Apache Spark: Izrada web aplikacije za pružanje personaliziranih preporuka filmova

Senković, Ivan

Sveučilišni diplomski studij Informatika

Ivan Senković

Apache Spark: Izrada web aplikacije za pružanje personaliziranih preporuka filmova

Diplomski rad

Mentor: doc. dr. sc. Danijela Jakšić

Rijeka, srpanj 2024.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Rijeka, 14. lipanj 2024.

Zadatak za diplomski rad

Pristupnik/ica: Ivan Senković

Naziv diplomskog rada: Apache Spark: Izrada web aplikacije za pružanje personaliziranih preporuka filmova

Naziv diplomskog rada na eng. jeziku: Apache Spark: Izrada web aplikacije za pružanje personaliziranih preporuka filmova

Sadržaj zadatka: Cilj je diplomskog rada izgraditi web aplikaciju s filmovima koja koristi tehnike strojnog učenja za pružanje personaliziranih preporuka filmova. Aplikacija treba omogućiti korisnicima pregled naslova, opisa, ocjena filmova te pristup detaljnim stranicama svakog filma gdje mogu vidjeti dodatne detalje i ostaviti vlastitu ocjenu. Za razvoj frontend-a aplikacije potrebno je koristiti tehnologije koje omogućuju kreiranje interaktivnih i korisničkih sučelja. Backend treba biti razvijen koristeći tehnologije i okvire koji omogućavaju brzu izgradnju sigurnih i održivih web aplikacija. Sustav za upravljanje bazom podataka treba podržavati sustave za preporučivanje, s obzirom da je ključni aspekt ove aplikacije implementacija strojnog učenja za generiranje personaliziranih preporuka filmova. U tu svrhu, u radu je potrebno predstaviti i koristiti Apache Spark, napredni sustav za obradu velikih skupova podataka koji omogućava brzu analizu i obradu podataka potrebnih za treniranje modela strojnog učenja. Cjelokupan proces izrade praktičnog rada potrebno je opisati kroz prikaz koda, ekrana i funkcionalnosti. Na kraju rada potrebno je dati kritički osvrt te iznijeti prigodne zaključke.

Mentor/ica:
Doc. dr. sc. Danijela Jakšić

Komentor/ica:
dr. sc. Kristian Stančin

Voditeljica za diplomske radove:
Doc. dr. sc. Lucia Načinović Prskalo

Zadatak preuzet: 14. lipanj 2024

(potpis pristupnika/ce)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

U diplomskom radu opisan je razvoj aplikacije koja služi za preporuku filmova korisnicima. Web aplikacija nudi preporuku filmova ovisno o odabranom prethodno ocjenjenom filmu. Razvijena je pomoću web framwork-a Django koji je zasnovan na Python programskom jeziku. Za prikaz korisničko sučelje i prikaz svih podataka korišteni su programski jezici za razvijanje web aplikacija, a to su HTML, CSS te Javascript. Za spremanje podataka i njihovo organiziranje koristi se PostgreSQL. Apache Spark se koristi za pripremu podataka prije strojnog učenja kako bi podaci bili potpuni, ispravni te u pravilnom formatu za odabranu metodu strojnog učenja. Strojno učenje odrađeno je putem Python programskog jezika za preporuku filmova, a koriste se metode „Content-Based filtering“ i „Collaborative filtering“. Podaci u filmovima prikupljaju se putem pozivanja API-ja stranice „The Movide DB“ koja ima razan izbor filmova sličan onome na „IMDB“ i nudi ocjenjivanje filmova. Dolaskom na web stranicu korisnika dočeka naslovna stranica gdje može pogledati razne trenutno popularne filmove gdje na prijavljen korisnik ima mogućnost samo pregled informacija o filmovima i njihov prosjek ocjena. Korisnik ima mogućnost prijave ili kreiranja novog korisničkog profila te kad se kreira novi profil, ima mogućnost ocjenjivanja filmova kako bi bilo moguće preporučiti filmove. Za izradu aplikacije korišten je API i IDE „Visual Studio Code“ zbog toga što je aplikacija lokalna pa nije potrebno korištenje ostalih servisa kao što je na primjer Amazon AWS za pohranu podataka i pokretanje Python skripta na dnevnoj bazi, što bi bilo potrebno da je aplikacija na uživo produkciji.

Ključne riječi: diplomski rad; Apache Spark; Django; Python; Preporuka; API; PostgreSQL; HTML; CSS; Javascript; Content-Based filtering; Collaborative filtering.

SADRŽAJ

1. Uvod ... 2
2. Korištene tehnologije ... 4
2.1. The Movie Database(TMDB) ... 6
2.2. PostgreSQL ... 8
2.3. Apache spark ... 15
2.4. Django ... 17
1.1.1. Python virtual environment ... 20
1.1.2. Settings.py ... 21
1.1.3. Urls.py ... 27
1.1.4. Admin.py ... 28
1.1.5. Views.py ... 29
1.1.6. utils.py ... 39
1.1.7. fetch_popular_movies.py ... 42
1.1.8. fetch_top_rated_movies.py ... 44
1.1.9. content_based.py ... 46
1.1.10. recommendations.py ... 51
1.1.11. fake_data.py ... 57
3. Korištenje aplikacije ... 61
4. Zaključak ... 70
5. Reference ... 71
6. Literatura ... 73
7. Popis slika ... 74
8. Popis tablica ... 75

1. Uvod

Apache Spark je open-source okvir za obradu velikih podataka koji omogućava brzu i jednostavnu obradu podataka u distribuiranom računalnom okruženju. Razvijen je da proširi model MapReduce kako bi podržao različite vrste računalnih operacija, uključujući interaktivne upite i obradu strujnih podataka. Spark koristi fleksibilnu i skalabilnu arhitekturu koja omogućava korisnicima da obrade velike količine podataka brzo i učinkovito. Spark je brzo usvojen od strane mnogih industrija zbog svoje fleksibilnosti i performansi. Koristi se u raznim aplikacijama, uključujući analizu društvenih medija, prediktivnu analitiku, optimizaciju marketinških kampanja i obradu znanstvenih podataka. [1]

Sustavi preporuka(engl. Recommender Systems) su postali ključni alati kod digitalnih informacija, pomažu korisnicima da navigiraju kroz ogromne količine podataka i pronalaze relevantne stavke. Ovi sustavi su posebno važni za e-trgovinu, društvene mreže i platforme za streaming medija, gdje personalizirane preporuke mogu značajno poboljšati korisničko iskustvo i povećati angažman korisnika. Sustavi preporuka se općenito dijele na tri glavne kategorije: kolaborativno filtriranje, filtriranje temeljeno na sadržaju i hibridne metode.

Kolaborativno filtriranje (Collaborative Filtering) se dijeli na 2 djela: Temeljeno na korisnicima (User-based) gdje se preporuke generiraju na temelju sličnosti između korisnika. Ako korisnik A ima slične preferencije kao korisnik B, stavke koje su se svidjele korisniku B preporučit će se korisniku A. Temeljeno na stavkama, ovaj pristup identificira sličnost između stavki. Ako je korisnik ocijenio stavku X visokom ocjenom, sustav će preporučiti slične stavke kao što je X drugim korisnicima. [2]

Filtriranje Temeljeno na Sadržaju (Content-based Filtering) je pristup koji analizira sadržaj stavki i preferencije korisnika kako bi preporučio slične stavke koje dijele karakteristike s onima koje je korisnik prethodno ocijenio pozitivno. Na primjer, ako korisnik voli filmove određenog žanra, preporučit će mu se više filmova iz tog žanra. [2]

Također postoje Hibridni Sustavi (Hybrid Systems) koji kombiniraju više tehnika preporuke kako bi poboljšali performanse i točnost preporuka. Hibridne metode mogu uključivati kombinaciju kolaborativnog filtriranja i filtriranja temeljenog na sadržaju, koristeći najbolje karakteristike oba pristupa. Ovaj pristup je jednim dijelom prikazan u ovome radu pošto se koriste dva sustava za jednu aplikaciju. [3]

Razvoj web i mobilnih aplikacija kao sustava za preporučivanje brzo napreduje, zbog potrebe za personaliziranim korisničkim iskustvima. Ovi sustavi koriste napredne algoritme kako bi korisnicima preporučili relevantne proizvode, usluge ili sadržaje, čime se poboljšava angažman korisnika i povećava zadovoljstvo. Amazon koristi napredne preporučivačke algoritme kako bi personalizirao korisničko iskustvo, predlažući proizvode na temelju prethodnih kupovina i pretraga korisnika. Platforme poput Netflixa i Spotifya koriste sustave preporuka za personalizaciju sadržaja, čime povećavaju vrijeme provedeno na platformi i zadovoljstvo korisnika. Facebook i LinkedIn implementiraju preporuke kako bi predložili nove kontakte i relevantan sadržaj, poboljšavajući angažman korisnika. Teškoća u preporučivanju stavki novim korisnicima ili novih stavki postojećim korisnicima. Rješenja uključuju

prikupljanje dodatnih informacija o korisnicima ili korištenje hibridnih modela koji kombiniraju različite izvore podataka. Rukovanje velikim količinama podataka zahtijeva efikasne algoritme i infrastrukturu. Tehnologije poput Hadoop i Spark omogućuju obradu velikih podataka kroz distribuirane računalne resurse. [4]

Budući razvoj će se fokusirati na daljnju integraciju AI tehnologija, poput dubokog učenja i grafičkih neuronskih mreža, kako bi se poboljšala preciznost i personalizacija preporuka, te na rješavanje izazova vezanih uz privatnost i etiku korištenja podataka. [5]

Web aplikacije se svakim danom sve više koriste i dostupne su svim korisnicima neovisno o uređaju gdje kod mobilnih aplikacija ima više vrsta sustava i samim time nije najbolji način prikupljanja velikog broja korisnika. Aplikacija je izrađena tako da pruža preporuku filmova samo korisnicima koji su prijavljeni, kako bi to motiviralo posjetitelje stranice da izrade korisnički račun. Sve podatke o filmovima koje aplikacija sadrži, dio su API poziva gdje ulaskom na naslovnu stranicu u tom trenutku zove API, a za preporuku i ostale funkcije podaci se vuku iz baze koji su prethodno bili spremljeni putem API kako bi se izbjeglo ponovno pozivanje.

Početni cilj diplomskog rada bio je izrada aplikacije koja nudi općenitu preporuku filmova, ali tijekom razvoja aplikacija se proširila i samim time njena funkcionalnost. Aplikacija je razvijena da služi kao preporuka i ocjenjivanje filmova, ali uz API ima mogućnost razviti dodatno preporuku televizijskih serija ili odvojiti u zasebnu aplikaciju s vlastitim sučeljem kako bi pružalo korisnicima raznovrsnost.

Aplikacija je izrađena da korisnik dolaskom na stranicu ima izbor najpopularnijih filmove, što ne mora bit isključivo najpopularniji u Američkim državama nego i cijelom svijetu. Napravljeno je taj način tako da odmah privuče korisnike da pogledaju svoje najdraže filmove, imaju pregled filmova kojima se raduju za pogledat ili čak istraže još ne viđene filmove. Svaki film ima svoju „karticu“ gdje je moguće vidjet najbitnije informacije, a to su: poster ili slika, naslov, prosječna ocjena i broj ocjena gdje su ove ocjene povučene putem API poziva pošto je moguće da korisnici još nisu ocijenili nove popularne filmove. Prijavljenom korisniku je omogućeno ocjenjivanje ne samo popularnih filmove nego i najbolje ocijenjenih tako što vizualnih „zvijezdama“ mogu odabrati ocjenu od 1 do 10 te pritiskom na gumb spremiti svoju ocjenu i čak kratki osvrt.

U sljedećim cjelinama opisan je detaljan proces rada i razvoja aplikacije, od samog dolaska na stranicu pa sve to prikaza preporuka filmova. Opisan je detaljan cilj i razlog uporabe svake biblioteke, programskih jezika i ostalih softvera te zašto je korišten određeni pristup umjesto nekog drugog. Sve je zajedno povezano s lokalnom bazom podataka i više tablica koje služe spremanje podataka o filmovima, ocjenama i osnovnih podataka o prijavljenim korisnicima. Na kraju rada opisan je proces prikazivanja preporučenih filmova prijavljenom korisniku za film koji je odabrao ovisno o ostalim filmovima iz baze podataka. U samom zaključku rada rezimiran je cijeli rad i mišljenje o odabiru ovakve teme, što se može dodatno razviti, mogućnosti objavljanja aplikacije na produkciju gdje bi bila dostupna svima, popis literature, slike korištene i dijelovi kodova.

2. Korištene tehnologije

Razvoj web i mobilnih aplikacija koje koriste sustave za preporučivanje postao je ključan za pružanje personaliziranog korisničkog iskustva. Ove aplikacije koriste različite tehnologije i frameworks kako bi analizirale korisničke preferencije i predložile relevantne proizvode, usluge ili sadržaje.

Prva i najvažnija tehnologija, to jest „framework“ je Django koji je zaslužan za izgradnju i postavljanje aplikacije. Django aplikacije se grade korištenjem Python programskog jezika kao backend što sadrži sve funkcije aplikacije koje korisnik ne vidi nego se u pozadini odrađuju. Frontend dio aplikacije sastoji se od HTML koji služi za postavljanje elemenata na stranicu, kako bi sučelje izgledalo bolje, koristite CSS za bolji dizajn i raspored elemenata te na kraju JavaScript koji povezuje backend-frontend dijelove aplikacije i također služi kao dodatne akcije koje su omogućene korisniku kako bi olakšalo korištenje i izgled aplikacije.

Kako bi se moglo razvijati Django aplikaciju potrebno je koristiti razne biblioteke koje framework pruža, a tako i biblioteke Python-a. najbitnije biblioteke Djanga su one kojima se aplikacija povezuje u cjelinu kao što su urls, admin, models, user i slično. Ostale biblioteke dolaze s Python strane, a neke od bitnijih su numpy za korištenje matrica svakavih dimenzija, pandas za manipulaciju, analizu pa čak i organiziranje podataka u tablice, pyspark koji je bitan za rad zbog pripreme i obrade podataka, requests kako bi se mogli povezati na API i json kako bi se omogućilo strukturirat i obradit podatke u pravilnom obliku. Ostale biblioteke su bitne, ali se ne koriste često u aplikaciji pošto služe za formatiranje, uporaba metoda iz drugih datoteka i slično.

Django koristi moćan ORM sustav za mapiranje modela na relacijske baze podataka. Ovaj sustav omogućuje razvojni inženjeri da rade s bazama podataka koristeći Python klase umjesto SQL upita. Sadrži automatski generirano administratorsko sučelje koje omogućuje upravljanje sadržajem web stranice bez potrebe za dodatnim razvojem. Koristi fleksibilan i moćan sustav za URL raspoređivanje koji omogućuje čiste i intuitivne URL obrasce. Django nudi ugrađene zaštite protiv mnogih sigurnosnih prijetnji kao što su SQL injekcije, cross-site scripting (XSS), cross-site request forgery (CSRF) i clickjacking. Dizajniran je za skalabilnost i može podržati visoko prometne web stranice. Njegova arhitektura omogućuje jednostavno proširenje i optimizaciju performansi. [6]

Za Django aplikacije predefinirana baza potaka je SQLite koja je dobra za klasičnim radom nad bazom i kad nema previše kompleksnih podataka, ali zbog daljnjeg razvoja aplikacije gdje bi se moglo dodatno koristiti mnogo više raznih vrsta podataka, potreba za brzim spremanjem i dohvaćanjem podataka, koristi se PostgreSQL. Oba sustava za upravljanjem bazom podataka koriste SQL strukturu to jest relacijski model gdje se podaci pohranjuju u tablicama s unaprijed definiranim shemama (kolone i redovi). Shema je strogo definirana i podaci moraju biti strukturirani prema toj shemi. Svaka tablica mora biti definirana unaprijed sa specifičnim kolonama i tipovima podataka. Ove kolone i tipovi podataka moraju se povezati i odgovarati modelima iz Django aplikacije koji se moraju prilagoditi tako da kasnije nije potrebno previše pretvarati podatke u različite formate. Jedan od razloga zašto se

koristi PostgreSQL za aplikaciju je zbog programa PgAdmin koji služi kao sučelje za upravljanje bazama podataka i tablicama. Ovaj program omogućuje pokretanje upita, ažuriranje, brisanja, stvaranja tablica i slično, ali također je moguće izvesti podatke u tablicama kako bi se moglo koristiti na drugim uređajima ili objaviti izvoz tablice na Internet servise kao što je Amazon AWS.

U Django frameworku, HTML se koristi za izradu korisničkih sučelja koja prikazuju podatke koje backend aplikacija obrađuje. Django koristi šablonski jezik (Django Template Language - DTL) za dinamiziranje HTML-a i omogućuje integraciju s backend logikom. Django šablone omogućuju umetanje dinamičkog sadržaja u HTML. Ovaj proces se ostvaruje pomoću posebnih oznaka i filtera unutar HTML datoteka. HTML u kombinaciji s Django šablonama omogućuje stvaranje dinamičnih i interaktivnih web stranica koje mogu učinkovito komunicirati s backend-om i pružiti korisnicima personalizirano iskustvo. Primjer je prikazan na isječku. [7]

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Article List</title>
</head>
<body>
    <h1>Articles</h1>
    <ul>
        {% for article in articles %}
            <li>{{ article.title }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

JavaScript je skriptni jezik koji se koristi za dodavanje interaktivnosti i dinamičnosti web stranicama. JavaScript omogućava manipulaciju DOM-om (Document Object Model), rukovanje događajima (events), validaciju podataka na strani klijenta, i asinkrono dohvaćanje podataka pomoću AJAX-a (Asynchronous JavaScript and XML). JavaScript se koristi u Django aplikacijama za dodavanje interaktivnosti i poboljšanje korisničkog iskustva. JavaScript datoteke se mogu dodati u Django šablone na sličan način kao i CSS datoteke, korištenjem <script> taga unutar HTML dokumenta. Primjer je prikazan na isječku. [8]

