

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-xxxx-xxxx

**MOVIE TINDER
BAKALÁRSKA PRÁCA**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-xxxx-xxxx

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 aplikácií. 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

Pod'akovanie

Obsah

Úvod	1
1 Teoretická časť	2
1.1 Odporúčacie systémy	2
1.1.1 Základné funkcie	2
1.1.2 Typy odporúčacích systémov	3
1.1.3 Collaborative filtering	4
1.1.4 Content based filtering	5
1.2 Mobilné aplikácie	6
1.2.1 Typy mobilných aplikácií	7
1.2.2 Prečo sme si vybrali React Native?	10
1.3 React Native	10
1.3.1 Princíp fungovania	11
1.3.2 Bridge	11
1.3.3 Virtual DOM (Document Object Model)	11
1.3.4 Základné komponenty	13
1.3.5 Props a state	14
1.3.6 Hooks	15
1.4 Expo alebo React Native CLI ?	15
1.4.1 Expo	15
1.4.2 React Native CLI	16
1.4.3 Odôvodnenie výberu	16
1.5 Node.js	17
1.5.1 Yarn	17
1.5.2 Axios	18
1.6 TMDb API	18
1.7 Firebase	18
1.7.1 Firebase authentication	18
1.7.2 Cloud firestore	19
2 Analytická časť	20
3 Návrhová časť	21
4 Implementačná časť	22

Záver	23
Zoznam použitej literatúry	24

Zoznam obrázkov a tabuliek

Obrázok 1	Collaborative filtering	5
Obrázok 2	Content based filtering	6
Obrázok 3	Ukážka fungovania React Native aplikácie	11
Obrázok 4	Virtual DOM vs. real DOM	12
Obrázok 5	Porovnanie blocking a non-blocking modelu	17
Obrázok 6	Znázornenie dátového modelu Cloud Firestore	19

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

1	Príklad class komponentu	13
2	Príklad function komponentu	14

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, doporučila film, alebo seriál, ktorý bude pre oboch čo najviac relevantný. Používatelia si potom nezávisle od seba môžu v navrhovaných filmoch pomocou jednoduchého swipeovania vyberať, či sa im daný film páči, alebo nie, pričom ak nastane medzi nimi zhoda a obaja označia ten istý film, aplikácia ich nato upozorní. Každý používateľ aplikácie bude mať svoju vlastnú filmotéku, teda zoznam filmov ktoré označil, že sa mu páčia, a zoznam filmov ktoré už videl. 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 alebo seriálu, pričom odpadá otravné prehľadávanie 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, rôzni 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 najobľúbenejších produktov. Personalizované odporúčania v ich najjednoduchšej 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 alebo IMDb majú vyvinuté svoje vlastné odporúčacie systémy, ktoré dlhodobo pomáhajú 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 používateľovi vybrať si položky, ktoré by bez odporúčania ťažšie našiel.
- **Zvýšiť spokojnosť používateľa** - dobre fungujúci odporúčací 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 zákazníka a bude sa rád vracieť.
- **Lepšie rozumieť tomu, čo zákazník chce** - pomocou zbierania rôznych používateľských preferencií a následného analyzovania, vedia firmy lepšie reagovať na trh, zmeny na ňom a prispôbovať svoj sortiment.

1.1.2 Typy odporúčacích systémov

Vo všeobecnosti, existujú odporúčacie techniky, ktoré nevyžadujú veľké množstvo vedomostí tj. používajú veľmi jednoduché základné dáta, ako napríklad používateľove hodnotenia položiek.

- **Položky** - sú objekty ktoré sú odporúčané. Môžu byť charakterizované ich zložitou a ich hodnotou. Hodnota môže 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 môžu byť štruktúrované viacerými spôsobmi a výber ktoré informácie treba modelovať systémom závisí od zvolenej techniky. Napríklad pri collaborative filteringu sa modelujú používatelia.
- **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 užitočné pre odporúčací algoritmus.

Najpopulárnejšou formou dát transakcií sú hodnotenia, ktoré systém zbiera. Tieto hodnotenia môžu 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 od do (1-5), slovné hodnotenia (veľmi súhlasím, súhlasím, neutrálny, nesúhlasím, veľmi neúhlasím) alebo 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 absencii presnejších informácií o preferenciách používateľa, populárna pieseň tj. položka, ktorá je obľúbená 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.

