú obmedzovaní a viazaní na jeden operačný systém. Zvolením frameworku React Native sme teda zabezpečili vytvorenie cross platformovej aplikácie. Autentifikácia používateľa je formou emailu a hesla implementovaná pomocou Firebase Authentication. Ako úložisko dát nám slúži cloudová NoSQL databáza Firestore. Ako zdroj filmových dát sme zvolili databázu TMDb, pričom na prácu s dátami sme využívali aj ich API.



Obr. 7: Znázornenie konceptu aplikácie

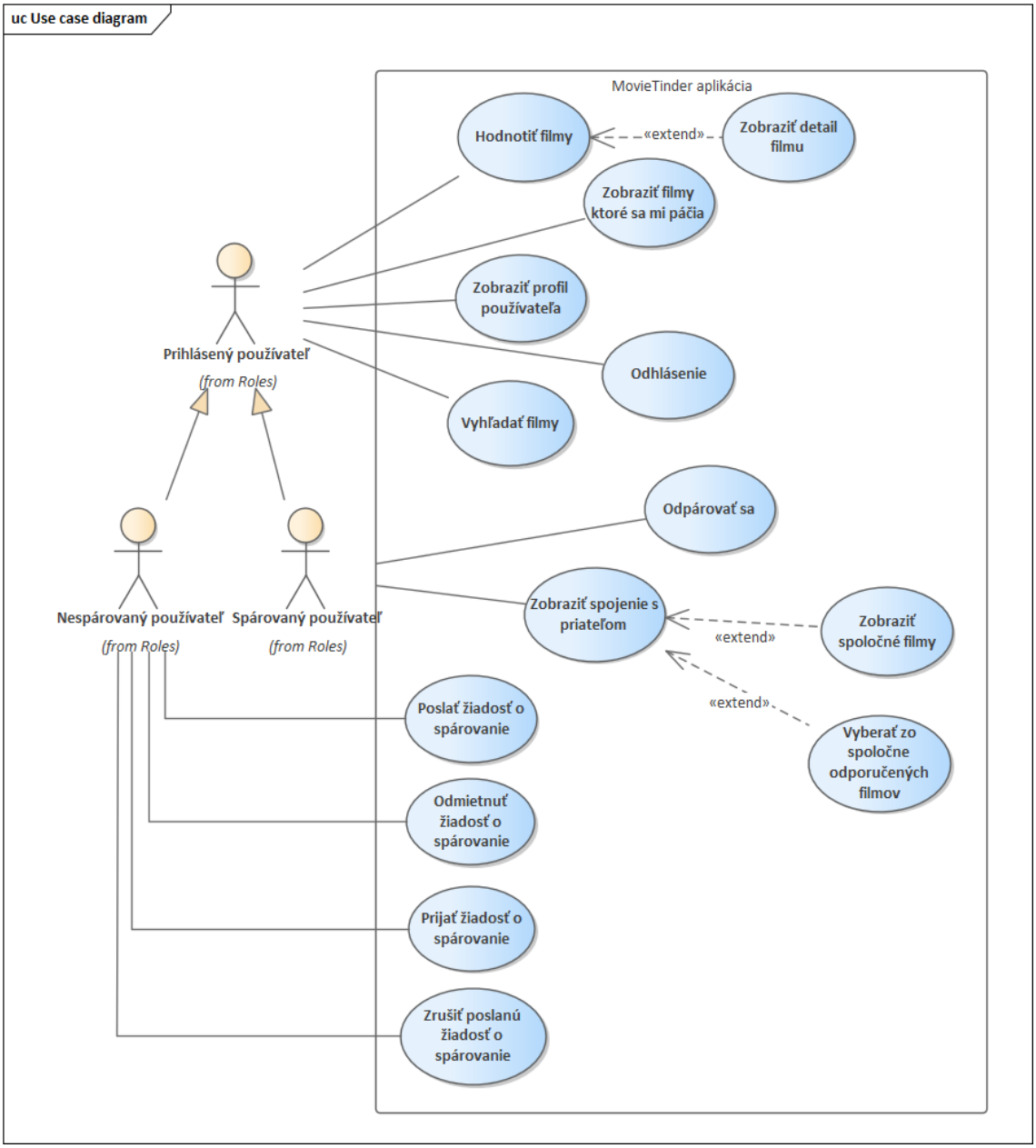
V ďalšej časti práce je návrh detailnejšie vizualizovaný pomocou vybraných UML diagramov.

2.3.1 Diagramy prípadov použitia



Obr. 8: Diagram prípadov použitia 1

Na diagrame 1 môžeme vidieť, že aplikácia rozlišuje 3 základné role a to neregistrovaný používateľ a neprihlásený (registrovaný) resp. prihlásený používateľ. Neregistrovaný používateľ sa vie zaregistrovať pomocou zvolenej prezývky, emailu a hesla. Následne ho aplikácia rovno prihlási pričom mu zobrazí krátky návod na oboznámenie sa s aplikáciou. Neprihlásený (registrovaný) používateľ sa vie prihlásiť pomocou emailu a hesla.