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Django Page</title>
    <script src="{% static 'js/myscript.js' %}"></script>
</head>
<body>
    <h1>Hello, Django!</h1>
</body>
</html>
```

API (Application Programming Interface) je skup definicija i protokola koji omogućavaju različitim softverskim komponentama da komuniciraju međusobno. API-ji omogućuju aplikacijama pristup funkcionalnostima ili podacima drugih aplikacija, servisa ili

platformi. U Django frameworku, API-jevi se koriste za dohvaćanje podataka iz vanjskih izvora, slanje podataka drugim aplikacijama, ili za izgradnju vlastitih API-ja koje druge aplikacije mogu koristiti. Tehnologija ili vrsta servisa koja je najbitnija u ovome radu je API poziv na stranicu The Movie Database(TMDB). Ovaj API poziv je besplatan za osobne svrhe koje nije komercijalne te je to bio razlog odabira pošto se može testirati više puta na dan i ima veliki izbor podataka, ne samo filmova, nego i televizijskih serija. [9]

2.1. The Movie Database(TMDB)

API (engl. Application Programming Interface) je skup definicija i protokola koji omogućuju različitim softverskim aplikacijama da komuniciraju međusobno. API definira način na koji se komponente softvera trebaju međusobno povezivati i komunicirati. Omogućuje aplikacijama da razmjenjuju podatke i funkcionalnosti bez potrebe da dijele svoj izvorni kod. API djeluje kao posrednik koji prihvaća zahtjeve od klijenta (npr. aplikacije) i prenosi ih serveru, a zatim vraća odgovor natrag klijentu, u ovom slučaju aplikacija za preporuku filmova. Postoje više vrsta API-ja, kao na primjer biblioteka ili framework API koja pruža funkcionalnosti koje mogu biti ugrađene unutar programa, operativni sustav API koji Omogućuje aplikacijama pristup funkcionalnostima operativnog sustava i web API koji se koristi za ovaj rad. API obično koristi standardizirane formate podataka, kao što su JSON ili XML, za prijenos informacija. Jedna od prednosti korištenja API je to što omogućuje različitim aplikacijama, napisanim u različitim programskim jezicima ili na različitim platformama, da međusobno komuniciraju. Također jedna od prednosti je to što omogućuje fleksibilnost i skalabilnost, na primjer nije potrebno da se povuku sve podaci kao što bi bilo na upita baze podataka nego već ima predefinirane krajnje točke(engl. endpoint) putem kojeg je moguće dobiti jedan podatak umjesto cijeli skup podataka koji se neće koristiti. [9]

The Movie Database(TMDB) [10] [11] je besplatan API servis koji pruža vlastite podatke u filmovima, televizijskim serijama i glumica, a također pružaju slike za koje se nalaze na njihovim serverima i nije potrebno preuzeti kako bi se prikazalo pošto imaju svoju web adresu. Prvo je potrebno registrirati se na TMDB web stranici te nakon registracije, moguće kreirati novi projekt i dobiti API ključ koji je potreban za autentifikaciju pri svakom API pozivu. API ključ (API key) je jedinstveni identifikator koji se koristi za autentifikaciju korisnika ili aplikacije koja pristupa API-ju. API ključevi su obično nizovi znakova koji su specifični za korisnika ili projekt, a služe kao sredstvo kontrole pristupa i praćenja upotrebe API-ja.

Vrste zahtjeva (Requests) koje je moguće raditi putem TMDB su:

- GET zahtjevi: koriste se za dohvaćanje podataka iz TMDB baze.
    - Primjer: GET /movie/{movie_id} za dohvaćanje detalja o određenom filmu.
- POST zahtjevi: koriste se za slanje podataka (obično za korisničke radnje kao što je ocjenjivanje filmova).

- DELETE zahtjevi: koriste se za uklanjanje određenih podataka ili resursa iz baze podataka.
    - Primjer: uklanjanje filma ili TV emisije s vašeg prilagođenog popisa

Konstrukcija URL-a za API poziv uključuje osnovni URL za API pozive je https://api.themoviedb.org/3 gdje broj 3 služi kao id filma, za dodavanje API ključa koristi se parametar „api_key“. Primjer URL-a:

https://api.themoviedb.org/3/movie/157336?api_key=API_KEY.

Neki od primjera za korištenje različitih endpoint-ova:

- Filmovi: https://api.themoviedb.org/3/movie/{movie_id}
- Pretragu filmova: https://api.themoviedb.org/3/search/movie?query={query}
- TV emisije: https://api.themoviedb.org/3/tv/{tv_id}
- Glumci: https://api.themoviedb.org/3/person/{person_id}
- Žanrovi: https://api.themoviedb.org/3/genre/movie/list

Podaci se redovito ažuriraju, što osigurava da korisnici uvijek imaju pristup najnovijim informacijama. Osnovni pristup TMDB API-ju je besplatan, što ga čini pristupačnim za razvojne projekte svih veličina. TMDB API je široko korišten u industriji, što znači da postoji mnogo resursa, vodiča i podrške dostupnih online. Neki od primjera korištenja API-ja su streaming servisi, aplikacije za preporuke filmova, filmske baze podataka i vođenje povijesti o gledanju filmova.

Za ovaj rad potrebno je koristiti samo GET pozive pošto je cilj izrade vlastite aplikacije koja se ne oslanja na podatke ostalih korisnika koji korite TMDB. POST se ne koristi jer svako spremanje je odrađeno na lokalnoj strani u osobnu bazu podataka.

```
import requests

api_key = 'YOUR_API_KEY'
movie_id = 157336
url = f'https://api.themoviedb.org/3/movie/{movie_id}?api_key={api_key}'

response = requests.get(url)
data = response.json()

print(data)
```

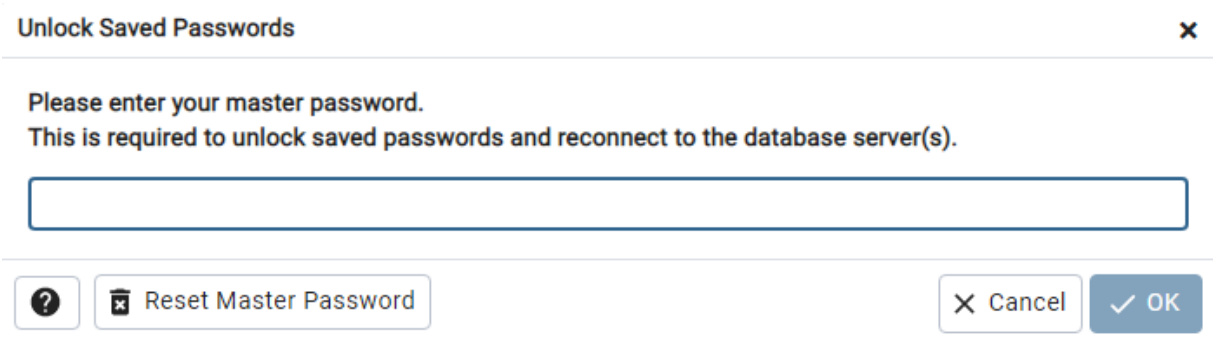

```
{
  "adult": false,
  "backdrop_path": "/path_to_backdrop.jpg",
  "belongs_to_collection": null,
  "budget": 63000000,
  "genres": [
    {
      "id": 18,
      "name": "Drama"
    }
  ],
  "homepage": "http://example.com",
  "id": 550,
```

```
  "imdb_id": "tt0137523",
  "original_language": "en",
  "original_title": "Fight Club",
  "overview": "A ticking-time-bomb insomniac...",
  "popularity": 22.446,
  "poster_path": "/path_to_poster.jpg",
  "release_date": "1999-10-15",
  "revenue": 100853753,
  "runtime": 139,
  "status": "Released",
  "tagline": "Mischief. Mayhem. Soap.",
  "title": "Fight Club",
  "video": false,
  "vote_average": 8.4,
  "vote_count": 3439
}
```


2.2. PostgreSQL

PostgreSQL [12] [13] je napredni, open-source relacijski sustav za upravljanje bazama podataka (RDBMS) koji koristi i proširuje SQL standard. Razvijen je s naglaskom na skalabilnost, sigurnost i podršku za složene podatkovne tipove. PostgreSQL je poznat po svojoj stabilnosti, robusnosti i bogatom skupu značajki. PostgreSQL se može instalirati na različite operativne sustave (Windows, macOS, Linux). Nakon instalacije, potrebno je konfigurirati bazu podataka, kreirati korisnike i dodijeliti im prava, to je moguće učiniti putem programa PgAdmin. PostgreSQL je idealan za aplikacije koje zahtijevaju visoku pouzdanost, skalabilnost i podršku za složene upite. Kada je potrebno da više korisnika pristupa bazi podataka istovremeno, PostgreSQL nudi bolju podršku za konkurentnost nego SQLite koji je zadani sustav kod Django aplikacije. SQLite: Ima ograničenja u pogledu višekorisničkog pristupa, što može dovesti do problema pri višestrukim istovremenim pristupima bazi podataka. PostgreSQL također nudi napredne sigurnosne značajke kao što su enkripcija podataka, kontrola pristupa, sigurnosne kopije i oporavak od grešaka. Bolje podržava složene migracije shema i nadogradnje baza podataka, što je ključno za dugoročne projekte koji se šire s vremenom.

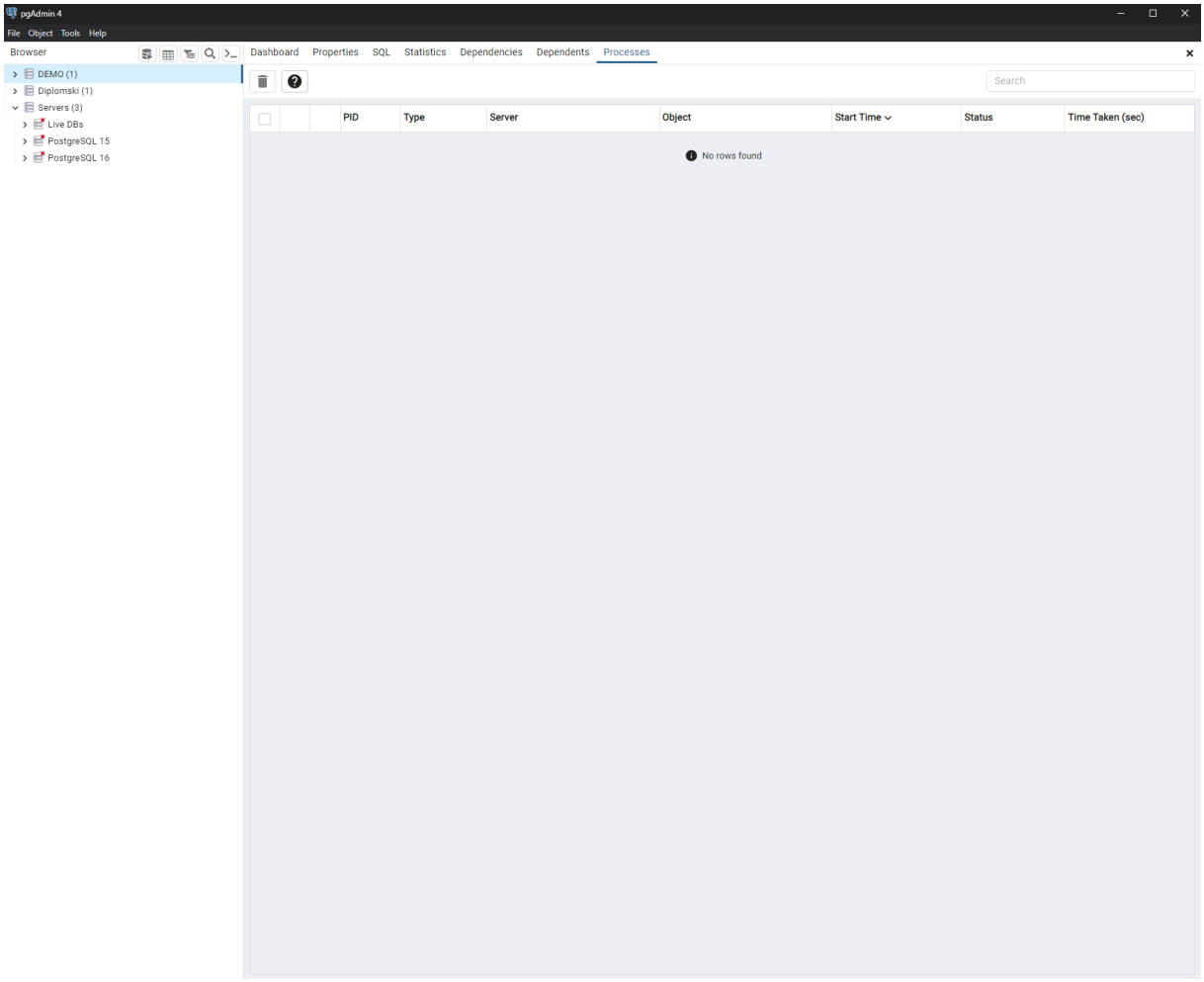
Za ovaj rad PgAdmin [14] je pomogao za jednostavan i vizualno prihvatljiv prikaz svih podataka koji su spremljeni u bazu. Svakim ulaskom u program potrebno se prijaviti s lozinkom koju se postavilo pri instalaciji(vidljivo na slici 1). Lozinka može biti kompleksna ako računalo i profil koriste različite osobe ili je javno dostupno, ali općenito ako je na osobnom računalo nije potrebno kako bi prijava prošla brže i bez poteškoća zaborava lozinke.



Slika 1. PgAdmin prijava

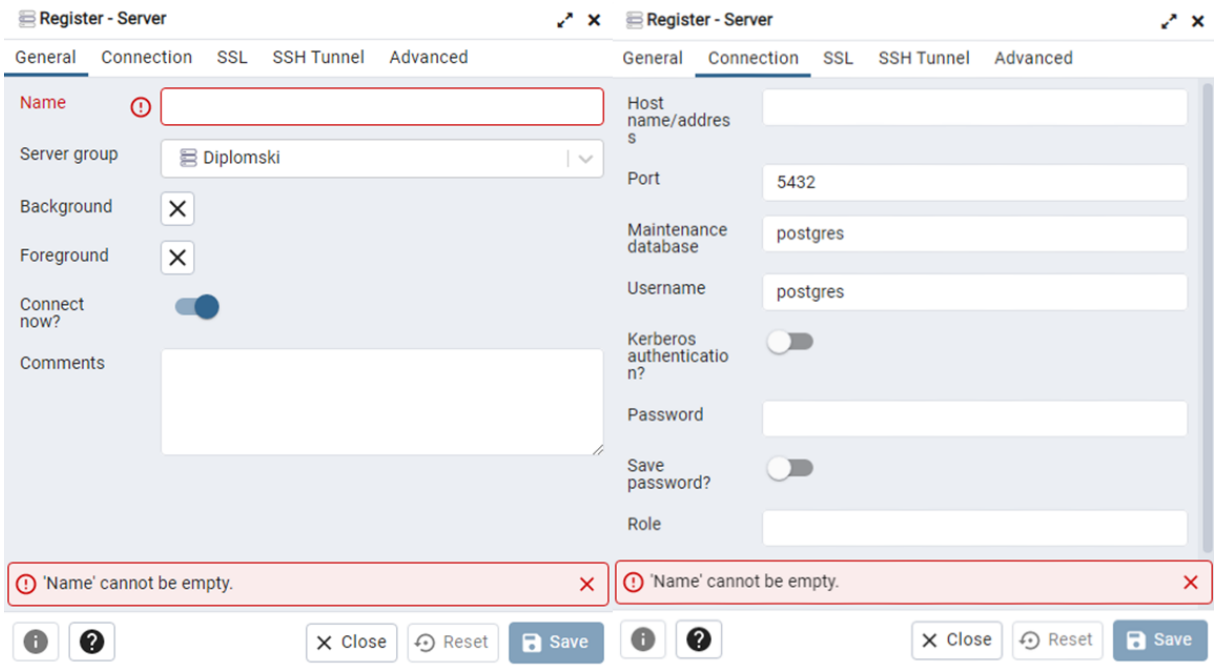
Pruža jednostavan način za interakciju s PostgreSQL bazom podataka putem grafičkog sučelja, umjesto korištenja komandne linije, ali moguće je koristiti SQL skripte za naprednije upite. Omogućava jednostavno kreiranje, modificiranje i brisanje baza podataka, tablica, indeksa i drugih objekata baze podataka. Pruža alate za upravljanje korisnicima baze podataka, dodjeljivanje prava i postavljanje sigurnosnih postavki.

Uspješnom prijavim dobije se prozor za upravljanje serverima i bazama koje su registrirane na određeni profil i računalo. Nakon povezivanja, sve baze podataka na serveru bit će dostupne za upravljanje. S lijeve strane vidljivi su svi dostupni serveri koji su kreirani na računalu. Svaki server može sadržati svoje servere gdje će svaki imati vlastite baze podataka. PgAdmin do samog početka nudi vlastite zadane servere gdje svaki ima dodatnu zaštitu lozinkom.



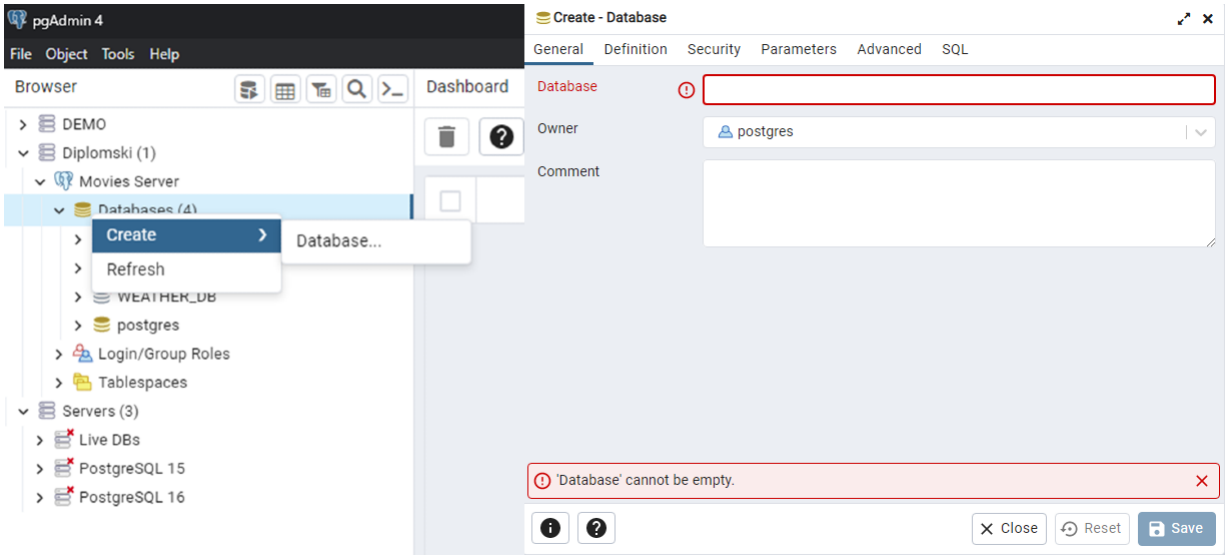
Slika 2. PgAdmin sučelje

U kartici "General", potrebno je unijeti ime servera, na primjer "MojServer". Server group može ostati zadano ako se izrađuje server na grupi koja je odabrana. Na kartici „Connection“ potrebno je unijeti informacije za povezivanje s PostgreSQL serverom. Host name/address je IP adresa ili ime hosta servera, na primjer localhost za lokalni server pošto je aplikacija lokalna i na taj način će se testirati. Port na kojem PostgreSQL server radi, što je najčešće 5432, ali može biti koji god broj ako je 5432 zadužen za ostale servise na računalu. Maintenance database je naziv postojeće baze podataka za povezivanje, postgres je zadano i može tako i ostati. Username ili korisničko ime za povezivanje (npr. postgres). Password ili lozinka za korisnički račun koji će se povezati iz Django aplikacije.



Slika 3. Kreiranje servera

Desnim klikom na server u stablu objekata i odabirom opcije za kreiranje nove baze podataka može se lako dodati novu bazu. Kako bi se kreirala baza potrebno je samo unijeti njeno ime i vlasnika, to jest korisnik putem kojeg je povezana. Dodatno je potrebno kreirate tablice unutar baze i njene stupce, ali pošto će se izvršiti povezivanje na Django aplikaciju s unaprijed definiranim modelima, ovdje to nije potrebno učiniti. Ako je potrebno dodatno kreirati tablicu s podatcima to je moguće izvršiti pisanjem klasične SQL skripte(vidljivo na kodu ispod slike 4).

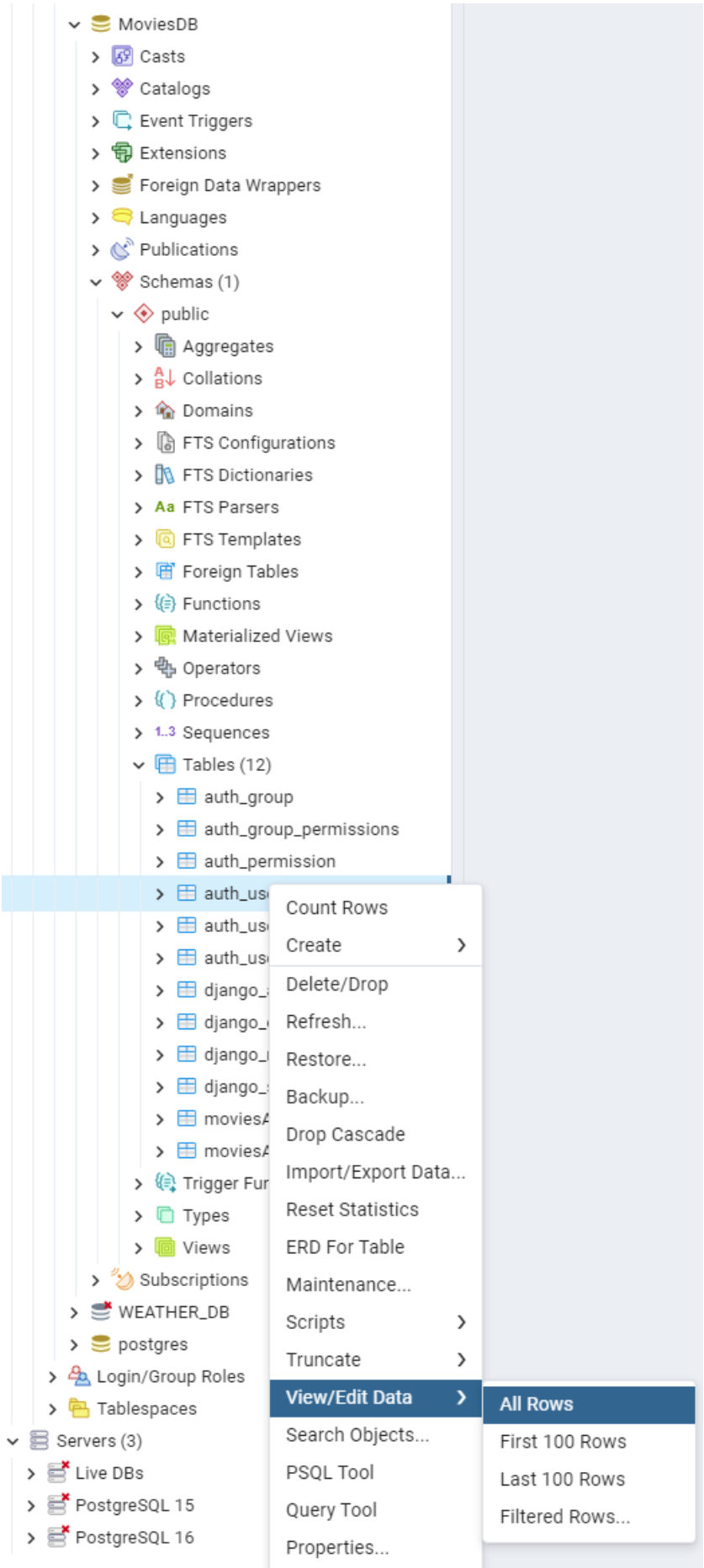


Slika 4. Kreiranje tablice