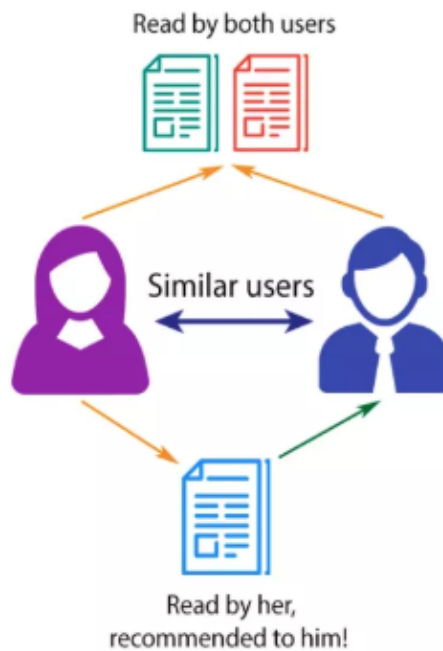
Pri odporúčacích systémoch existujú dve hlavné techniky návrhu algoritmu a to tzv. collaborative filtering a content based filtering. Poďme sa teda pozrieť na rôzne koncepty týchto dvoch techník.

1.1.3 Collaborative filtering

Collaborative filtering je metóda vytvárania automatických predpovedí o záujmoch používateľa zbieraním preferencií od väčšieho počtu používateľov. Základným predpokladom tejto metódy je, že ak používateľ A má rovnaký názor na produkt ako používateľ B, je vysoko pravdepodobné, že používateľ A bude mať na iný produkt ten istý názor ako používateľ B. Týmto postupom by systém vedel napríklad predpovedať, ktoré televízne šou ma daný používateľ rád, na základe jeho doterajších názorov (napr. vo forme likes/dislikes). Treba poznamenať, že tieto predpovede sú špecifické pre daného používateľa, ale vytvárajú sa na základe informácií zhromaždených od veľkého počtu iných používateľov.

Hlavnou výhodou tejto metódy je, že sa časom s pribúdajúcim počtom interakcií zlepšuje a stáva sa efektívnejšou. Avšak, vzhľadom nato, že berie do úvahy iba interakcie z minulosti, jej hlavným problémom je tzv. "cold start problem": znamená, že je takmer nemožné niečo odporučiť novému používateľovi, alebo odporúčať nový produkt používateľom, pokiaľ či už používateľ alebo produkt nemajú žiadne interakcie. Navyše mnoho používateľov a produktov má príliš málo interakcií nato, aby algoritmus s nimi vedel na začiatku efektívne pracovať. Riešenia tohto problému bývajú, že novým používateľom sa odporúčajú náhodné produkty a nové produkty sa odporúčajú náhodným používateľom (tzv. "random strategy"), odporúčanie populárnych produktov novým používateľom a odporúčanie nových produktov najviac aktívnym používateľom (tzv. "maximum expectation strategy"). Tak isto sa často používa, že v skorej fáze "života" používateľa alebo produktu sa použije iná metóda ako collaborative filtering.

Existujú dva prístupy k tejto metóde.



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

Obr. 1: Collaborative filtering

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

- **User-based collaborative filtering:**

Nájsť iných podobných používateľov a odporúčať produkty ktoré sa páčia im.

- **Item-based collaborative filtering:**

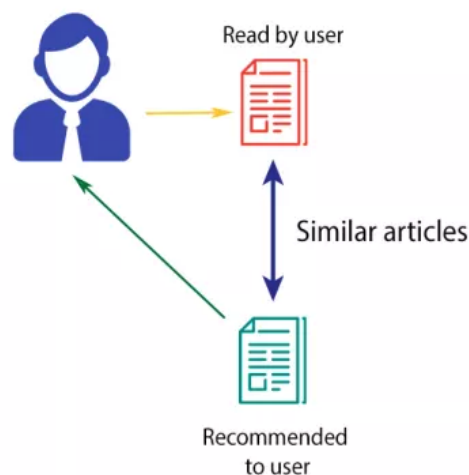
Odporúčať produkty ktoré používatelia označili, že sa im páčia, pričom niektoré iné produkty máte spoločne označené.