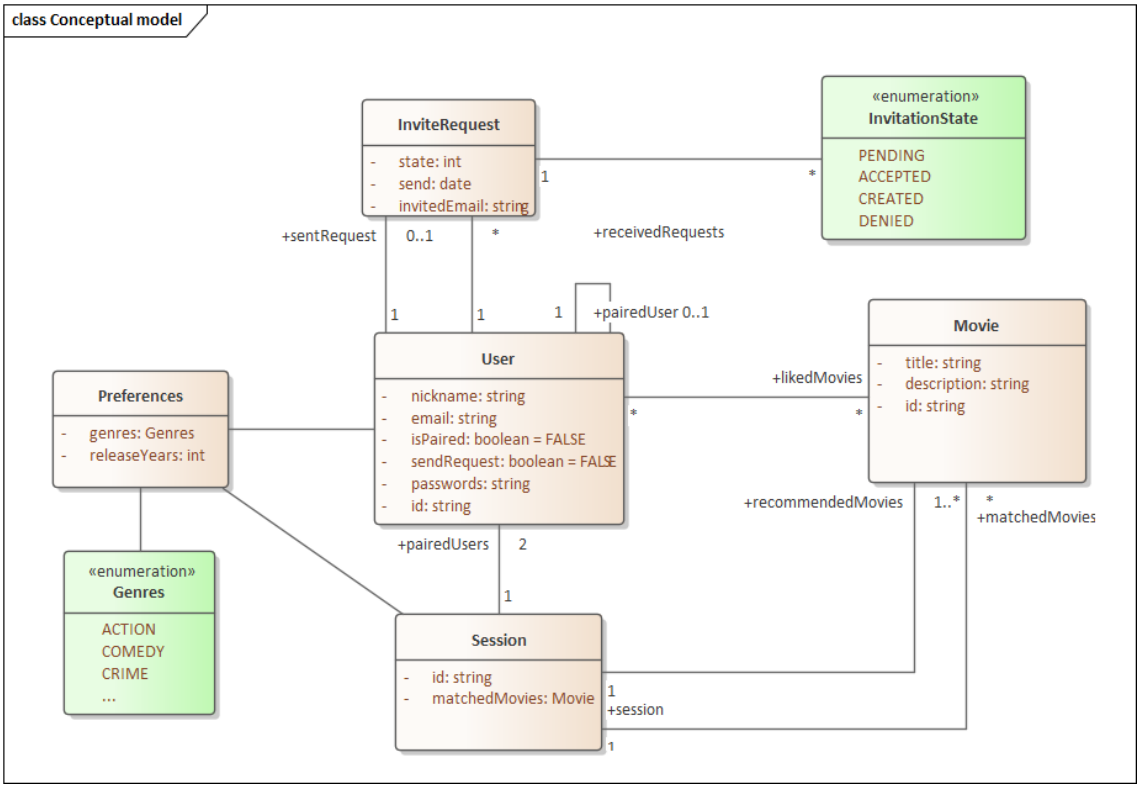


Obr. 9: Diagram prípadov použitia 2

V diagrame 2 máme detailnejší popis aplikácie po prihlásení používateľa. Prihlásený používateľ má možnosť hodnotiť filmy, prípadne si pozrieť detail aktuálne zobrazeného filmu. Tak isto si vie zobraziť filmy, ktoré sa mu páčili. Vie si tiež zobraziť svoj profil so základnými informáciami o jeho účte a odhlásiť sa z aplikácie. Po prihlásení môže používateľ nadobudnúť dve ďalšie podrole. Nespárovaný používateľ vie poslať žiadosť o spárovanie inému používateľovi, ktorú v prípade neobdržania odpovede vie aj zrušiť. Samozrejme vie aj prijať, prípadne odmietnuť žiadosť o spárovanie zo zoznamu žiadostí. Spárovaný používateľ si môže zobraziť aktuálne spojenie, hodnotiť v ňom filmy a pozrieť

si spoločné zhody.

2.3.2 Diagram tried



Obr. 10: UML diagram tried

Obrázok 10 zobrazuje diagram tried aplikácie. Trieda user vyjadruje používateľa, session spojenie medzi dvoma používateľmi. Jeden používateľ môže byť v jeden moment súčasťou iba jedného spojenia. Trieda Movie vyjadruje film a InviteRequest pozvánku na spojenie. Každý používateľ môže obdržať viacero pozvánok, avšak prijať vie iba jednu. Trieda preferences je dôležitým prvkom v systéme odporúčaní, keďže pomocou nej sa generujú jednotlivé odporúčania.

Význam väčšiny atribútov je z ich názvu jasný, preto spomenieme len tie, ktoré považujeme za potrebné objasniť. Pri triede user atribút isPaired vyjadruje, či daný používateľ je spárovaný s iným používateľom a atribút sendRequest, či daný používateľ poslal žiadosť o spárovanie inému používateľovi.

3 Implementačná časť

V tejto časti práce sa pozrieme na celý implementačný proces od úvodného nastavenia a inštalácie až po konkrétne časti implementácie aplikácie.

Záver

Conclusion is going to be where?
Here.

Zoznam použitej literatúry

1. RICCI, ROKACH, SHAPIRA and KANTOR. *Recommender Systems Handbook: Introduction to Recommender Systems*. First. Springer Science & Business Media, 2010. ISBN 978-0-387-85819-7.
2. *Cloud Firestore: Introduction*. Dostupné tiež z: `https://firebase.google.com/docs/firestore`.