```
CREATE TABLE korisnici (
    id INT PRIMARY KEY,
    ime VARCHAR(50),
    prezime VARCHAR(50),
    email VARCHAR(100)
```

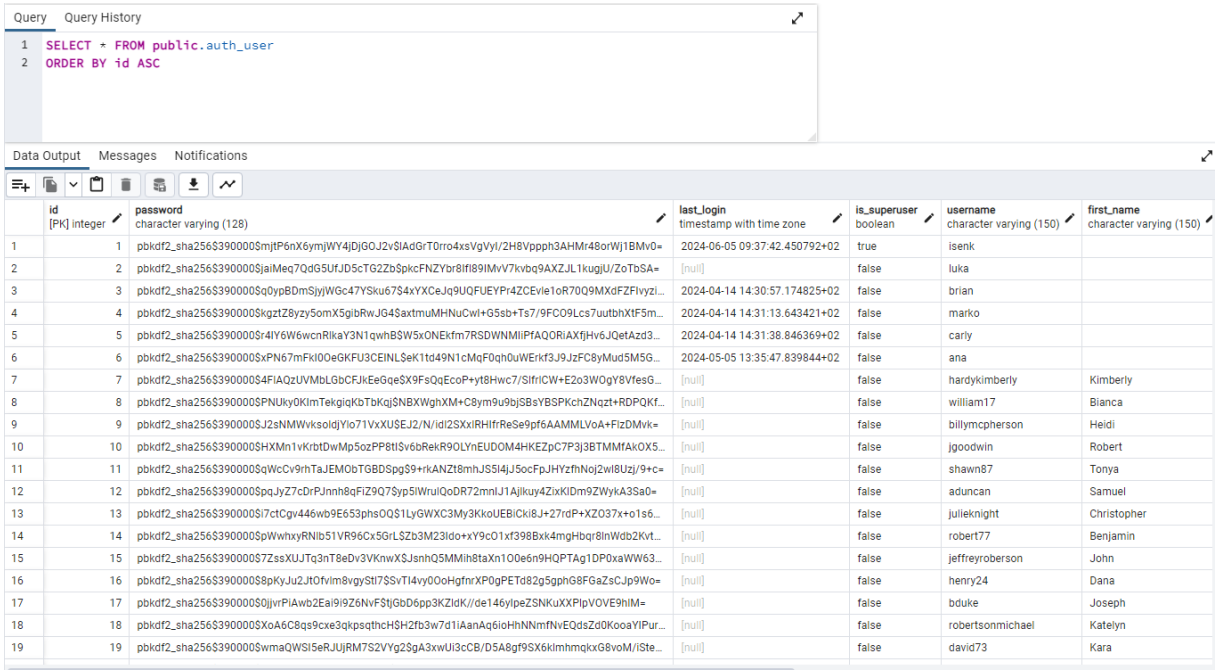
```
);

INSERT INTO korisnici (ime, prezime, email) VALUES ('Ivan', Senkovic,
'ivan.senkovic@mail.hr');
```

Glavni razlog korištenja PgAdmin je zbog toga što nudi dobar vizualni pregled svih podataka. Prvobitno znanje pisanja SQL skripta nije potrebno za pregled podataka zato što je sve moguće izvršiti klikom na tablice putem prije definiranim skriptama. Kako bi se pogledali podaci u tablicama potrebno je ići klikom na „Schemas“ -> „Tables“ -> „ime_tablice“ zatim desnim klikom odabrati „View/Edit Dana“ i na kraju odabrati „All Rows“ ako je potrebno pregledati sve redove sa svim podacima. Pregled svih redova može biti sporiji, ali je potrebno ako hoćemo provjeriti ako su svi stupci popunjeni ili slično.


MoviesDB
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (12)
auth_group
auth_group_permissions
auth_permission
Trigger Fur
Types
Views
Subscriptions
WEATHER_DB
postgres
Login/Group Roles
Tablespaces
Servers (3)
Live DBs
PostgreSQL 15
PostgreSQL 16
Count Rows
Create
Delete/Drop
Refresh...
Restore...
Backup...
Drop Cascade
Import/Export Data...
Reset Statistics
ERD For Table
Maintenance...
Scripts
Truncate
View/Edit Data
Search Objects...
PSQL Tool
Query Tool
Properties...
All Rows
First 100 Rows
Last 100 Rows
Filtered Rows...

Slika 5. Pregled podataka tablice

Nakon odabira određene opcije pokrene se unaprijed definirana SQL skripta za odabir svih redova(vidljivo na slici 6). Također na slici 6 moguće je vidjeti podatke koji su spremljeni s Django strane gdje se nalazi „id“ svih korisnika, njihova lozinka ili password koji je enkriptan zbog sigurnosti, zadnji put kad se korisnik prijavio(neki korisnici nemaju taj podatak pošto nisu „pravi“ korisnici), ako je korisnik superuser ili ako ima administratorska prava i username koji simbolizira korisničko ima. Ovo su jedna od najbitnijih polja koja su potrebna za prijavu korisnika i moguće je koristiti neke od ovih podataka za dodatne metode preporuke filmova.



	id [PK] integer	password character varying (128)	last_login timestamp with time zone	is_superuser boolean	username character varying (150)	first_name character varying (150)
1	1	pbkdf2_sha256$390000$mjtP6nX6ymjWY4jDjGOJ2v$IAdGrT0rro4xsVgVyI/2H8Vppph3AHMr48orWj1BMv0=	2024-06-05 09:37:42.450792+02	true	isenk	
2	2	pbkdf2_sha256$390000$jaiMeq7QdG5UfJD5cTG2Zb$pkcFNZYbr8lfI89IMvV7kvbq9AXZJL1kugjU/ZoTbSA=	[null]	false	luka	
3	3	pbkdf2_sha256$390000$q0ypBDmSjyjWGc47YSku67$4xYXCeJq9UQFUEYPr4ZCEvle1oR70Q9MXdFZFIvyzi...	2024-04-14 14:30:57.174825+02	false	brian	
4	4	pbkdf2_sha256$390000$kgztZ8yzy5omX5gibRwJG4$axtmuMHNuCwI+G5sb+Ts7/9FCO9Lcs7uutbhXtF5m...	2024-04-14 14:31:13.643421+02	false	marko	
5	5	pbkdf2_sha256$390000$r4lY6W6wcnRlkaY3N1qwhB$W5xONEkfm7RSDWNMliPfAQORiAXfjHv6JQetAzd3...	2024-04-14 14:31:38.846369+02	false	carly	
6	6	pbkdf2_sha256$390000$xPN67mFkI0OeGKFU3CEINL$eK1td49N1cMqF0qh0uWErkf3J9JzFC8yMud5M5G...	2024-05-05 13:35:47.839844+02	false	ana	
7	7	pbkdf2_sha256$390000$4FlAQzUVMbLGbCFJkEeGqe$X9FsQqEcoP+yt8Hwc7/SlfrICW+E2o3WOgY8VfesG...	[null]	false	hardykimberly	Kimberly
8	8	pbkdf2_sha256$390000$PNUky0KImTekgiqKbTbKqj$NBXWghXM+C8ym9u9bjSBsYBSPKchZNqzt+RDPQKf...	[null]	false	william17	Bianca
9	9	pbkdf2_sha256$390000$J2sNMWvksoldjYlo71VxXU$EJ2/N/idI2SXxlRHIfrReSe9pf6AAMMLVoA+FIzDMvk=	[null]	false	billymcpherson	Heidi
10	10	pbkdf2_sha256$390000$SHXMn1vKrbtDwMp5ozPP8tI$v6bRekR9OLYnEUDOM4HKEZpC7P3j3BTMMfAkOX5...	[null]	false	jgoodwin	Robert
11	11	pbkdf2_sha256$390000$qWcCv9rhTaJEMObTGBDSpg$9+rkANZt8mhJS5l4jJ5ocFpJHYzfhNoj2wI8Uzj/9+c=	[null]	false	shawn87	Tonya
12	12	pbkdf2_sha256$390000$pqJyZ7cDrPJnnh8qFiZ9Q7$yp5lWrulQoDR72mnlJ1AjIkuy4ZixKlDm9ZWykA3Sa0=	[null]	false	aduncan	Samuel
13	13	pbkdf2_sha256$390000$l7ctCgv446wb9E653phsOQ$1LyGWXC3My3KkoUEBiCki8J+27rdP+XZO37x+o1s6...	[null]	false	julieknight	Christopher
14	14	pbkdf2_sha256$390000$pWwhxyRNIb51VR96Cx5GrL$Zb3M23Ido+xY9cO1xf398Bxk4mgHbqr8lnWdb2Kvt...	[null]	false	robert77	Benjamin
15	15	pbkdf2_sha256$390000$7ZssXUJTq3nT8eDv3VKnwX$JsnhQ5MMih8taXn1O0e6n9HQPTAg1DP0xaWW63...	[null]	false	jeffreyroberson	John
16	16	pbkdf2_sha256$390000$8pKyJu2JtOfvlm8vgySTl7$SvTI4vy0OoHgfnrXP0gPETd82g5gphG8FGaZsCJp9Wo=	[null]	false	henry24	Dana
17	17	pbkdf2_sha256$390000$0jjvrPiAwb2Eal9l9Z6NvF$tjGbD6pp3KZldK//de146ylpeZSNKuXXPlpVOVE9hlM=	[null]	false	bduke	Joseph
18	18	pbkdf2_sha256$390000$XoA6C8qs9cxe3qkpsqthcH$H2fb3w7d1iAanAq6ioHhNNmfNvEQdsZd0KooaYlPur...	[null]	false	robertsonmichael	Katelyn
19	19	pbkdf2_sha256$390000$wmaQWSI5eRJUjRM7S2VYg2$gA3xwUi3cCB/D5A8gf9SX6klmhmqkxG8voM/iSte...	[null]	false	david73	Kara

Slika 6. Upit na bazi podataka

Neke od prednosti korištenja pgAdmina:

- Intuitivno grafičko sučelje: Pruža jednostavan način za upravljanje PostgreSQL bazama podataka bez potrebe za komandnom linijom.
- Snažni alati za administraciju: Omogućava upravljanje korisnicima, pravima pristupa, sigurnosnim postavkama i migracijama.
- Izvršavanje SQL upita: Omogućava pisanje, izvršavanje i analizu SQL upita u grafičkom sučelju.
- Vizualizacija podataka: Pruža alate za pregled i uređivanje podataka u tabličnom formatu.
- Podrška za više platformi: Dostupan na Windows, macOS i Linux operativnim sustavima.

2.3. Apache spark

Apache Spark je moćan open-source alat ili stroj za obradu podataka u stvarnom vremenu i velikih podataka(engl. big data). Klaster računala (engl. Computer Cluster) je skup povezanih računala koja rade zajedno kao jedan sustav. Ova računala, često nazivana čvorovima (engl. nodes), međusobno su povezana putem brzih mreža i zajedno izvršavaju zadatke kako bi postigla bolje performanse, pouzdanost i skalabilnost nego što bi to bilo moguće s jednim računalom. Klasteri omogućavaju paralelno izvršavanje zadataka, čime se značajno povećavaju računalne performanse i smanjuje vrijeme obrade velikih podataka. Nude visoku dostupnost aplikacija jer kvar jednog čvora ne uzrokuje prekid rada cijelog sustava. Ostali čvorovi mogu preuzeti zadatke neuspjelog čvora. Razvijen je za brzu obradu i analizu velikih setova podataka distribuiranih preko klastera računala. Spark podržava razne API-je za Java, Scala, Python i R, što ga čini vrlo fleksibilnim za različite programere.

Spark je dizajniran za rad s velikim količinama podataka koje se ne mogu učinkovito obraditi na jednom računalu. Koristi distribuiranu obradu podataka preko klastera. Nisu sva alati sposobni raditi na velikim količinama podataka te je iznimno bitno odabrati pravi i samim time bio je Spark odabran pošto je dobro razvijen i testiran. Koristeći klaster računala, Spark dijeli podatke na manje dijelove koji se paralelno obrađuju. RDD (engl. Resilient Distributed Dataset): Osnovna struktura podataka u Sparku. RDD je distribuirani niz podataka otporan na greške, koji se može obraditi paralelno. Za distribuiranu obradu Spark koristi više čvorova u klasteru za paralelno izvršavanje zadataka. Podaci su podijeljeni u particije koje se paralelno obrađuju na različitim čvorovima. Spark koristi razne ostale biblioteke Pythona kako bi ubrzalo i poboljšalo rad, ali ima i sam svoje dodatno definirane metode. U Django aplikaciji, potrebno je kreirati skripte ili pozadinske zadatke koji će koristiti Spark za obradu podataka.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("DjangoSparkApp") \
    .getOrCreate()

df = spark.read.format("csv").option("header", "true").load("path/to/csvfile")
df.show()
```

Spark MLlib je ugrađena biblioteka za strojno učenje koja omogućuje izradu i izvođenje složenih modela strojnog učenja na velikim skupovima podataka. Za algoritme MLlib pruža razne algoritme za klasifikaciju, regresiju, klasteriranje i filtriranje preporuka. Kod pipeline-a Spark MLlib podržava izradu pipeline-ova za kombiniranje višestrukih transformacija i procjena u jedinstven model. Jedan od klasičnih promjera se može vidjeti u kodu ispod. Uvijek je potrebno incijalizirati Spark te postaviti željeno ime koje se može mijenjati svaki put kad se kod pokrene. Podaci su najčešće spremljeni u csv datotekama pošto je globalno prepoznatljivo te lagano je putem Python-a obraditi takve podatke i preipremiti za daljnje korištenje. Također prednost je što se podatke može izvesti iz tablica u bazi podataka kao csv datoteka i odabrati željeni „delimiter“ što je oznaka kojima se vrijednosti odvajaju.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import SparkSession
```

```
# Inicijalizacija Spark sesije
spark = SparkSession.builder.appName("MLApp").getOrCreate()

# Učitavanje podataka
data = spark.read.csv("hdfs://path/to/data.csv", header=True, inferSchema=True)

# Priprema podataka
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
data = assembler.transform(data)

# Podjela podataka na trening i testne skupove
train, test = data.randomSplit([0.7, 0.3])

# Izrada i treniranje modela
lr = LogisticRegression(labelCol="label", featuresCol="features")
model = lr.fit(train)

# Evaluacija modela
predictions = model.transform(test)
predictions.select("features", "label", "prediction").show()
```

Spark Streaming omogućava obradu podataka u stvarnom vremenu, korisno za aplikacije koje trebaju trenutne analize i reakcije na dolazne podatke. Ovu vrstu nije potrebno koristiti za ovaj rad pošto se aplikacija ne nalazi u produkciji uživo, nego već lokalno. Ako se odabrao ovaj način bilo bi potrebno povezati razne ostale servise i samim time rad bi bio kompleksniji, ali u konačnici dobro povezan i sve bi se odrađivalo automatski bez dodatne kontrole korisnika. DStream (Discretized Stream) je osnovna jedinica podataka u Spark Streamingu. DStream je niz RDD-ova koji predstavljaju podatke iz stream izvora. Real-time obrada znači da Spark Streaming prikuplja podatke u intervalima i obrađuje ih kao male RDD-ove.

Spark omogućava interaktivno ispitivanje podataka koristeći SQL i DataFrame API-je, što je korisno za analitičare i znanstvenike podataka. Ovaj način je koristan za ovaj rad kad je potrebno programeru da ima uvid u sve što se događa u trenutku pokretanja koda. Ovakav kod je dobro pustiti aktivan samo dok se aplikacija gradi kako ne bi došlo do dodatnog čekanja izvršenja što bi bilo loše tijekom korištenja aplikacije. Nakon što sve radi moguće je rezultate jednom izvesti u datoteku te komentirati ovakve dijelove koda jer je moguće da se negdje u konzoli takav rezultat pregleda dok je aplikacija aktivna čak i lokalno. Primjer je moguće vidjeti u kodu.