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.

1.1.4 Content based filtering

Táto metóda zahŕňa odporúčanie produktov na základe ich samotných vlastností. Odporúčania sú tu vytvárané na základe predchádzajúcich interakcií jednotlivého používateľa s produktami. Systém pracujúci touto metódou, sa snaží hľadať podobnosti medzi produktami, 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 si používateľ v internetovom obchode s knihami kúpi knihu ktorá patrí do kategórie ‘literatúra’, je logické mu odporučiť knihy v tej istej alebo podobnej kategórii. Tak isto odporučiť mu knihy z toho istého roku vydania, alebo od toho istého autora. Výhodou content-based filteringu je, že nepotrebujeme veľa interakcií z minulosti nato, aby navrhoval relevantné produkty. Oproti tomu nevýhodou je, že sa systém časom neučí z interakcií a tým pádom sa ani časom nezlepšuje, keďže sa striktne drží informácií ktoré má o danom používateľovi. Ľudia sú však nepredvídateľní a ich vkus sa časom mení, preto v tomto ohľade má collaborative filtering navrch.



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

Obr. 2: Content based filtering

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 počasi, 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 pri niektorých frameworkoch je ťažké určiť, do ktorej kategórie presne patria, keďže kombinujú vlastnosti viacerých z nich.

- **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 úložisku zariadenia.

Používané technológie: Na vývoj natívnych 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. Sú 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 funkcionálnosť a opravu chýb, mal by pri každej aktualizácii 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 prispôbujú 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 dispozícii plná funkcionálnosť, 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 princípe miešania prvkov natívnych a webových aplikácií. Jadro aplikácie je napísané 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 funkcionality 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 činitele v ktorých hybridné aplikácie predčia konkurenciu. Vďaka tomu, že je z veľkej časti použitý rovnaký zdrojový kód, nezávisle od mobilného operačného systému. Na vývoj je potrebných menej vývojárov (napr. nemusia byť zvlášť tímy vývojárov pre iOS a zvlášť pre Android).

Nevýhody: Najväčším mínusom hybridných aplikácií je výkon. Keďže sa načítavajú 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 udržať. 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 aplikácii.

- **Cross platform aplikácie** - sú špeciálnym typom aplikácií, ktoré sú veľmi podobné hybridným, pričom ale ponúkajú 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 technológii na vývoj našej 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

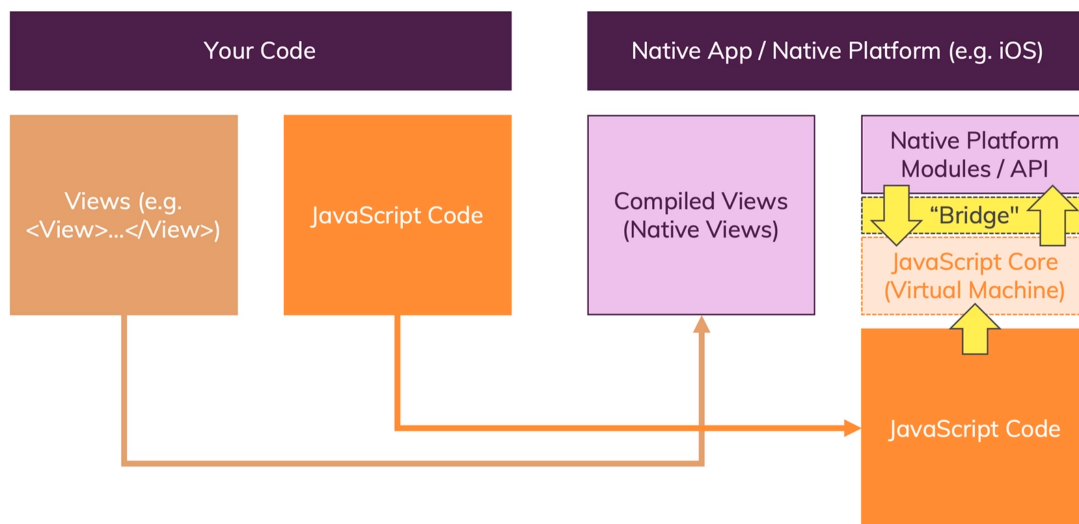
React Native je cross-platformový framework na vývoj mobilných aplikácií, ktorý je postavený na populárnom webovom frameworku React. Rovnako ako React, aj React Native je open-source projekt udržiavaný prevažne vývojármi Facebooku a Instagramu. Potom ako sa v roku 2012 Mark Zuckerberg vyjadril, že používanie veľkého množstva HTML5 v mobilnej aplikácii 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, softvérový 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 štylovanie 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 platfome. V prípade, ak je to potrebné, React Native umožňuje písať kód, logiku a štylovanie 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 zaujímavé, 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/>

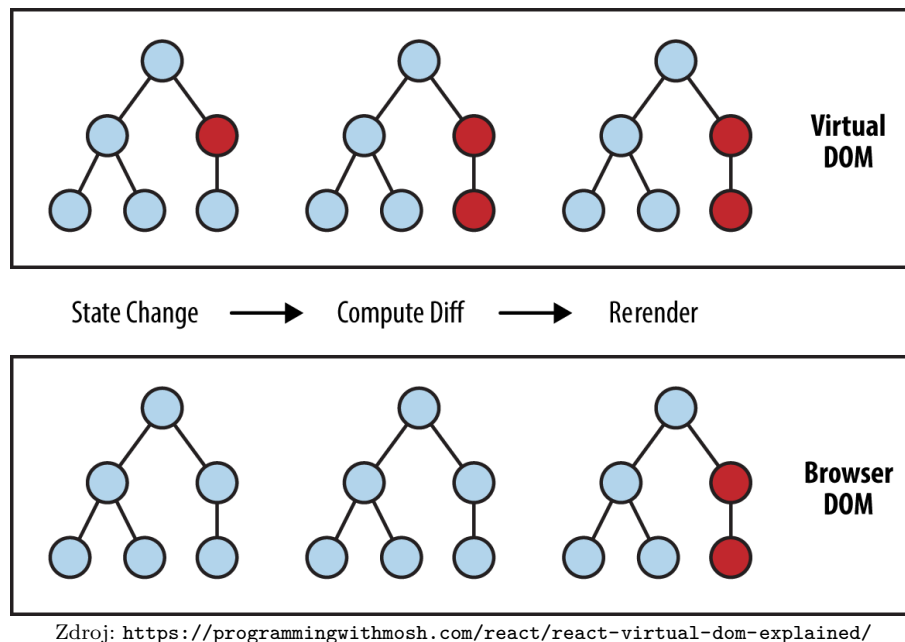
Obr. 3: Ukážka fungovania React Native aplikácie

1.3.3 Virtual DOM (Document Object Model)

Ďalším dôležitým konceptom 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 dátový strom, vďaka čomu je každá zmena rýchla. Avšak po tejto zmene, zmenené elementy a ich potomkovia musia byť nanovo vyrenderované aby nastal aj priamo update UI danej aplikácie. Proces renderovania je to, čo spôsobuje nezanedbateľné spomalenie výkonu, ktoré je navyše priamo úmerné so zväčšujúcim sa počtom UI komponentov, ktoré treba nanovo vyrenderovať.

Tu prichádza na scénu Virtual DOM a dosahuje podstatne lepšie výkonnostné 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 ktoréhokolvek elementu, vytvorí sa nový virtual DOM, prebehne proces porovnania (nazývaný "diffing") aktuálnej verzie a prechádzajúcej verzie virtual 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.



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 prijímať vstupy (nazývané “props”), s ktorými potom umožňujú pracovať vo vnútri komponentu a vracajú elementy ktoré popisujú čo sa má zobrazíť 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 komponentu. Vo výpise 1 máme uvedený 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 štandardom. 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 kliknutí na tlačidlo alebo zaškrtnutí checkboxu. Napríklad pri vyplňaní 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 detských 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 funkcionálnosť.

1.4 Expo alebo React Native CLI ?

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

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.
- Uvoľňuje OTA updates.