```
from pyspark.sql import SparkSession

# Inicijalizacija Spark sesije
spark = SparkSession.builder.appName("SQLApp").getOrCreate()

# Učitavanje podataka u DataFrame
df = spark.read.csv("hdfs://path/to/data.csv", header=True, inferSchema=True)

# Izvršavanje SQL upita
df.createOrReplaceTempView("data")
result = spark.sql("SELECT feature1, feature2 FROM data WHERE label = 1")

# Prikaz rezultata
result.show()

# Korištenje DataFrame API
filtered_df = df.filter(df["label"] == 1).select("feature1", "feature2")
```

```
filtered_df.show()
```

Apache Spark je moćan alat za obradu velikih podataka, strojno učenje, stream obradu i interaktivnu analizu podataka. Njegova sposobnost rada s velikim setovima podataka, podrška za različite jezike i fleksibilnost u obavljanju raznih zadataka čini ga idealnim izborom za mnoge podatkovno intenzivne aplikacije. Postavljanje i održavanje Spark okruženja može biti složeno i zahtijeva dodatne resurse i stručnost. Spark omogućava korisnicima da učinkovito obrade, analiziraju i interpretiraju podatke, čime se povećava produktivnost i mogućnosti donošenja informiranih odluka te je to dovelo do odluke odabira za glavni dio ovog rada. Potrebno je pažljivo razmotriti zahtjeve i resurse aplikacije prije implementacije Sparka, jer može biti zahtjevan po pitanju memorije i kompleksnosti upravljanja.

2.4. Django

Django je visoko razrađeni web framework za programski jezik Python koji olakšava izradu i održavanje web aplikacija. Razvijen je s ciljem da se ubrza razvoj kompleksnih, bazno vođenih web aplikacija i istovremeno promovira čisti i pragmatičan dizajn. Django se temelji na principu "baterije uključene" (engl. "batteries-included"), što znači da dolazi s nizom ugrađenih značajki koje pokrivaju mnoge zajedničke web razvojne zadatke. Baza koja se koristi za razvoj aplikacija je SQL, to jest relacijski sustav što znači da NoSQL sustavi nisu mogući. U slučaju ako je nužno koristiti nerelacijske sustave to je moguće putem raznih Django ili Python biblioteka. Jedan primjer je Elastic Search koji je zasnovan na NoSQL te metodama moguće je obraditi podatke u JSON formatu te pretvoriti u SQL kako bi Django mogao raditi s takvim podacima i spremati ih u bazu.

Dostupno je mnogo različitih web framework-a za izradu aplikacija, ali ovaj je odabran zbog toga što je intuitivan i brz za uspostavljanje. Jedan o velikih prednosti je taj što je napisan pomoću Python programskog jezika i samim time sve skripte za preporuku filmova ili općenito algoritmi strojnog učenja odmah su ugrađeni u sam projekt. Neovisno o tome što se Python koristi kao osnova, moguće je koristiti ostale programske jezike ili frontend framework-ove kako bi se moglo poboljšati korisničko sučelje i općenito izgled aplikacije. Jedini nedostatak ovoga pristupa je taj što je potrebno imati znanje o više programskih jezika, na primjer ako više programera radi na aplikaciji svi moraju znati Python i JavaScript ili se svaka osoba podijeli na svoj dio koji zna. DRF pruža ugrađene mehanizme za autentifikaciju i autorizaciju, uključujući podršku za OAuth1a i OAuth2.

OAuth (Open Authorization) je standard za autorizaciju koji omogućava trećim stranama da dobiju ograničeni pristup korisničkim računima na web servisima, bez potrebe da dijele svoje lozinke. OAuth omogućava korisnicima da daju aplikacijama treće strane (kao što su web aplikacije ili mobilne aplikacije) pristup specifičnim resursima koje kontroliraju korisnici na nekom serveru (npr. Facebook), bez potrebe da dijele svoje lozinke s tim aplikacijama.

Django je dizajniran da olakša razvoj i ubrza proces izrade web aplikacija. Pruža razvojni okvir koji omogućuje brzu izradu prototipova, a njegov modularni pristup omogućava

lako proširivanje funkcionalnosti. Django aplikaciju pokreću se putem samo nekolika komandnih linije, bilo to cmd(engl. Command Prompt) ili powershell, zatim se automatski postavi projekt. Kod ostalih framework-ova ovakve instalacije nisu uvijek tako brzo, nego već zahtijevaju dodatna postavljanja okolina ili konfiguracije unutar projekta. Još jedna prednost je inicijalizacija baze podataka gdje kod ostalih framework-ova potrebne su dodane SQL skripte ili programi.

Django ima ugrađene sigurnosne značajke koje pomažu u zaštiti web aplikacija od uobičajenih prijetnji, poput SQL injekcija, cross-site scripting (XSS) i cross-site request forgery (CSRF). Automatski obrađuje sigurnosne aspekte tako da developeri mogu izbjeći mnoge uobičajene pogreške. Jedna od funkcija je to što nudi uključivanje ili isključivanje „DEBUG“ što znači da ispisi kodova, „print“ komanda u Python-u, se neće prikazivat u konzoli ili slično i samim time štiti aplikaciju od napada.

Django je dovoljno skalabilan za rukovanje najvećim i najprometnijim web aplikacijama. Njegova arhitektura omogućuje horizontalnu skalabilnost i lako dodavanje novih funkcionalnosti. Neke od aplikacija koju si napisane putem Django su: Bootcamp stranica koja služi za učenje, Disqus koja je jedna od najpoznatijih Django aplikacija nudi korisnicima integraciju komentara i diskusija na vlastite aplikacije te The Washington Post, Instagram, DropBox su jedne od aplikacija koje imaju dijelove izrađene korištenjem Django framework-a.

S obzirom na to da je razvijen s ciljem održavanja koda, Django koristi čisti i pragmatičan dizajn. Trudi se na tome da koristi ponovnu uporabu koda i koncept aplikacija koje se mogu lako proširivati i modificirati. Django se često ažurira s novim konceptima i funkcijama koji nisu samo od programera koji razvijaju, nego već i od zajednice koja je aktivna na njihovima forumima.

Kako bi se započeo rad s Django-m, potrebno je prvo instalirati Python i Django. Najprije, prvo je potrebna instalacija Python-a i preporučuje se uvijek imati najnoviju verziju instaliranu zbog toga što Django uvijek radi na zadnjoj verziji kako ne bi bilo metoda koje nisu dostupne. Naredbe za instalaciju ili unaprjeđenje Python-a i instalaciju Django su vidljive u isječku koda.

```
sudo apt-get update
sudo apt-get install python3
pip install django
```

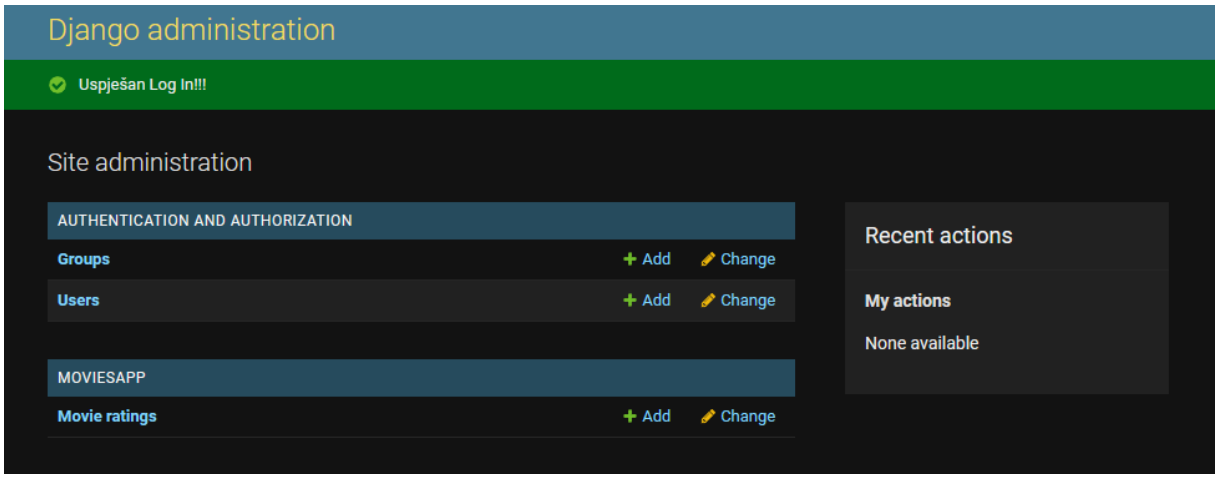
Naredba koja pokreće cijeli projekt je „django-admin startproject moj_projekt“ te njom se kreiraju sve potrebne mape i datoteke koje su unaprijed izrađene s Django strane kako bi nilo moguće pokrenuti praznu web aplikaciju i testirati. Ime projekt može biti koje god proizvoljno, ali mora biti sve zajedno napisano ili ako je potreban razmak moguće je staviti znak „_“. Ime projekta je bitno pošto će se koristiti kroz cijelu aplikaciju i projekt stoga na početku je potrebno unaprijed pravilno ime odabrati. Unutar glavnog direktorija projekta, pokreće se ugrađeni razvojni server pomoću komande „python manage.py runserver“ koja pokreće server na „localhost“ adresi, to jest pokreće aplikaciju lokalno najčešće na http://127.0.0.1:8000/ adresi. Adresu je moguće promijeniti na bilo koju što je iznimno bitno učiniti ako se aplikacija nalazi uživo na internetu. Sve ove komande se pokreću unutar cmd ili

powershell koji se nalaze unutar Visual Studio Code IDE(integrirano razvojno okruženje) što je uređivač koda koji je korišten za izradu ovog rada. Datoteke i mape imaju sljedeći raspored vidljivo u isječku.

```
moj_projekt/
    manage.py
    moj_projekt/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Većina promjena na projektu automatski se prenesu na aplikaciju gdje je potrebno samo ažurirati stranicu da se vide promjene. Promjene koje su napravljene unutar „settings.py“ preporuča se da se prvo ugasi server i ponovno upali nakon promjena kako bi se osiguralo da je sve spremljeno i konfigurirano za nastavak rada bez poteškoća.

Sljedeći korak je izrada same aplikacije komandom „python manage.py startapp moja_aplikacija“ gdje ime može biti proizvoljno kao i za projekt, ali se imena moraju razlikovati kako ne bi došlo do problema u pokretanju projekta i organizaciji. Aplikacija sadrži sve dokumente koji se često mijenjaju i konfiguriraju kako bi se mijenjala funkcionalnost aplikacije, njen izgled i raspored. U Django aplikaciji, svaka datoteka ima specifičnu svrhu i ulogu u okviru aplikacije. „__init__.py“ datoteka označava direktorij kao Python paket. To omogućava Pythonu da tretira direktorij kao modul, što znači da možete uvoziti (import) module i funkcije iz ovog direktorija. U kontekstu Django aplikacije, __init__.py je obično prazan, ali se može koristiti za inicijalizaciju modula ili definiranje paketnkog prostora imena. „admin.py“ se koristi za konfiguriranje administrativnog sučelja Django aplikacije. U ovoj datoteci registrirate modele aplikacije s Django administrativnim sučeljem kako bi se mogli upravljati putem admin sučelja. Ove promjene vidljive su na admin sučelju nakon što se administrator prijavi putem http://127.0.0.1:8000/admin adrese. Klasično sučelje moguće je vidjeti na slici 7.



Slika 7. Django administrator sučelje

Izgled se može dodatno promjeniti kontekstualno i vizualno s dodatnim Django modulima, ali za ovaj rad to nije bilo potrebno. „apps.py“ sadrži konfiguraciju aplikacije.

Definira klasu konfiguracije koja nasljeđuje AppConfig. Klasa konfiguracije se koristi za postavljanje određenih opcija specifičnih za aplikaciju, kao što su naziv aplikacije i put do aplikacije. Projekti koji su kompleksniji i zahtijevaju više aplikacija da rade zajedno ovdje će se nalazi imena svih aplikacija, ali za jednostavne projekte nalazi se ime samo jedne aplikacije i nije potrebno dodatno konfigurirati s time da se automatski generira. „models.py“ se koristi za definiranje modela podataka. Modeli predstavljaju strukturu podataka u aplikaciji i mapiraju se na tablice u bazi podataka koja je u ovom slučaju PostgreSQL. Modeli se definiraju kao klase koje nasljeđuju models.Model o kojima će se poslije u radu više objasniti. „tests.py“ se koristi za pisanje testova za aplikaciju. Testovi pomažu u osiguravanju da aplikacija radi ispravno i da promjene u kodu ne uvode bugove(greške u kodu). „views.py“ sadrži poglede (views) aplikacije. Pogledi određuju kako će aplikacija obraditi i odgovoriti na web zahtjeve. Pogledi mogu biti funkcije ili klase koje obrađuju zahtjeve i vraćaju odgovore kao što su HTML stranice, JSON podaci ili preusmjerenja. Ovo je možda najbitnija datoteka Django aplikacije pošto se ovdje nalaze funkcije i izgled svih stranica. „forms.py“ datoteka koja nije uvijek dio Django aplikacije, se koristi za definiranje formi koje se koriste za unos podataka putem web sučelja. „serializers.py“ se koristi u Django REST frameworku za definiranje serializatora koji pretvaraju modele u JSON i obrnuto, na primjer kad je u pitanje Elastic Search gdje se takvi podaci obrađuju zbog toga što se različite strukture baza koriste. „urls.py“ se koristi za definiranje URL ruta specifičnih za aplikaciju. Ova datoteka uključuje mapiranje URL obrazaca na odgovarajuće poglede bez kojih se stranice neće učitati ako nije pravilno postavljeno. Svaka od ovih datoteka igra ključnu ulogu u strukturi Django aplikacije, omogućujući modularnost i čistoću koda. Organiziranjem koda na ovaj način, Django omogućava jednostavno održavanje i proširenje aplikacija, te osigurava da su svi dijelovi aplikacije jasno odvojeni i odgovorni za specifične zadatke.

Sve komande izvršavaju se unutar glavne mape gdje se projekt nalazi, osim ako je izrađena Python virtualno okruženje(engl. Python virtual environment). Django-va teorija "baterije uključene", omogućuje developerima da se usredotoče na pisanje aplikacijske logike bez potrebe za implementacijom osnovnih funkcionalnosti iz nule. S obzirom na sve prednosti koje nudi, Django ostaje jedan od najpopularnijih i najpouzdanijih okvira za web razvoj u Pythonu.

1.1.1. Python virtual environment

Python virtual environment [15] (virtualno okruženje) je alat koji pomaže u održavanju izoliranog Python okruženja za projekte. Svako virtualno okruženje ima vlastitu instalaciju Pythona i vlastite pakete, što omogućava nezavisno upravljanje paketima i verzijama za svaki projekt. To je posebno korisno za projekte koji zahtijevaju različite verzije istih paketa ili za izbjegavanje sukoba među paketima. Ovaj način je koristan ako se na računalo lokalno nalazi mnogo instaliranih paketa ili modula stoga je teško voditi računa o tome koji se koriste, a koji ne. Također ako se radi „version control“ putem GitHub-a i ako drugi programer ili isto programer na drugom računalu nastavlja projekt, putem ovog pristupa mnogo je lakše klonirat projekt i započet raditi bez poteškoća.

Iako Python dolazi s ugrađenim alatom venv za kreiranje virtualnih okruženja, mnogi korisnici preferiraju virtualenv zbog dodatnih mogućnosti koje nudi. Instalacija se pokreće

komandom „pip install virtualenv“ te okruženje se kreira pomoću „virtualenv myenv“. Aktivacija se pokreće sa „myenv\Scripts\activate“ te nakon što je sve uspostavljeno može se krenuti s instalacijom svih paketa.

Komanda „pip freeze > requirements.txt“ je možda i najbitnija pošto je to najveći razlog korištenja ovoga okruženja. Unutar ove teksutalne datoteke nalaze se svi paketi s njihovim instaliranim verzijama, potrebni za razvoj aplikacije koji se instaliraju komandom „pip install -r requirements.txt“. Primjer sastava ove datoteke moguće je vidjet u isječku.

```
Django==3.2
djangorestframework==3.12.4
psycopg2==2.8.6
```

1.1.2. Settings.py

Datoteka settings.py u Django projektu sadrži sve osnovne postavke i konfiguracije koje Django koristi za upravljanje aplikacijom. U sljedećim isječcima koda biti će objašnjeno što svaka komponenta ove datoteke radi pošto je bitno za postavljenje projekta. „BASE_DIR“ definira osnovnu putanju projekta. Sve druge putanje u projektu definiraju se relativno prema ovoj putanji. Ovo je bitno zbog toga što se ovime definira kasnije također gdje će se spremati ili od kud učitavati ostale datoteke i slično.

```
from pathlib import Path

# Postavljanje osnovne putanje unutar projekta
BASE_DIR = Path(__file__).resolve().parent.parent
```

„SECRET_KEY“ je tajni ključ koji Django koristi za razne sigurnosne funkcije. Treba biti skriven i siguran u produkciji. „DEBUG“ koji ako je postavljen na True, Django prikazuje detaljne greške. U produkciji treba biti False kako se ispisi u kodu ako postoje ili ostale informacije ne ispisuju vanjskim korisnicima zbog sigurnosti. „ALLOWED_HOSTS“ je lista dopuštenih host-ova koji mogu pristupiti aplikaciji. U razvoju je obično postavljen na 127.0.0.1. Kad se aplikacija isporučuje na produkciju, ovu listu je bitno ažurirati s novom adresom koja je uživo ne internetu vidljiva i aktivna. Moguće je postaviti više od host-a, ali je potrebno obratiti pozornost ako je ispravno i da ih nije previše kako se ne bi zlonamjerno koristilo. Vidljivo u isječku.

```
# Tajni ključ (treba biti skriven u produkciji)
SECRET_KEY = "django-insecure-t*(y8-i$ja=(n@-=wqhc^q_5jp%5mrq)vhy_uxz4x&6@495ivy"

# Debug mod (ne smije biti uključen u produkciji)
DEBUG = True

# Dopušteni hostovi (u produkciji treba postaviti na stvarne domene)
ALLOWED_HOSTS = ["127.0.0.1"]
```

„INSTALLED_APPS“ je lista svih aplikacija koje su instalirane i aktivne u Django projektu. Ovdje uključuje osnovne Django aplikacije i dodatne aplikacije kao što su corsheaders i moviesApp. „django.contrib.admin“ koje je bilo prikazano na slici prije te omogućava administraciju modela putem grafičkog sučelja, dodavanje, izmjenu i brisanje podataka, te upravljanje korisnicima i dozvolama. Ovo je jedno od najkorisnijih Django alata jer omogućava administratorima lak pristup i upravljanje sadržajem baze podataka bez potrebe za izravnim pristupom bazi podataka. „django.contrib.auth“ pruža sustav autentifikacije i

autorizacije. Koristi se za kreiranje korisničkih računa, upravljanje prijavom i odjavom korisnika, te za definiranje i provjeru dozvola korisnika unutar aplikacije. „django.contrib.sessions“ služi za upravljanje sesijama korisnika. Koristi se za praćenje stanja korisnika kroz višestruke zahtjeve, primjerice za pohranu podataka o košarici u web trgovinama ili za čuvanje privremenih podataka. Ovaj se dio također koristi ako se radi dodatna preporuke proizvoda te bitno je kako se korisnik ponaša dok koristi stranicu. Sesije se koriste paralelno s „cookies“ kojima se može pristupiti putem JavaScript koda kako bi svaki korisnik ima svoj jedinstven ključ ili id(identifikacijski broj). „django.contrib.messages“ pruža sustav za slanje privremenih poruka korisnicima. Ove poruke mogu biti ispisane na sučelju stranice ovisno o tome gdje se postavi putem frontend konfiguracije. Poruke također mogu biti ispisane nakon što se generiraju putem JavaScript koda, ali nije dobro zbog toga što se ne može ispisati uvijek točno gdje je problem ili točna poruka uspješnosti. „django.contrib.staticfiles“ omogućava skupljanje i posluživanje statičkih datoteka kao što su CSS, JavaScript, slike i druge datoteke. Koristi se za organizaciju i posluživanje statičkih resursa potrebnih za frontend aplikacije, omogućujući lakše upravljanje i pristup tim datotekama. Ovaj dio aplikacije također je moguće prbaciti na vanjske servise popus Amazon AWS S3 gdje bi se sve datoteke nalazile na vanjskom cloud serveru te dohvaćalo kad je potrebno aplikaciji. „corsheaders“, to jest CORS (engl. Cross-Origin Resource Sharing), koristi se za postavljanje CORS pravila koja omogućuju ili ograničavaju pristup resursima iz različitih domena, što je korisno u situacijama kada frontend aplikacija i API server rade na različitim domenama ili portovima. Drugim riječima CORS je sigurnosna značajka web preglednika koja kontrolira kako resursi na web stranici mogu biti zatraženi s druge domene. “MoviesApp“ je ime za specifičnu aplikaciju unutar projekta za upravljanje funkcionalnostima vezanim za filmove. Ako ima više aplikacija, potrebno ih je sve uključiti u konfiguraciju. Sadrži modele, poglede, URL-ove, templejte i druge komponente potrebne za rad s podacima o filmovima. Koristi se za upravljanje podacima o filmovima, kao što su informacije o naslovima, glumcima, redateljima, recenzijama i slično. Sastav ove konfiguracije prikazan je na isječku koda. Na kraju „django_faker“ je dodatak koji je specifičan za slučaj ovoga rada pošto pruža alate za generiranje lažnih podataka. Koristi se za automatsko generiranje testnih ili lažnih podataka za razvoj i testiranje, što omogućava brže popunjavanje baze podataka sa stvarnim podacima za potrebe testiranja i razvoja. Ovo je dodano aplikaciji pošto bez da je aplikacije na produkciji, teško je imati dovaljan broj podataka.

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "corsheaders",
    “MoviesApp“,
    "django_faker",
]
```

„ROOT_URLCONF“ definira glavni modul za URL konfiguraciju projekta. Postavlja se koristeći sljedeći kod.

```
ROOT_URLCONF = "moviesProject.urls"
```

„CORS_ALLOW_ALL_ORIGINS“ dozvoljava sve domene da pristupaju resursima na serveru. Koristi se za Cross-Origin Resource Sharing (CORS).

```
CORS_ALLOW_ALL_ORIGINS = True
```

TEMPLATES je lista koja sadrži konfiguracije za različite backendove za templejte, direktorije gdje su smješteni templejti (DIRS), te kontekstne procesore koji su dostupni svim templejtima. U većini slučajeva koristi se samo jedan backend, ali Django podržava mogućnost korištenja više backendova. Svaki unos u listi TEMPLATES je rječnik (dictionary) koji sadrži postavke za određeni backend.

„BACKEND“ određuje koji se backend koristi za renderiranje predložaka, u ovom slučaju koristi Django-ov ugrađeni sistem za templejte. „DIRS“ je lista direktorija gdje Django traži predloške. „BASE_DIR / "templates"“ je putanja do direktorija templates unutar osnovnog direktorija projekta. Ovdje možete pohraniti vlastite predloške koji nisu specifični za pojedinačne aplikacije. „APP_DIRS“ je Boolean (True ili False) koji određuje da li Django treba automatski tražiti predloške unutar direktorija templates u svakoj instaliranoj aplikaciji. ada je postavljeno na True, Django će tražiti predloške unutar templates direktorija svake aplikacije navedene u INSTALLED_APPS. „OPTIONS“ su dodatne opcije za konfiguraciju templata. Rječnik koji može sadržavati razne postavke. U ovom primjeru, koristi se za definiranje context_processors. „context_processors“ unutar OPTIONS je lista putanja do funkcija koje dodaju varijable u kontekst svakog renderiranog predloška. Kontekstni procesori omogućuju dostupnost određenih varijabli svim predlošcima. Ostale vrijednosti:

„django.template.context_processors.debug“:

- Dodaje varijable za debuggiranje (samo ako je DEBUG postavljen na True).

„django.template.context_processors.request“:

- Dodaje request objekt u kontekst predloška, omogućujući pristup informacijama o trenutnom HTTP zahtjevu.

„django.contrib.auth.context_processors.auth“:

- Dodaje user objekt u kontekst, omogućujući pristup trenutnom korisniku (ako je korisnik prijavljen).

„django.contrib.messages.context_processors.messages“:

- Dodaje varijable za sustav poruka (messages framework), omogućujući prikazivanje privremenih poruka korisnicima. Kao što je prethodno bilo definiramo pod konfiguracijom za instalirane aplikacije.

TEMPLATES kao konfiguracija funkcionira za traženje predloška, tako što kada Django treba renderirati predložak, pretražuje direktorije navedene u DIRS i unutar templates direktorija svake aplikacije (ako je APP_DIRS postavljen na True). Prvo se traže predlošci u DIRS, a zatim unutar templates direktorija svake aplikacije po redoslijedu kojim su aplikacije navedene u INSTALLED_APPS. Za renderiranje predložaka kada se pronađe predložak, Django onda koristi odabrani BACKEND za renderiranje predloška. Kontekstni procesori definirani u OPTIONS dodaju varijable u kontekst predloška prije renderiranja, omogućujući

pristup dodatnim podacima unutar predloška. Na isječku je moguće vidjeti primjer konfiguracije predložaka.

```
moj_projekt/
│
├── manage.py
├── moj_projekt/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── templates/
│   ├── base.html
│   └── home.html
├── moja_aplikacija/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── views.py
│   └── templates/
│       └── moja_aplikacija/
│           └── detail.html
```

Također na primjeru u isječku je vidljivo kako se rendera predložak home.html iz views.py. Django će prvo potražiti home.html u BASE_DIR / "templates" i koristiti ga za renderiranje.

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

Ono što je moguće je renderirat predložak detail.html iz views.py unutar moja_aplikacija. Django će prvo potražiti predložak u templates direktoriju unutar moja_aplikacija aplikacije vidljivo u isječku.

```
from django.shortcuts import render

def detail(request):
    return render(request, 'moja_aplikacija/detail.html')
```

Dakle konfiguracija TEMPLATES u Django-u omogućuje definiranje kako se predlošci pronalaze, renderiraju i koje varijable im se dodaju. Ove postavke pružaju fleksibilnost i kontrolu nad načinom rada vaših predložaka, omogućujući jednostavno upravljanje i organizaciju vaših predložaka u projektu. Izgled ove konfiguracije za ovaj rad prikazano je u isječku.

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / "templates"],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
```

```
        },
    },
]
```

„DATABASES“ je konfiguracija za povezivanje s bazom podataka. U ovom slučaju koristi se PostgreSQL s nazivom baze podataka MoviesDB, korisničkim imenom postgres, lozinkom 1234, na localhost i portu 5432. ako se radi o vanjskom serveru na kojem je baza povezana bitno je ovdje također promijeniti vrijednosti s time da će se nalaziti skriveni ključ kao lozinka za povezivanje. Ključeve i slično moguće je spremiti u različitu datoteku zbog sigurnosti kako se ne bi moglo iz ove konfiguracije pročitati tijekom napada. Ovi podaci su bili prethodno postavljeni prilikom kreiranje baze podataka unutar PgAdmin programa. Port je ovdje bitan zbog toga što svako računalo ima svoje vlastite programe i servise te moguće je da jedan od njih već koristi traženi port pa je bitno testirati različite brojeve portova ako je to slučaj. Konfiguracija za SQLite koja je zadana prilikom pokretanja je jednostavna i sadrži samo njeno ime i engine. Konfiguracija za SQLite vidljiva je na isječku, kao i konfiguracija za PostgreSQL.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}
```

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "MoviesDB",
        "USER": "postgres",
        "PASSWORD": "1234",
        "HOST": "localhost",
        "PORT": "5432",
    }
}
```

„AUTH_PASSWORD_VALIDATORS“ ili validacija lozinki, je lista validatora koji se koriste za validaciju lozinki prilikom kreiranja ili mijenjanja korisničkih lozinki. U slučaju da nema ove konfiguracije, validacije bi se trebale izvršavati pomoću regex ili sličnih funkcija. Ova konfiguracija vidljiva je na isječku.

```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]
```

Međunarodizacija služi za određivanje lokalne zone aplikacije. Ovo je bitna komponenta zbog toga što vrijeme unutar aplikacije, ali i ostale funkcije ovise o vremenskoj

zoni te bez ovoga bilo bi grešaka. Samim time moguće je dodatno raditi konverzije vremenske ako je potrebno ali bitan je izvor. Vrijednosti nisu mijenjane zbog toga što nije bilo potrebno dodatno konfigurirati za ovaj rad. Konfiguracija je vidljiva na isječku.

- "LANGUAGE_CODE": Definira jezik koji će se koristiti u aplikaciji.
- "TIME_ZONE": Definira vremensku zonu za aplikaciju.
- "USE_I18N": Omogućuje međunarodizaciju (i18n).
- "USE_TZ": Omogućuje podršku za vremenske zone.

```
LANGUAGE_CODE = "en-us"
TIME_ZONE = "UTC"
USE_I18N = True
USE_TZ = True
```

Statičke i medijske datoteke su definirane u zasebnim varijablama unutar settings.py.

- STATIC_URL: URL putanja za pristup statičkim datotekama. Statičke datoteke uključuju CSS datoteke, JavaScript datoteke, slike koje nisu korisnički učitane, fontove i druge slične resurse. Kada se koristi STATIC_URL, web preglednik koristi ovaj URL za dohvaćanje statičkih datoteka. Primjer vrijednosti: STATIC_URL = "static/" znači da će sve statičke datoteke biti dostupne na URL putanji http://moja-domena/static/.
- MEDIA_URL: URL putanja za pristup medijskim datotekama. Medijske datoteke obuhvaćaju slike, dokumente, video zapise i druge vrste datoteka koje korisnici mogu učitati putem web aplikacije. Primjer vrijednosti: MEDIA_URL = "/images/" znači da će sve medijske datoteke biti dostupne na URL putanji http://moja-domena/images/.
- STATICFILES_DIRS: Ovo je popis direktorija u kojima Django traži dodatne statičke datoteke koje nisu dio pojedinačnih aplikacija. Uobičajeno je koristiti ovaj popis za specificiranje direktorija koji sadrže zajedničke statičke resurse za cijeli projekt. Primjer vrijednosti: STATICFILES_DIRS = [BASE_DIR / "static"] znači da se statičke datoteke nalaze u direktoriju static unutar vašeg osnovnog direktorija projekta (BASE_DIR).

```
STATIC_URL = "static/"
MEDIA_URL = "/images/"
STATICFILES_DIRS = [BASE_DIR / "static"]
```

U HTML predlošcima koristi se {% static 'path/to/file' %} za dohvaćanje statičkih datoteka i {% media 'path/to/file' %} za dohvaćanje medijskih datoteka (sko se koristi prilagođeni template tag za medijske datoteke). Potrebno je kreirati static direktorij u osnovnom direktoriju projekta za zajedničke statičke resurse. Također media direktorij za učitane datoteke je potrebno kreirati, te postaviti odgovarajuće postavke u settings.py kako bi Django znao gdje tražiti te datoteke. Tijekom razvoja, Django automatski služi statičke i medijske datoteke. U produkcijskom okruženju, statičke i medijske datoteke obično se služe putem posebnog web poslužitelja poput Nginxa ili Apachea za bolje performanse.

Ova settings.py datoteka pruža sve osnovne postavke koje su potrebne za rad Django projekta. Od konfiguracije baze podataka, aplikacija i middleware-a, do postavki za templejte

i statičke datoteke, sve ove postavke osiguravaju da Django projekt radi glatko i sigurno. Uvijek je važno prilagoditi ove postavke specifičnim potrebama vašeg projekta, osobito kada se prelazi iz razvojne u produkcijsku okolinu.

1.1.3. Urls.py

Prvo je potreban uvoz potrebnih modula i pogleda koji dolaze iz unaprijed definirah Django metoda. „from django.urls import path“ služi za uvoz path funkcije iz Django urls modula. Ova funkcija se koristi za definiranje URL obrazaca. „from . import views“ uvoz views modula iz trenutnog direktorija aplikacije. To omogućuje pristup funkcijama i klasama definiranima u views.py. „from django.conf import settings“ je uvoz settings modula iz Django conf. Koristi se za pristup postavkama definiranima u settings.py. „from django.conf.urls.static import static“ je uvoz static funkcije iz Django conf.urls.static. Koristi se za dodavanje statičkih i medijskih datoteka tijekom razvoja. Definiranje URL obrazaca je bitna stavka konfiguracije zbog toga što bez nje je nemoguće otvarati stranice koje su definirane unutar views. „urlpatterns“ je lista URL obrazaca, gdje svaki obrazac povezuje određeni URL s odgovarajućim pogledom.

path("", views.index, name="index") povezuje osnovni URL ("", što znači početnu stranicu) s pogledom index. views.index je funkcija ili klasa u views.py koja obrađuje zahtjev za ovu URL putanju. name="index" predstavlja ime URL obrasca, omogućava referenciranje ovog URL-a unutar predložaka i drugih dijelova koda. Index stranica je klasična stranica koja se koristi kod izgradnje web aplikacija te za ovaj projekt zadržano je ovo zadano ime da se raspoznaje koji view služi za prikaz naslovne stranice.

Za dodavanje statičkih i medijskih datoteka tijekom razvoja, koristi se static funkcija kako bi se povezali prije definirane direktoriji unutar settings.py datoteke. Dodaje URL obrasce za posluživanje medijskih datoteka tijekom razvoja. settings.MEDIA_URL je URL putanja za medijske datoteke (npr. /media/). settings.MEDIA_ROOT je direktorij na datotečnom sustavu gdje su pohranjene medijske datoteke. Za ovaj rad nije bilo potrebno koristiti zadnju konfiguraciju pošto nakon razvoja API poziva, slike su se posluživale putem URL koji je bio zaslužan za prikaz slike što je pomoglo za štednju memorije i moguće većom brzinom upita.

Datoteka urls.py definira URL obrasce za vašu Django aplikaciju, povezujući svaku URL putanju s odgovarajućim pogledom. Ova konfiguracija omogućava korisnicima da pristupe različitim dijelovima vaše aplikacije putem preglednika. Svaki path unutar urlpatterns definira specifičnu URL putanju, povezuje je s pogledom u views.py i daje joj ime za lakše referenciranje unutar aplikacije.

```
from django.urls import path
from . import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path("", views.index, name="index"),
    path("login/", views.loginPage, name="login"),
    path("logout/", views.logoutUser, name="logout"),
    path("register/", views.registerPage, name="register"),
    path("movies/", views.movie_list, name="movie_list"),
    path("movies/<str:api_id>/", views.movie_detail, name="movie_detail"),
```

```
    path("ratings/", views.user_ratings, name="user_ratings"),
    path("recommendation/", views.recommend_movies, name="recommend_movies"),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

1.1.4. Admin.py

Ova datoteka koristi se za konfiguraciju administrativnog sučelja Django aplikacije, omogućujući prilagodbu načina na koji se modeli prikazuju i upravljaju u Django administraciji.

„from django.contrib import admin“ znači uvoz admin modula iz Django contrib. Ovaj modul pruža sve potrebne alate za konfiguraciju administrativnog sučelja. „from .models import *“ je uvoz svih modela iz trenutnog modula (tj. aplikacije). * označava uvoz svih modela, što omogućava registraciju modela bez potrebe za njihovim pojedinačnim navođenjem. „from django.contrib.auth.models import User“ uvozi User modela iz Django auth aplikacije. Ovaj model predstavlja korisničke račune u Django-u. from „django.contrib.auth.admin import UserAdmin as BaseUserAdmin“ uvozi UserAdmin klase iz Django auth administracije i preimenovanje u BaseUserAdmin kako bi se izbjegao sukob imena kada definiramo vlastitu UserAdmin klasu. Ovi uvozi modula prikazani su u isječku.

```
from django.contrib import admin
from .models import *
from django.contrib.auth.models import User
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
```

Sljedeće prikazano u isječku je definicija prilagođenog korisničkog admina. „class UserAdmin(BaseUserAdmin)“ je definicija nove klase UserAdmin koja nasljeđuje BaseUserAdmin. Ova klasa se koristi za prilagodbu administrativnog sučelja za User model. „list_display = ("id", "username", "email", "first_name", "last_name", "is_staff")“ gdje list_display atribut koji određuje koje će se kolone prikazivati na popisu korisnika u administracijskom sučelju. Ovdje se specificira da je potrebno prikazati id, username, email, first_name, last_name i is_staff.

```
class UserAdmin(BaseUserAdmin):
    # Define a new User admin
    list_display = ("id", "username", "email", "first_name", "last_name", "is_staff")
```

U isječku koda je prikazano kako bi se također moglo definirati vlastite admin klase te na isječku je prikazano kako bi se definiralo klasu za narudžbe. „class OrderAdmin(admin.ModelAdmin)“ je definicija nove klase OrderAdmin koja nasljeđuje ModelAdmin. Ova klasa se koristi za prilagodbu administrativnog sučelja za Order model (pretpostavlja se da Order model postoji u models.py). „list_display = ("id", "customer", "date_ordered", "complete", "transaction_id")“ gjde je list_display atribut koji određuje koje će se kolone prikazivati na popisu narudžbi u administracijskom sučelju. Ovdje smo specificirali da želimo prikazati id, customer, date_ordered, complete i transaction_id.

```
class OrderAdmin(admin.ModelAdmin):
    list_display = ("id", "customer", "date_ordered", "complete", "transaction_id")
```

Potrebno je ponovno registrirati prilagođeni UserAdmin model. „admin.site.unregister(User)“ uklanja (odjavljuje) User model iz administracijskog sučelja.

Ovaj korak je potreban jer želimo registrirati prilagođeni UserAdmin. „admin.site.register(User, UserAdmin)“ koji ponovno registrira User model, ali sada s prilagođenom UserAdmin klasom. To znači da će User model koristiti prilagođenu konfiguraciju u administracijskom sučelju. Ova konfiguracija ponovnom registrirana prikazana je na isječku koda. Zadnje što je ostalo je registracija modela u administracijskom sučelju. „admin.site.register(MovieRating)“ koje registrira MovieRating model u administracijskom sučelju. To omogućava upravljanje MovieRating modelom putem administracijskog sučelja. Pretpostavlja se da MovieRating model postoji u models.py. Prikazano na isječku ispod.

```
admin.site.unregister(User)
admin.site.register(User, UserAdmin)
admin.site.register(MovieRating)
```

Ova admin.py datoteka konfigurira način na koji se User i MovieRating modeli prikazuju i upravljaju u Django administracijskom sučelju. Ključne funkcionalnosti uključuju prilagodbu prikaza kolona za modele, prilagođavanje postojećeg UserAdmin sučelja i registraciju modela kako bi bili dostupni u administracijskom sučelju. Ova konfiguracija omogućava administratorima da učinkovito upravljaju podacima unutar aplikacije koristeći intuitivno grafičko sučelje.

1.1.5. Views.py

Datoteka views.py u Django aplikaciji sadrži logiku pogleda (views) koja određuje kako će se HTTP zahtjevi obraditi i koji će se odgovori vratiti korisnicima. Pogledi mogu biti definirani kao funkcije ili klase i koriste se za renderiranje HTML predložaka, vraćanje JSON odgovora, preusmjeravanje korisnika i druge zadatke.

„from django.shortcuts import render, redirect“ gjde render je funkcija koja uzima zahtjev, naziv predloška i kontekst (podaci) te vraća renderirani HTML odgovor. Redirect je funkcija koja preusmjerava korisnika na drugu URL adresu.

„from django.http import HttpResponse“, HttpResponse je klasa koja vraća HTTP odgovor s proizvoljnim sadržajem (obično se koristi za jednostavne tekstualne odgovore).

„from django.contrib.auth import authenticate, login, logout“ gdje je authenticate funkcija koja provjerava korisničke kredencijale. Login je funkcija koja prijavljuje korisnika. Logout funkcija koja odjavljuje korisnika.

„from django.contrib.auth.decorators import login_required“ je dekorator koji osigurava da korisnik mora biti prijavljen kako bi pristupio određenom pogledu. Ovo se automatski održava gdje nije potrebno definirati ništa posebno, nego se samo priloži nekoj od funkcija, zatim Django vodi brigu o tome ako je korisnik prijavljen.

„from .models import Movie, MovieRating“ je uvoz modela Movie i MovieRating iz trenutnog modula. Ovo je potrebno kako bi mogli vršiti pozive na bazu podataka i njene tablice ili modele kako se zovu u Django.

„from .forms import UserRegisterForm, RatingForm“ je uvoz formi UserRegisterForm i RatingForm iz trenutnog modula.

„from .recommendation import recommend_movies_for_user“ je uvoz funkcije recommend_movies_for_user iz recommendation modula.

Ovi uvozi biblioteka prikazani su na isječku ispod.

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from .models import Movie, MovieRating
from .forms import UserRegisterForm, RatingForm
from .recommendation import recommend_movies_for_user
```

Pogled indeks renderira početnu stranicu (index.html) s time da se URL konstruira pomoću moviesApp pošto je tako konfigurirano u projektu. Vidljivo na isječku.

```
def index(request):
    return render(request, "moviesApp/index.html")
```

loginPage je definirana za rukovanje prijavom korisnika u Django aplikaciji. Funkcija obrađuje GET i POST zahtjeve za stranicu za prijavu. „def loginPage(request)“ definira funkciju loginPage koja prima request objekt kao argument. request objekt sadrži sve podatke o trenutnom HTTP zahtjevu. „page = "login"“ inicijalizira varijablu page sa string vrijednošću "login". Ova varijabla se koristi u kontekstu za renderiranje predloška kako bi se identificiralo da je riječ o stranici za prijavu.