- Vstavaný prístup k natívnym APIs.

Nevýhody

- Ďalšia vrstva abstrakcie.
- Neumožňuje zasahovať do natívneho kódu.
- Niesu k dispozícii 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.

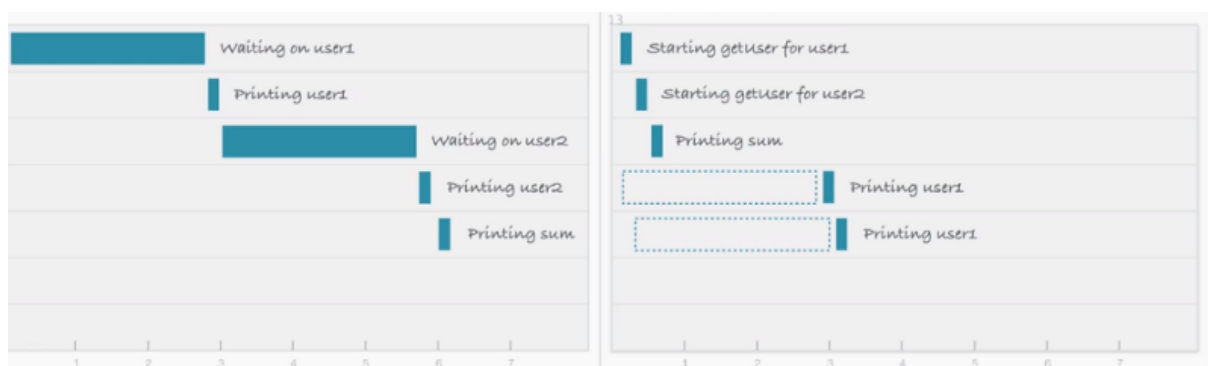
1.4.3 Odôvodnenie výberu

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 funkcionality 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 znamená input/output, čo 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 môžeme 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 funkcionálnosť 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, ktorý podporuje aj ES6 promise API.

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 priniesla 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.

1.7 Firebase

Firebase je platforma vyvinutá spoločnosťou Google, ktorá pomáha vytvárať, zlepšovať a rozširovať funkcionality 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

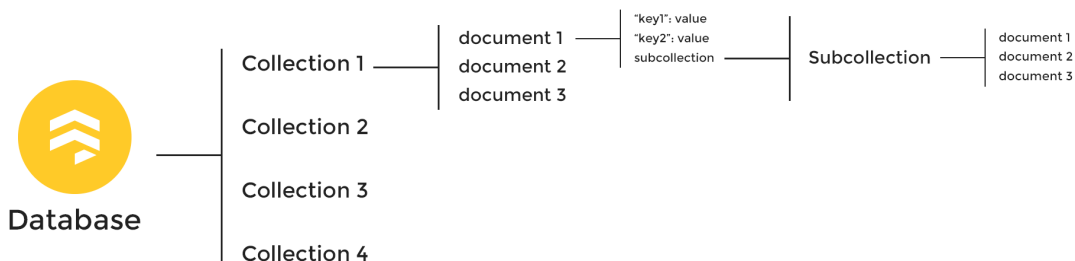
Poskytuje backendové služby, jednoducho použiteľné SDK a predpripravené UI knižnice na autentifikáciu používateľov v aplikácii. Podporuje autentifikáciu pomocou hesla, telefónneho čísla, Googlu, Facebooku a Twitteru. Je úzko integrovaná s ostatnými službami Firebase a využíva štandardy ako OAuth 2.0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

1.7.2 Cloud firestore

Cloud Firestore je škálovateľná NoSQL cloudová databáza, pre vývoj mobilných a webových aplikácií. Udržiava 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 (viď. obrázok 6), sa dáta ukladajú do tzv. dokumentov, ktoré obsahujú polia (fields) namapované na konkrétne hodnoty. Tieto dokumenty sú ukladané do kolekcií, ktoré slúžia ako kontajner 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 obsahujú dokumenty, ktoré obsahujú kolekcie a tie obsahujú ďalšie dokumenty atď.). [2]

Firebase 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 a tiež 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 část

3 Návrhová část

4 Implementačná časť

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>.