„if request.user.is_authenticated“ provjerava je li korisnik već prijavljen. request.user.is_authenticated vraća True ako je korisnik prijavljen, a False ako nije. „return redirect("index")“ provjerava ako je korisnik već prijavljen, preusmjerava ga na stranicu s URL imenom "index", koja je naslovna tako da se ne prijavljen korisnik ne upravlja stranicama koje nisu definirane za njega.

„if request.method == "POST"“ provjerava je li zahtjev metoda POST. POST zahtjevi se koriste za slanje podataka na server, u ovom projektu služi za prijavu korisnika. „username = request.POST.get("username").lower()“ dohvaća korisničko ime iz POST podataka i pretvara ga u mala slova radi dosljednosti. Ovo se radi često zbog toga što korisnici neće uvijek paziti ako su na točan način unijeli podatke, stoga uvijek gdje je to moguće dobro je raditi konverzije. „password = request.POST.get("password")“ dohvaća lozinku iz POST podataka.

„try“ pokušava dohvatiti korisnika iz baze podataka s korisničkim imenom koje je uneseno. U Pythonu if-else i try-except koriste se za različite svrhe. Obje strukture kontrolnog toka omogućuju različite načine upravljanja ponašanjem programa, ali se koriste u različitim kontekstima. if-else je kontrolna struktura koja se koristi za uvjetno izvršavanje koda. Koristi se za provjeru logičkih uvjeta i izvršavanje različitih blokova koda na temelju rezultata tih uvjeta. if-else se ne može koristiti za rukovanje iznimkama koje se javljaju tijekom izvođenja koda te ne može uhvatiti specifične greške koje se mogu pojaviti, kao što su greške pri radu s datotekama ili pristupom nepostojećim indeksima u listama. try-except je kontrolna struktura koja se koristi za rukovanje iznimkama (greškama) koje se mogu dogoditi tijekom izvršavanja koda. Omogućuje programerima da "uhvate" greške i izvrše odgovarajuće akcije kako bi spriječili pad programa. Povećava robusnost programa jer omogućuje rukovanje neočekivanim

situacijama bez pada programa. Koristi se za situacije gdje se očekuju iznimke; prekomjerno korištenje može dodati overhead (iako je to obično minimalno).

„user = User.objects.get(username=username)“ pokušava dohvatiti objekt korisnika iz modela User s korisničkim imenom koje odgovara unesenom. User model za ovaj projekt koji se koristi je unaprijed definiran od strane Django-a, ali moguće je izraditi vlastitu konfiguraciju za korisnike ako je potrebno da se njegovi podaci koristi na drugi način ili poveća broj informacija o korisnicima.

„except“ pregledava ako korisnik s tim korisničkim imenom ne postoji, baca se iznimka s porukom. „messages.error(request, "User does not exist!")“ znači da ako korisnik ne postoji, dodaje se poruka o grešci koja će se prikazati korisniku.

„user = authenticate(request, username=username, password=password)“ pokušava autentificirati korisnika pomoću unesenih korisničkih podataka (korisničko ime i lozinka). „if user is not None“ provjerava je li autentifikacija uspjela. authenticate vraća objekt korisnika ako su podaci ispravni, ili None ako nisu. „login(request, user)“ prijavljuje korisnika ako je autentifikacija uspjela. „messages.success(request, "Uspješan Log In!!!", extra_tags="login_success")“ dodaje poruku o uspješnoj prijavi koja će se prikazati korisniku. extra_tags se koristi za dodavanje dodatnih CSS klasa poruci. „return redirect("index")“ preusmjerava korisnika na početnu stranicu (URL ime "index") nakon uspješne prijave. „else“ znači ako autentifikacija nije uspjela (korisničko ime ili lozinka su netočni), izvršava se ovaj blok. Ovaj dio je moguće učiniti na kompleksnije načine, kao na primjer da gleda posebno korisničko ime i posebno lozinku nakon čega bi se točno ispisalo korisniku u kojem je polju greška pa čak i koje su znamenke netočno i slično.

„context = {"page": page}“ kreira kontekstni rječnik(engl. dictionary) koji će se proslijediti predlošku. Ovdje dodaje varijablu page koja ima vrijednost "login".

„return render(request, "moviesApp/login_register.html", context)“ renderira predložak login_register.html iz aplikacije moviesApp koristeći kontekstni rječnik. Ovo se izvršava kada je zahtjev metoda GET ili kada POST zahtjev nije uspio.

Funkcija loginPage obrađuje zahtjeve za prijavu korisnika. Ako je korisnik već prijavljen, preusmjerava ga na početnu stranicu. Ako je zahtjev metoda POST, pokušava autentificirati korisnika s unesenim korisničkim podacima. Ako je autentifikacija uspješna, prijavljuje korisnika i preusmjerava ga na početnu stranicu, uz prikazivanje poruke o uspjehu. Ako autentifikacija nije uspješna, prikazuje poruku o grešci. Ako je zahtjev metoda GET, jednostavno prikazuje stranicu za prijavu. Cijeli izvorni kod je vidljiv je na isječku.

```
def loginPage(request):
    page = "login"

    if request.user.is_authenticated:
        return redirect("index")

    if request.method == "POST":
        username = request.POST.get("username").lower()
        password = request.POST.get("password")

        try:
```

```
                user = User.objects.get(username=username)
            except:
                # ovaj message se nalazi u main.html
                messages.error(request, "User does not exist!")

            user = authenticate(request, username=username, password=password)

            if user is not None:
                login(request, user)
                messages.success(request,          "Uspješan          Log          In!!!",
extra_tags="login_success")
                return redirect("index")
            else:
                messages.error(request, "Username or password does not exist")

        context = {"page": page}
        return render(request, "moviesApp/login_register.html", context)
```

registerPage je funkcija definirana za rukovanje registracijom novih korisnika u Django aplikaciji. Funkcija obrađuje GET i POST zahtjeve za stranicu za registraciju. „def registerPage(request)“ definira funkciju registerPage koja prima request objekt kao argument. request objekt sadrži sve podatke o trenutnom HTTP zahtjevu. „form = UserCreationForm()“ inicijalizira UserCreationForm formu. Ova forma se koristi za kreiranje novog korisničkog računa i pruža polja za unos korisničkog imena i lozinke.

Sljedeće je Obrada POST zahtjeva unutar funkcije. „if request.method == "POST"“ provjerava je li zahtjev metoda POST.

„form = UserCreationForm(request.POST)“ ponovno inicijalizira UserCreationForm formu, ali sada s podacima iz POST zahtjeva (request.POST). Ovo omogućuje popunjavanje forme podacima koje je korisnik unio. Forme mogu biti konfigurarane s Django strane, ali i specifično je moguće definirati svako polje, kako će polje izgledati, koje se vrijednosti mogu unijeti u polje te kakav format moraju imati.

„if form.is_valid()“ provjerava je li forma valjana. Metoda is_valid provjerava sve validacijske uvjete definirane na formi i vraća True ako su svi uvjeti ispunjeni.

„user = form.save(commit=False)“ sprema podatke iz forme u novi User objekt, ali još ne zapisuje u bazu podataka (commit=False). Spremanje u bazu je dobro zaustaviti zbog toga što je dobro izvršiti dodatne provjere ili formatirati podatke. Ovo se radi ako nije moguće imati kontrolu nad podacima pa se na ovaj način osigura validnost.

„user.username = user.username.lower()“ pretvara korisničko ime u mala slova radi dosljednosti. Ovo osigurava da su sva korisnička imena u bazi podataka pohranjena malim slovima.

„user.save()“ sprema korisnika u bazu podataka.

„login(request, user)“ automatski prijavljuje korisnika nakon uspješne registracije. Ovo je napravljeno tako da korisnik imat korak manje za početak korištenja aplikacije što može osigurati veću interakciju.

„return redirect("index")“ preusmjerava korisnika na početnu stranicu (URL ime "index") nakon uspješne registracije i prijave. Ovo je napravljeno zbog toga što korisnik može odmah početi koristiti aplikaciju od one stranice gdje je zamišljeno da je najbolje.

„return render(request, "moviesApp/login_register.html", {"form": form})“ renderira predložak login_register.html iz aplikacije moviesApp koristeći kontekstni rječnik koji sadrži formu. Ovo se izvršava kada je zahtjev metoda GET ili kada POST zahtjev nije uspio (npr. ako forma nije valjana). Cijeli kod prikazan je u isječku.

```
def registerPage(request):
    form = UserCreationForm()

    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.username = user.username.lower()
            user.save()
            login(request, user)
            return redirect("index")
        else:
            messages.error(request, "Error se pojavio prilikom registracije!")

    return render(request, "moviesApp/login_register.html", {"form": form})
```

„def logoutUser(request)“ definira funkciju logoutUser koja prima jedan argument request. request objekt sadrži sve podatke o trenutnom HTTP zahtjevu koji je poslan na server.

„logout(request)“ poziva funkciju logout iz Django modula django.contrib.auth. Ova funkcija uklanja sesiju korisnika iz baze podataka. Postavlja request.user na AnonymousUser, što znači da korisnik više nije autentificiran. Briše sve podatke sesije vezane uz korisničku prijavu. Ova funkcija ne zahtijeva nikakve dodatne argumente osim request objekta. Ova funkcija je korisna za osiguravanje da korisnici mogu lako izaći iz svojih sesija, osiguravajući pritom da se svi podaci sesije brišu i da se korisnik vraća na početnu stranicu aplikacije. Ovo je važno za sigurnost i za korisničko iskustvo, jer omogućuje korisnicima da znaju da su uspješno odjavljeni i da se vraćaju na početnu točku aplikacije. Prikazano u isječku.

```
def logoutUser(request):
    logout(request)
    return redirect("index")
```

movie_list funkcija je definirana za rukovanje prikazom popisa filmova u Django aplikaciji. Funkcija obrađuje GET zahtjeve, pretražuje filmove putem TMDB (The Movie Database) API-ja, i vraća rezultate korisniku. „def movie_list(request)“ definira funkciju movie_list koja prima jedan argument request. request objekt sadrži sve podatke o trenutnom HTTP zahtjevu koji je poslan na server.

„search_query = request.GET.get("query")“ dohvaća vrijednost GET parametra query iz URL-a. Ako parametar nije prisutan, search_query će biti None.

„movies = []“ inicijalizira praznu listu movies koja će sadržavati rezultate pretraživanja ili popularne filmove.

„if search_query“ provjerava je li search_query definiran (nije None). Ako je definiran, izvršava se blok koda za pretraživanje filmova.

„api_key = "5db79aa066a2a3d49ea80f0641bdfd67"“ definira API ključ za TMDB API. Ovaj ključ je preuzet putem TMDB web stranice te je jedinstven za svakog prijavljenog korisnii te besplatan je za jednostavne osobne projekte.

„search_url" = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&query={search_query}"""“ kreira URL za pretraživanje filmova na TMDB API-ju, koristeći API ključ i search_query. Ovaj URL je preuzet s TMDB stranice koja za svaki poziv ima specifičan ključ te unaprijed definirane parametre. Svaki URL je isti samo što se mijenja kako se poziva ovisno o programskog jeziku ili čak o biblioteci. „f“ je Python oznaka koja se koristi ako je potrebno unijeti vrijednosti u rečenicu ili „string“ koje se nalaze unutar Python koda. Ovaj se način koristi zbog toga što inače nije moguće unijeti dinamičke vrijednosti poput id-jeva filomva i slično.

„response = requests.get(search_url)“ šalje GET zahtjev na search_url i sprema odgovor u response.

„movies = response.json()["results"]“ parsira JSON odgovor iz response i sprema listu filmova (pod ključem "results") u movies. Izrađeno je na ovaj način zbog toga što se je vizualno bolje programeru, lakše je koristiti takve podatke pošto je JSON format globalan te kako bi se moglo na više način pristupiti ovim podacima.

„else“ ako search_query nije definiran, izvršava se blok koda za dohvaćanje popularnih filmova. "url = f"https://api.themoviedb.org/3/movie/popular?api_key={api_key}"""“ kreira URL za dohvaćanje popularnih filmova na TMDB API-ju, koristeći API ključ. Ovaj se URL razlikuje od prethodno zbog toga što se dodaje „popular“ vrijednost u adresu kako bi se razlikovalo i dohvatilo popularne filmove koji se dnevno ažurirani na TMDB stranici.

Funkcija movie_list obrađuje zahtjeve za prikazom popisa filmova. Ako je prisutan parametar query u URL-u, funkcija šalje zahtjev na TMDB API za pretraživanje filmova prema zadanoj riječi. Ako parametar nije prisutan, dohvaća se popis popularnih filmova. Rezultati (lista filmova) se zatim prosljeđuju predlošku movie_list.html za prikazivanje korisnicima. Ova funkcija omogućuje korisnicima da pretražuju filmove ili pregledavaju popularne filmove direktno iz vaše Django aplikacije. Ovaj kod prikazan je na isječku.

```
def movie_list(request):
    search_query = request.GET.get("query")
    movies = []
    if search_query:
        api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
        search_url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&query={search_query}"
        response = requests.get(search_url)
        movies = response.json()["results"]
    else:
        api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
        url = f"https://api.themoviedb.org/3/movie/popular?api_key={api_key}"
        response = requests.get(url)
        movies = response.json()["results"]
```

```
    return render(request, "moviesApp/movie_list.html", {"movies": movies})
```

„movie_detail“ funkcija je definirana za rukovanje prikazom detalja o pojedinom filmu i omogućava korisnicima da ocijene i pregledaju film u Django aplikaciji. Funkcija obrađuje GET i POST zahtjeve, dohvaća podatke o filmu putem TMDB API-ja, prikazuje preporuke filmova i obrađuje ocjene i recenzije korisnika.

„@login_required“ je dekorator koji osigurava da korisnik mora biti prijavljen kako bi pristupio ovom pogledu. Ako korisnik nije prijavljen, bit će preusmjeren na stranicu za prijavu. login_url argument može biti specificiran ako želite preusmjeriti korisnika na drugu stranicu za prijavu (na primjer @login_required(login_url='login')).

„def movie_detail(request, api_id)“ definira funkciju movie_detail koja prima dva argumenta:

- request: Objekt koji sadrži sve podatke o trenutnom HTTP zahtjevu.
- api_id: id filma koji se dohvaća iz URL-a. ovaj se id filma prosljeđuje klikom na pojedini film korisnika te se proslijedi u adresu i ovu funkciju.

„movie_url“ kao i u prethodnoj funkciji kreira URL za dohvaćanje detalja o filmu na TMDB API-ju, koristeći API ključ i api_id. Dodan je parametar append_to_response koji uključuje dodatne informacije o filmu kao što su ključne riječi, žanrovi i krediti (glumci i ekipa). Ovi su podaci bitni zbog toga što se koriste za preporuku filmova kako bi bila potpuna i točnija, to jest što više podataka sadrži, to će preporuka biti bolja.

„stars“ kreira listu stars koja sadrži brojeve od 1 do 10, predstavljajući zvjezdice za ocjenu filma.

„director“ koristeći next funkciju i generator expression, dohvaća ime direktora iz podataka o ekipi (credits.crew). Ako direktor nije pronađen, vraća None.

„actors“ kreira listu actors koja sadrži imena prvih pet glumaca iz podataka o glumačkoj postavi (credits.cast).

„overview“ dohvaća opis (sinopsis) filma iz podataka o filmu (overview).

„existing_rating“ dohvaća postojeću ocjenu korisnika za film, ako postoji. Koristi se filter za filtriranje ocjena prema korisniku i ID-u filma, te first za dohvaćanje prve (i jedine) ocjene ako postoji. To se koristi ovdje zbog toga kako bi korisnik vizualno ima uvid ako je film već ocijenio i koju je ocjenu stavio.

„if request.method == "POST"“ provjerava je li zahtjev metoda POST kao i u ostalim funkcijama.

„form = MovieRatingForm(request.POST)“ inicijalizira MovieRatingForm formu s podacima iz POST zahtjeva (request.POST).

„if form.is_valid()“ provjerava je li forma valjana. Metoda is_valid provjerava sve validacijske uvjete definirane na formi i vraća True ako su svi uvjeti ispunjeni.

„rating_value = form.cleaned_data["rating"]“ dohvaća ocjenu iz validiranih podataka forme.

„review_text = form.cleaned_data.get("review_text", "") dohvaća tekst recenzije iz validiranih podataka forme. Ako tekst nije unesen, vraća prazan string.

„genres_csv i keywords_csv“ kreiraju CSV (comma-separated values) stringove za žanrove i ključne riječi iz podataka o filmu. Ovaj format datoeke se kreira kako bi se spremljeni podaci mogli koristiti dalje u projektu bez poziva na bazu podataka, ali i zbog toga što je moguće korisiti ove podatke kao testne izvan aplikacije, na primjer za korištenje raznih algoritma za preporuku tako da se samo prenesu podaci.

„MovieRating.objects.update_or_create“ kreira ili ažurira ocjenu filma u bazi podataka. Ako postoji ocjena za kombinaciju korisnika i filma, ažurira je. Ako ne postoji, kreira novu ocjenu. Izrađeno je na ovaj način kako se u bazu ne bi zapisalo nove podatke i izbjeglo spremanje duplikata što bi ugrozilo preporuku filmova i napunilo bazu s nepotrebnim podacima. defaults argument sadrži podatke koji se trebaju ažurirati ili postaviti prilikom kreiranja nove ocjene.

„return redirect("movie_detail", api_id=api_id)“ preusmjerava korisnika na stranicu s detaljima filma nakon uspješnog pohranjivanja ocjene, to jest vraća ga natrag na stranicu filma koji je ocjenio.

„initial_data“ kreira inicijalne podatke za formu ako korisnik već ima ocjenu za film. Ako ocjena postoji, koristi njezine vrijednosti; inače koristi None ili prazan string. Na ovaj način moguće je imati kontrolu kako će se nepostojeći podaci obraditi kako ne bi došlo do greške na aplikaciji.

„recommendations“ dohvaća preporuke filmova koristeći funkciju get_movie_recommendations. Ovaj je funkcija izvan ove datoteke te se ovdje poziva umjesto da je cijeli kod unutar jedne funkcije.

„recommended_movies_details“ kreira praznu listu za detalje preporučenih filmova.

„for movie_id in recommendations“ iterira kroz preporučene filmove i dohvaća njihove detalje koristeći funkciju get_movie_details. Ova funkcije je također vanjska te služi za dohvat najbitnijih informacija koje je potrebno prikazati korisniku, ali isto tako kako bi se ispisali programeru podatke da može vidjeti ako su podaci valjani.

„if movie_data_new“ provjerava ako su podaci o filmu uspješno dohvaćeni, zatim ih dodaje u listu recommended_movies_details.

„metrics“ evaluira preporuke koristeći funkciju evaluate_recommendations i ispisuje rezultate. Rezultati se ispisuju u konzolu ili komandnu liniju pošto to nije potrebno korisnicima aplikacije, već samo programeru tijekom razboj aplikacije.

„return render(request, "moviesApp/movie_detail.html", {...})“ renderira predložak movie_detail.html iz aplikacije moviesApp koristeći kontekstni rječnik koji sadrži:

- movie: Podaci o filmu.

- form: Forma za ocjenu i recenziju.
- stars: Lista zvjezdica za ocjenu.
- existing_rating: Postojeća ocjena korisnika za film (ako postoji).
- recommended_movies: Detalji preporučenih filmova.

Funkcija movie_detail obrađuje zahtjeve za prikazom detalja o pojedinom filmu, omogućava korisnicima da ocijene i pregledaju film, te prikazuje preporuke filmova. Funkcija dohvaća podatke o filmu putem TMDB API-ja, prikazuje postojeće ocjene i recenzije, te omogućava korisnicima da unesu nove ocjene i recenzije. Također, dohvaća preporučene filmove i prikazuje ih korisniku. Ova funkcija pruža bogato korisničko iskustvo za interakciju s detaljima filmova unutar Django aplikacije. Cijeli koda prikazan je na isječku.

```
@login_required
def movie_detail(request, api_id):
    api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
    movie_url = f"https://api.themoviedb.org/3/movie/{api_id}?api_key={api_key}&append_to_response=keywords,genres,credits"
    response = requests.get(movie_url)
    movie = response.json()

    stars = list(range(1, 11))

    director = next(
        (
            member["name"]
            for member in movie.get("credits", {}).get("crew", [])
            if member["job"] == "Director"
        ),
        None,
    )
    actors = [actor["name"] for actor in movie.get("credits", {}).get("cast", [])[:5]]
    overview = movie.get("overview", "")

    existing_rating = MovieRating.objects.filter(
        user=request.user, movie_id=api_id
    ).first()

    if request.method == "POST":
        form = MovieRatingForm(request.POST)
        if form.is_valid():
            rating_value = form.cleaned_data["rating"]
            review_text = form.cleaned_data.get("review_text", "")

            genres_csv = ",".join([genre["name"] for genre in movie.get("genres", [])])
            keywords_csv = ",".join(
                [
                    keyword["name"]
                    for keyword in movie.get("keywords", {}).get("keywords", [])
                ]
            )

            print("Genres: ", genres_csv)
            print("Keywords: ", keywords_csv)

            MovieRating.objects.update_or_create(
                user=request.user,
                movie_id=api_id,
```

```
                defaults={
                    "rating": rating_value,
                    "review_text": review_text,
                    "genres": genres_csv,
                    "keywords": keywords_csv,
                    "director": director,
                    "actors": ",".join(actors),
                    "overview": overview,
                },
            )
            return redirect("movie_detail", api_id=api_id)
    else:
        initial_data = {
            "rating": existing_rating.rating if existing_rating else None,
            "review_text": existing_rating.review_text if existing_rating else "",
        }
        form = MovieRatingForm(initial=initial_data)

    recommendations = get_movie_recommendations(api_id)
    recommended_movies_details = []
    for movie_id in recommendations:
        movie_data_new = get_movie_details(movie_id, api_key)
        if movie_data_new:
            recommended_movies_details.append(movie_data_new)

    metrics = evaluate_recommendations(api_id, recommendations, is_relevant)
    print(metrics)

    return render(
        request,
        "moviesApp/movie_detail.html",
        {
            "movie": movie,
            "form": form,
            "stars": stars,
            "existing_rating": existing_rating,
            "recommended_movies": recommended_movies_details,
        },
    )
```

user_ratings funkcija je definirana za rukovanje prikazom ocjena filmova koje je korisnik dao u Django aplikaciji. Funkcija obrađuje GET zahtjeve, dohvaća podatke o ocjenama korisnika iz baze podataka, koristi TMDB API za dohvaćanje detalja o filmovima, i vraća rezultate korisniku.

„user_ratings = MovieRating.objects.filter(user=request.user).order_by("-created_at")“ dohvaća sve ocjene iz modela MovieRating za trenutno prijavljenog korisnika (request.user) i sortira ih prema datumu kreiranja u opadajućem redoslijedu (-created_at). Na ovaj način Django obavlja pozive prema bazi podataka, umjesto da korisnik koristi SQL funkcije. Ovim putem moguće je filtrirati podatke na mnogo načina što je Django unaprijed definirao kako bi pozivi bili brzi i bez SQL koda koji može biti dugačak i kompleksan. Unutar Django dokumentacije mogući je pronaći sve funkcije za dohvaćanje podataka te moguće kombinacije. Sortiranje je još jedna funkcija koja omogućuje sortiranje podataka po kojem god polju, a znak „-„ znači da je opadajuće ili Z-A umjesto A-Z kao inače. Izrađeno je na ovaj način tako da korisnik uvijek ima zadnje ocijenjen film na prvom mjesto ako je napravio grešku prilikom ocjenjivanja ili slično.

„for rating in user_ratings“ iterira kroz svaku korisničku ocjenu u user_ratings. Iterira se na ovaj način kako bi se ispis mogao izvesti za svaki film zasebno. Također ovdje je moguće mijenjat format ili slično pa čak i dodati dodatne informacije koje nisu bile bitne za zapisati u bazu podatka, ali bi bile bitne krajnjem korisniku.

„movie_detail["user_rating"] = rating.rating“ dodaje korisničku ocjenu (rating.rating) u movie_detail pod ključem "user_rating".

„movies_details.append(movie_detail)“ dodaje movie_detail (koji sada sadrži i korisničku ocjenu) u listu movies_details. Ove se informacije spremaju u listu kako bi se moglo opet na frontend strani pokrenuti iteracija za dohvaćanje ovih podataka i prikazati ih ne sučelju korisniku.

Funkcija user_ratings obrađuje zahtjeve za prikazom ocjena koje je korisnik dao filmovima. Dohvaća korisničke ocjene iz baze podataka, koristi TMDB API za dohvaćanje detalja o filmovima, i kombinira podatke o filmovima s korisničkim ocjenama. Na kraju, prikazuje rezultate korisniku koristeći predložak user_ratings.html. Ova funkcija omogućuje korisnicima da vide popis svih filmova koje su ocijenili zajedno s detaljima o tim filmovima, pružajući bogato korisničko iskustvo za pregledavanje njihovih ocjena unutar Django aplikacije. Cijela funkcija prikazana je na isječku.

```
@login_required
def user_ratings(request):
    user_ratings     =     MovieRating.objects.filter(user=request.user).order_by("-
created_at")
    movies_details = []

    for rating in user_ratings:
        api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
        movie_url = (

f"https://api.themoviedb.org/3/movie/{rating.movie_id}?api_key={api_key}"
        )
        response = requests.get(movie_url)
        movie_detail = response.json()
        movie_detail["user_rating"] = rating.rating
        movies_details.append(movie_detail)

    return render(
        request, "moviesApp/user_ratings.html", {"movies_details": movies_details}
    )
```

1.1.6. utils.py

U Django projektima, utils.py (ili utilities.py) je datoteka koja se koristi za pohranu pomoćnih funkcija, klasa i modula koje su korisne za različite dijelove aplikacije. utils.py nije dio Django strukture po default-u, ali je korisna praksa kada želite organizirati zajednički kod koji se koristi u više dijelova aplikacije.

Svrha utils.py je organizacija koda tako što centralizira pomoćne funkcije i klase koje su korisne na više mjesta u projektu. omogućava ponovno korištenje koda, smanjujući dupliciranje. Ovo je bitno ako se neka funkcija koristi često za neki ispis ili provjeru podataka, dobro je imati ovakve funkcije zasebno koje se mogu pozivati umjesto uvijek pisati novi

programski kod. Poboljšava čitljivost i održavanje koda jer su sve pomoćne funkcije na jednom mjestu. U views.py ili bilo kojoj drugoj datoteci, moguće uvesti i koristiti funkcije iz utils.py. Najbolje je kreirati ovu datoteku u istom direktoriju kao i views.py zbog bolje organizacije i lakše uvoza funkcija.

is_relevant definirana je za provjeru relevantnosti preporučenih filmova u odnosu na određeni film. Funkcija provjerava da li dva filma dijele barem jedan zajednički žanr. def „is_relevant(movie_id, recommended_movie_id)“ definira funkciju is_relevant koja prima dva argumenta. Prvi argument je movie_id, to jest id filma za koji se provjerava relevantnost preporuke. Drugi argument recommended_movie_id, koji predstavlja ID preporučenog filma koji se uspoređuje s originalnim filmom.

„try“ započinje blok koda koji pokušava izvršiti dohvaćanje filmova iz baze podataka.

„movie = Movie.objects.get(movie_id=movie_id)“ pokušava dohvatiti objekt Movie iz baze podataka koji odgovara movie_id. Ako film s tim ID-om ne postoji, baca se Movie.DoesNotExist iznimka.

„recommended_movie = Movie.objects.get(movie_id=recommended_movie_id)“ pokušava dohvatiti objekt Movie iz baze podataka koji odgovara recommended_movie_id. Ako film s tim ID-om ne postoji, baca se Movie.DoesNotExist iznimka.

„set(movie.genres.split(", "))“ uzimaju se žanrovi filma, razdvajaju se prema zarezu i razmacima, te se pretvaraju u skup (set). Razdvajanje je prema zarezu zbog toga što je to u Python-u zadani separator ili „;“, ne samo Python nego i csv datoteke. Skup se koristi jer omogućava jednostavnu provjeru presjeka.

„set(recommended_movie.genres.split(", "))“ uzimaju se žanrovi preporučenog filma, razdvajaju se prema zarezu i razmacima, te se pretvaraju u skup (set).

„set(movie.genres.split(", ")) & set(recommended_movie.genres.split(", "))“ orovjerava presjek (intersection) dvaju skupova žanrova. Presjek dva skupa sadrži elemente koji su prisutni u oba skupa. Ako postoji barem jedan zajednički žanr, presjek će biti ne-prazan.

„bool(...)“ pretvara rezultat presjeka u logičku vrijednost (True ili False). Ako je presjek ne-prazan, vraća True, što znači da filmovi dijele barem jedan zajednički žanr. Ako je presjek prazan, vraća False.

„except Movie.DoesNotExist“ provjerava ako bilo koji od filmova ne postoji u bazi podataka, tada se uhvati iznimka Movie.DoesNotExist.

„return False“ vraća False, što znači da preporučeni film nije relevantan jer jedan ili oba filma ne postoje u bazi podataka.

Dakle funkcija is_relevant provjerava relevantnost preporučenih filmova temeljem zajedničkih žanrova. Koristeći try-except blok, osigurava se da se funkcija nosi s mogućim situacijama gdje filmovi ne postoje u bazi podataka. Ako oba filma postoje, provjerava se da li dijele barem jedan zajednički žanr i vraća se odgovarajuća logička vrijednost (True ili False). Ova funkcija je korisna za filtriranje preporuka filmova kako bi se osiguralo da preporuke budu relevantne prema žanrovima koje korisnik preferira. Prikazano na isječku.

```
def is_relevant(movie_id, recommended_movie_id):
    try:
        movie = Movie.objects.get(movie_id=movie_id)
        recommended_movie = Movie.objects.get(movie_id=recommended_movie_id)
        return bool(
            set(movie.genres.split(", ")) & set(recommended_movie.genres.split(", "))
        )
    except Movie.DoesNotExist:
        return False
```

Funkcija get_movie_details je definirana za dohvaćanje detalja o filmu s TMDB API-ja. Funkcija koristi requests biblioteku za slanje HTTP GET zahtjeva i rukovanje mogućim iznimkama koje se mogu pojaviti tijekom izvršenja zahtjeva.

„def get_movie_details(movie_id, api_key)“ definira funkciju get_movie_details koja prima dva argumenta. movie_id predstavlja id filma koji se dohvaća iz TMDB API-ja. api_key je API ključ za autentifikaciju pristupa TMDB API-ju.

Sljedeće je pokušaj slanja HTTP GET zahtjeva. „try“ započinje blok koda koji pokušava izvršiti HTTP GET zahtjev na TMDB API. Koristi requests.get za slanje HTTP GET zahtjeva na TMDB API koristeći zadani movie_id i api_key. Formatira URL koristeći f-string (f"...{movie_id}...{api_key}...") kako bi uključio movie_id i api_key u URL.

„response.raise_for_status()“ poziva raise_for_status metodu na response objektu kako bi provjerio je li zahtjev uspješno izvršen. Ova je funkcija dio response biblioteke što omogućuje lakše praćenje problema s pozivom. Sustavni kod 2xx znači da je zahtjev uspješan, ali ne mora uvijek značiti da ima podatke. Ako je statusni kod odgovora 4xx ili 5xx (što označava grešku), raise_for_status će podići HTTPError iznimku. Ako je zahtjev uspješan, parsira JSON odgovor i vraća ga kao Python rječnik. response.json() metoda parsira tijelo odgovora kao JSON i vraća odgovarajući Python objekt (obično rječnik).

„except requests.RequestException as e“ uhvaća sve iznimke koje su pod klase requests.RequestException. Ova klasa pokriva većinu grešaka koje se mogu dogoditi prilikom slanja HTTP zahtjeva, uključujući HTTPError, ConnectionError, Timeout i druge.

„print(f"Error fetching data for movie ID {movie_id}: {e}")“ ispisuje poruku o grešci na standardni izlaz (obično konzolu). Poruka uključuje movie_id i detalje iznimke e. Ako je došlo do iznimke, funkcija vraća None kako bi označila da dohvaćanje podataka nije uspjelo.

Funkcija get_movie_details koristi se za dohvaćanje detalja o filmu iz TMDB API-ja. Funkcija šalje HTTP GET zahtjev, rukuje mogućim greškama tijekom slanja zahtjeva i vraća parsirani JSON odgovor ako je zahtjev uspješan. Ako dođe do greške, funkcija ispisuje poruku o grešci i vraća None. Prikazano na isječku.

```
def get_movie_details(movie_id, api_key):
    try:
        response = requests.get(
            f"https://api.themoviedb.org/3/movie/{movie_id}?api_key={api_key}"
        )
        response.raise_for_status()
        return response.json()
    except requests.RequestException as e:
        print(f"Error fetching data for movie ID {movie_id}: {e}")
        return None
```

Dakle utils.py je korisna datoteka za organizaciju pomoćnih funkcija i klasa koje su zajedničke za više dijelova Django aplikacije. Korištenjem utils.py, vaš kod postaje organiziraniji, čitljiviji i lakši za održavanje. Kada imate funkcije koje se koriste na više mjesta u aplikaciji, preporuča se njihovo premještanje u utils.py kako bi se smanjila dupliciranost i poboljšala struktura koda.

1.1.7. fetch_popular_movies.py

Ova datoteka je Django skripta za upravljanje koja dohvaća popularne filmove s TMDB API-ja i sprema ih u bazu podataka. Skripta se pokreće pomoću Django manage.py komande: python manage.py fetch_popular_movies, slično kako se pokreće lokalni server.

Prvo je potrebno uvesti sve potrebne module i definicija klase. „from django.core.management.base import BaseCommand“ uvozi BaseCommand klasu iz Django core.management.base modula. Ovo je osnovna klasa za kreiranje prilagođenih komandi.

„import requests“ uvozi requests biblioteku koja se koristi za slanje HTTP zahtjeva.

„from datetime import datetime“ uvozi datetime klasu iz datetime modula za rad s datumima.

„from moviesApp.models import Movie“ uvozi Movie model iz moviesApp aplikacije. Ovo je model u koji će se spremati podaci o filmovima.

„class Command(BaseCommand)“ definira novu klasu Command koja nasljeđuje BaseCommand.

„help = "Fetches popular movies from TMDB and stores them in the database"“ pruža opis komande koji se prikazuje kada pokrenete python manage.py help fetch_popular_movies.

„def handle(self, *args, **options)“ definira handle metodu koja sadrži logiku komande. Ova metoda se poziva kada se pokreće ovu komandu.

„base_url = https://api.themoviedb.org/3/movie/popular"“ definira osnovni URL za dohvaćanje popularnih filmova. „detail_url = https://api.themoviedb.org/3/movie/{id}"“ definira URL za dohvaćanje detalja o pojedinom filmu.

„for page in range(1, 11)“ iterira kroz stranice rezultata popularnih filmova (od 1 do 10). ovaj se broj može povećati ovisno o tome koliko je potrebno filmova preuzeti, u ovom slučaju broj je ostao isti zbog toga što druga integracija ima bolji raspon filmova.

„url = f"{base_url}?api_key={api_key}&page={page}" kreira URL za dohvaćanje popularnih filmova za trenutnu stranicu.

„response = requests.get(url)“ šalje GET zahtjev na URL i sprema odgovor u response.

„if response.status_code == 200“ provjerava je li zahtjev uspješan (statusni kod 200).

„data = response.json()“ parsira JSON odgovor i sprema podatke u data.

„for movie in data["results"]“ iterira kroz filmove u rezultatima.

„movie_id = movie.get("id")“ dohvaća ID filma.

„movie_detail_response = requests.get(detail_url.format(id=movie_id) + f"?api_key={api_key}")“ šalje GET zahtjev za detalje o filmu koristeći movie_id i API ključ.

„if movie_detail_response.status_code == 200“ provjerava je li zahtjev za detalje uspješan (statusni kod 200).

„movie_details = movie_detail_response.json()“ parsira JSON odgovor i sprema podatke o filmu u movie_details.

„title = movie_details.get("title")“ dohvaća naslov filma.

„overview = movie_details.get("overview", "")“ dohvaća pregled (opis) filma. Ako opis nije dostupan, koristi prazan string. Ovaj je podatak bitan kako bi grafičko sučelje izgledalo „bogatije“ korisniku s količinom podataka.

„genres = ", ".join([genre["name"] for genre in movie_details.get("genres", [])])“ kreira CSV string žanrova filma. Ovdje se koristi „,“ kao separator kao što je bilo ranije opisano za CSV datoteke ili Python parsiranje podataka.

„release_date = datetime.strptime(movie_details.get("release_date", "1900-01-01"), "%Y-%m-%d").date()“ pretvara datum izlaska filma iz stringa u date objekt. Ako datum nije dostupan, koristi se defaultni datum "1900-01-01".

„Movie.objects.update_or_create“ sprema podatke o filmu u bazu podataka. Ako film s istim movie_id već postoji, ažurira ga; inače kreira novi unos. „movie_id=str(movie_id)“ koristi movie_id kao identifikator. „defaults={...}“ postavlja ostala polja (naslov, pregled, žanrove, datum izlaska) koristeći podatke iz movie_details.

Prvi „else“ argument se pokreće ako zahtjev za detalje o filmu nije uspješan, ispisuje poruku o grešci s ID-om filma i statusnim kodom odgovora. Drugi „else“ argument se pokreće ako zahtjev za popularne filmove nije uspješan, ispisuje poruku o grešci sa stranicom i statusnim kodom odgovora.

Na samom kraju print("Completed fetching and saving popular movies.")“ komanda ispisuje poruku koja označava da je dohvaćanje i spremanje popularnih filmova završeno.

Ovaj kod je prilagođena Django upravljačka komanda koja dohvaća popularne filmove s TMDB API-ja i sprema ih u Django bazu podataka. Koristi requests biblioteku za slanje HTTP zahtjeva i rukuje odgovorima API-ja. Komanda se može pokrenuti pomoću manage.py, omogućujući jednostavno ažuriranje baze podataka s najnovijim popularnim filmovima. Prikazano na isječku.

```
from django.core.management.base import BaseCommand
import requests
from datetime import datetime
from moviesApp.models import Movie


class Command(BaseCommand):
    help = "Fetches popular movies from TMDB and stores them in the database"

    def handle(self, *args, **options):
        api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
        base_url = "https://api.themoviedb.org/3/movie/popular"
```

```
                detail_url = "https://api.themoviedb.org/3/movie/{id}"

                for page in range(1, 11):
                    url = f"{base_url}?api_key={api_key}&page={page}"
                    response = requests.get(url)
                    if response.status_code == 200:
                        data = response.json()
                        for movie in data["results"]:
                            movie_id = movie.get("id")
                            movie_detail_response = requests.get(
                                detail_url.format(id=movie_id) + f"?api_key={api_key}"
                            )
                            if movie_detail_response.status_code == 200:
                                movie_details = movie_detail_response.json()
                                title = movie_details.get("title")
                                overview = movie_details.get("overview", "")
                                genres = ", ".join(
                                    [genre["name"] for genre in movie_details.get("genres",
[])]
                                )
                                release_date = datetime.strptime(
                                    movie_details.get("release_date", "1900-01-01"), "%Y-%m-
%d"
                                ).date()

                                # Save to database
                                Movie.objects.update_or_create(
                                    movie_id=str(movie_id),
                                    defaults={
                                        "title": title,
                                        "overview": overview,
                                        "genres": genres,
                                        "release_date": release_date,
                                    },
                                )
                            else:
                                print(
                                    f"Failed to fetch details for movie ID {movie_id}:
{movie_detail_response.status_code}"
                                )
                    else:
                        print(
                            f"Failed to fetch popular movies for page {page}:
{response.status_code}"
                        )

                print("Completed fetching and saving popular movies.")
```

1.1.8. fetch_top_rated_movies.py

Ovaj kod je vrlo sličan prethodnom kodu za dohvaćanje popularnih filmova, ali se umjesto toga fokusira na dohvaćanje filmova s najboljim ocjenama. Uvoz modula i definicija klase je identično kao i kod prethodnog koda pošto je najveća razlika URL.

„base_url = https://api.themoviedb.org/3/movie/top_rated“ definira osnovni URL za dohvaćanje filmova s najboljim ocjenama. „detail_url = https://api.themoviedb.org/3/movie/{id}“ definira URL za dohvaćanje detalja o pojedinom filmu.

„for page in range(1, 25)“ iterira kroz 24 stranice rezultata top-rated filmova. Razlika je to što prethodna skripta iterira kroz 10 stranica popularnih filmova. Ovaj se broj mijenjao kroz razvoj aplikacije zbog toga što je bilo potrebno imati dostupno što više podataka. Kombinira se ovaj poziv s prethodnim kako bi korisnik mogao ocijeniti veći broj filmova i dobiti preporuku. „url = f"{base_url}?api_key={api_key}&page={page}"“ kreira URL za dohvaćanje top-rated filmova za trenutnu stranicu.

Ovaj kod je vrlo sličan prethodnom primjeru, ali se fokusira na dohvaćanje top-rated filmova umjesto popularnih. Razlike između dvije skripte uključuju:

- API Endpoint: Prethodna skripta koristi https://api.themoviedb.org/3/movie/popular. Ova skripta koristi https://api.themoviedb.org/3/movie/top_rated.
- Broj stranica: Prethodna skripta iterira kroz 10 stranica popularnih filmova. Ova skripta iterira kroz 24 stranice top-rated filmova.
- Opseg podataka: Obje skripte dohvaćaju podatke o filmovima, uključujući naslov, pregled, žanrove i datum izlaska, te ih spremaju u bazu podataka, s time da druga skripta dohvaća češće mnogo više podataka. Prva skripta dohvaća često podatke i samim time ako se popularni filmovi mijenjaju onda će češće imati novije podatke.

Skripta se pokreće pomoću Django manage.py komande „python manage.py fetch_top_rated_movies“. Ovo će pokrenuti skriptu koja dohvaća top-rated filmove s TMDB API-ja i sprema ih u Django bazu podataka. Skripta koristi requests biblioteku za slanje HTTP zahtjeva i rukuje odgovorima API-ja, te pohranjuje podatke u bazu koristeći Django ORM. Kod je prikazan na isječku.

```
from django.core.management.base import BaseCommand
import requests
from datetime import datetime
from moviesApp.models import Movie


class Command(BaseCommand):
    help = "Fetches top-rated movies from TMDB and stores them in the database"

    def handle(self, *args, **options):
        api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
        base_url = "https://api.themoviedb.org/3/movie/top_rated"
        detail_url = "https://api.themoviedb.org/3/movie/{id}"

        for page in range(1, 25):
            url = f"{base_url}?api_key={api_key}&page={page}"
            response = requests.get(url)
            if response.status_code == 200:
                data = response.json()
                for movie in data["results"]:
                    movie_id = movie.get("id")
                    movie_detail_response = requests.get(
                        detail_url.format(id=movie_id) + f"?api_key={api_key}"
                    )
                    if movie_detail_response.status_code == 200:
                        movie_details = movie_detail_response.json()
                        title = movie_details.get("title")
```

```
                            overview = movie_details.get("overview", "")
                            genres = ", ".join(
                                [genre["name"] for genre in movie_details.get("genres",
[])]
                            )
                            release_date = datetime.strptime(
                                movie_details.get("release_date", "1900-01-01"), "%Y-%m-
%d"
                            ).date()

                            Movie.objects.update_or_create(
                                movie_id=str(movie_id),
                                defaults={
                                    "title": title,
                                    "overview": overview,
                                    "genres": genres,
                                    "release_date": release_date,
                                },
                            )
                        else:
                            print(
                                f"Failed to fetch details for movie ID {movie_id}:
{movie_detail_response.status_code}"
                            )
                else:
                    print(
                        f"Failed to fetch top-rated movies for page {page}:
{response.status_code}"
                    )

        print("Completed fetching and saving top-rated movies.")
```

1.1.9. content_based.py

Content-based filtering [16] je metoda preporučivanja koja koristi podatke o stavkama (na primjer filmovima, knjigama, proizvodima) kako bi preporučila korisnicima slične stavke. U kontekstu filmova, koristi informacije kao što su žanrovi, opisi, glumci, redatelji itd. kako bi pronašla sličnosti između filmova. [4]

Prvo je potrebno Prikupljanje podataka gdje se prikuplja skup podataka koji opisuje karakteristike stavki. U slučaju filmova, to može uključivati žanrove, sinopse, glumce, redatelje i slično. Pretvaranje podataka u vektore koristi tehnike poput CountVectorizer ili TF-IDF Vectorizer kako bi se tekstualni podaci pretvorili u numeričke vektore. Za mjerenje sličnosti koristi se mjera sličnosti poput kosinusne sličnosti kako bi se izračunala sličnost između stavki. Na temelju sličnosti, preporučuju se stavke koje su najsličnije onima koje je korisnik prethodno gledao ili ocijenio.

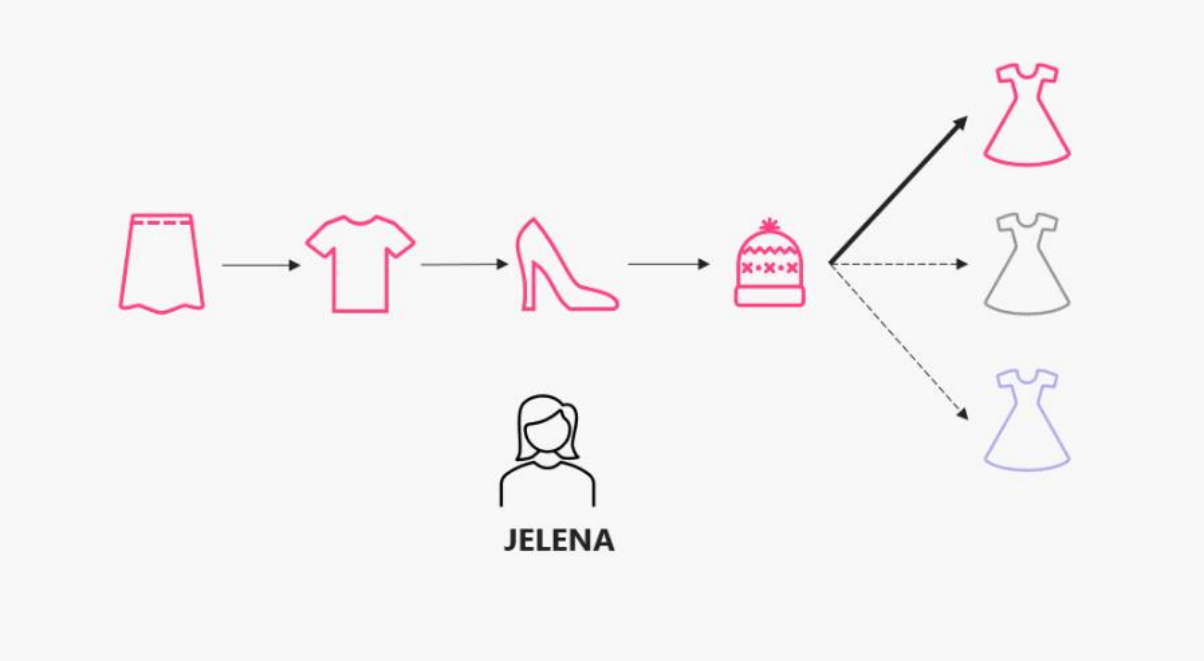
Content-based filtering se koristi u raznim sustavima preporuka, kao na primjer:

- Preporučivanje filmova i serija, na primjer Netflix
- Preporučivanje glazbe, Spotify
- Preporučivanje proizvoda, Amazon
- Preporučivanje članaka, novinarski portali

Koristi se kako bi se pružalo personalizirane preporuke na temelju korisničkih preferencija. Za razliku od collaborative filtering-a, ne treba podatke o ponašanju drugih korisnika, što je velika prednost. Relativno je jednostavno implementirati koristeći tekstualne podatke o stavkama.

Prednosti su što se ne oslanja se na ponašanje drugih korisnika. Ostali sustavi preporuka zahtijevaju takve podatke što može biti točnije i bolje, ali velika količina podataka je potrebno samo da se takvi algoritmi pokrenu prvi put. S takvom količinom podataka dolazi i do grešaka gdje podaci mogu biti netočni ili nedovoljni što programer ne može kontrolirati za razliku od ovih podataka koji su većinom jednaki. Pruža personalizirane preporuke na temelju specifičnih preferencija korisnika. Može preporučivati nove stavke koje još nisu ocijenjene (tzv. "cold start problem").

Neke od mana su to što može preporučivati samo slične stavke, što može rezultirati manjom raznolikošću preporuka. Potrebne su detaljne informacije o stavkama kako bi sustav bio učinkovit. Na primjer za svaki film je potrebno da sadrži što više glumaca, opširan opis, veći broj žanrova i slično. Ne može preporučivati stavke koje su izvan korisnikovih poznatih preferencija, što znači da je potrebno da je baza bogata podacima za stavku za koju se radi preporuka. Primjer kako se content-based filtering koristi za preporuku prikazan je na slici 8.



Slika 8. Content-Based filtering primjer

„def create_soup(row)“ funkcija je dizajnirana za stvaranje "soupe" (mješavine tekstualnih podataka) iz redaka tablice (obično u obliku Pandas DataFrame-a) koja sadrži informacije o filmovima ili sličnim entitetima. Ova funkcija kombinira dva specifična polja iz svakog retka u jedan tekstualni niz. Row je jedan redak iz Pandas DataFrame-a. Redak je u obliku Pandas Series objekta koji sadrži različite informacije o jednom filmu.

„return f"{row['genres']} {row['overview']}"“ stvara "soup" (mješavinu) podataka iz žanrova i sinopsisa filma. Spaja žanrove i sinopsis u jedan string koristeći f-string formatiranje.

Korištenjem formata f-string u Pythonu, moguće direktno uključiti vrijednosti iz stupaca unutar stringa.Ovaj string će se koristiti za vektorizaciju. Funkcija je prikazana na isječku.

```
def create_soup(row):
    return f"{row['genres']} {row['overview']}"
```

„def get_recommendation_dataframe()“ definira funkciju get_recommendation_dataframe koja ne prima argumente. Dohvaća sve filmove iz Django modela Movie, selektirajući samo polja movie_id, genres i overview. Vraća QuerySet s vrijednostima kao rječnike. Zatim pomoću df = pd.DataFrame(list(movies)), konvertira QuerySet u Pandas DataFrame. list(movies) pretvara QuerySet u listu rječnika, a pd.DataFrame pretvara tu listu u DataFrame. Zamjenjuje NaN vrijednosti praznim stringovima unutar DataFrame-a. Ovo je korisno za izbjegavanje problema tijekom vektorizacije jer inače Python vrati grešku ako ima takvih vrijednosti pa se ovim putem unaprijed to spriječi. Primjenjuje funkciju create_soup na svaki redak DataFrame-a kako bi stvorila novu kolonu soup koja sadrži spojene žanrove i sinopsise za svaki film. Na kraju „return df“ vraća pripremljeni DataFrame koji sadrži podatke o filmovima zajedno s generiranom "soup" kolonom.

Dakle funkcija get_recommendation_dataframe priprema podatke o filmovima za vektorizaciju. Nakon što je DataFrame pripremljen, možete koristiti CountVectorizer za pretvaranje "soup" kolone u numeričke vektore. Zatim možete koristiti cosine_similarity za izračunavanje sličnosti između filmova. Ovaj kod priprema podatke za content-based filtering koristeći podatke iz Django modela Movie i prikazan je na isječku.

```
def get_recommendation_dataframe():
    movies = Movie.objects.all().values("movie_id", "genres", "overview")
    df = pd.DataFrame(list(movies))
    df.fillna("", inplace=True)  # Replace NaN with empty string
    df["soup"] = df.apply(create_soup, axis=1)
    return df
```

Funkcija build_cosine_similarity priprema podatke o filmovima, vektorizira tekstualne podatke koristeći CountVectorizer, te izračunava kosinusnu sličnost između filmova. def build_cosine_similarity() definira funkciju build_cosine_similarity koja ne prima argumente. Poziva funkciju get_recommendation_dataframe koja dohvaća podatke o filmovima iz Django modela Movie, stvara DataFrame i generira kolonu soup koja sadrži spojene žanrove i sinopsise za svaki film.

„count = CountVectorizer(stop_words="english")“ inicijalizira CountVectorizer objekt. CountVectorizer je alat iz sklearn.feature_extraction.text modula koji pretvara tekstualne podatke u numeričke vektore koristeći tehniku vreće riječi (engl. bag-of-words). stop_words="english" argument specificira korištenje engleskih stop riječi koje će biti uklonjene tijekom vektorizacije, na primjer riječi koje su česte, ali ne dodaju značajnu informaciju, poput "the", "is", "in" i slično.

„count_matrix = count.fit_transform(df["soup"])“ koristi CountVectorizer za vektorizaciju kolone soup iz DataFrame-a df. fit_transform metoda uči vokabular iz tekstualnih

podataka i istovremeno transformira tekst u numeričke vektore. Rezultat je matrica count_matrix u kojoj su redci dokumenti (filmovi), a stupci značajke (riječi iz vokabulara), dok su vrijednosti broj pojavljivanja riječi u dokumentu.

Izračun kosinusne sličnosti se radi putem „ cosine_sim = cosine_similarity(count_matrix, count_matrix)“. Izračunava kosinusnu sličnost između redaka matrice count_matrix koristeći funkciju cosine_similarity iz sklearn.metrics.pairwise modula. Kosinusna sličnost je mjera sličnosti između dva vektora koja izračunava kosinus kuta između njih. Vrijednost sličnosti varira između 0 i 1, gdje je 1 maksimalna sličnost. Rezultat je matrica cosine_sim u kojoj su redci i stupci filmovi, a vrijednosti predstavljaju kosinusnu sličnost između odgovarajućih parova filmova.

Na kraju „return df, cosine_sim“ vraća DataFrame df i matricu kosinusne sličnosti cosine_sim. df sadrži informacije o filmovima zajedno s generiranom kolonom soup, a cosine_sim sadrži izračunate sličnosti između filmova.

Funkcija build_cosine_similarity priprema podatke o filmovima, vektorizira tekstualne podatke koristeći CountVectorizer, te izračunava kosinusnu sličnost između filmova. Ovdje se dohvaća podatke iz Django modela Movie i priprema DataFrame sa svim potrebnim informacijama. Ova funkcija je ključna za implementaciju content-based filtering sustava preporuka, jer omogućuje mjerenje sličnosti između filmova na temelju njihovih karakteristika. Prikazano na isječku.

```
def build_cosine_similarity():
    df = get_recommendation_dataframe()
    count = CountVectorizer(stop_words="english")
    count_matrix = count.fit_transform(df["soup"])
    cosine_sim = cosine_similarity(count_matrix, count_matrix)
    return df, cosine_sim
```

Funkcija get_movie_recommendations pruža preporuke filmova na temelju kosinusne sličnosti između filmova. Ako matrica sličnosti (cosine_sim) i DataFrame (df) nisu predani kao argumenti, funkcija će ih izgraditi koristeći build_cosine_similarity funkciju. Definira funkciju get_movie_ koja prima tri argumenta: movie_id koji je ID filma za koji se traže preporuke, cosine_sim je matrica kosinusne sličnosti (opcionalno) i df je DataFrame koji sadrži podatke o filmovima (također opcionalno).

„if cosine_sim is None or df is None“ provjerava jesu li cosine_sim ili df None. Ako bilo koji od njih nije predan kao argument, izgrađuje ih pozivajući build_cosine_similarity funkciju. „df, cosine_sim = build_cosine_similarity()“ poziva build_cosine_similarity kako bi izgradio DataFrame i matricu kosinusne sličnosti.

„indices“ stvara Pandas Series objekt gdje su indeksi movie_id vrijednosti, a vrijednosti su indeksi redova DataFrame-a. drop_duplicates() osigurava da nema duplikata movie_id vrijednosti. Ovo je dodatna provjera sigurnosti, ali u kodu za dohvaćanje filmova koristi se „update or create“ funkcija koja se brine o tome da ne bude diplikata. „idx“ varijabla dohvaća indeks retka u DataFrame-u koji odgovara zadanom movie_id. Ako movie_id nije pronađen, vraća None.

„if idx is None“ provjerava je li idx None (tj. ako movie_id nije pronađen u df). Ako je idx None, vraća prazan popis, jer nema preporuka za nepostojeći film.

Varijabla „sim_scores“ prvo izračunava kosinusnu sličnost između zadanog filma (idx) i svih ostalih filmova. enumerate stvara popis tuplova gdje je prvi element indeks filma, a drugi element je sličnost s zadanim filmom. Zatim sortira popis tuplova sim_scores po vrijednosti sličnosti u opadajućem redoslijedu (najprije najviše vrijednosti). Lambda funkcija osigurava da se sličnosti pravilno sortiraju, čak i ako su ugniježđene u listama ili nizovima. lamba funkcija unutar Python koda smanjuje broj linija ako je iteracija jednostavna kako kod ne bi bio nepotrebno opsežan. Na kraju uzima prvih 5 preporuka iz sortirane liste sličnosti, isključujući prvi element koji je zadani film.

„movie_indices“ Izvlači indekse filmova iz tuplova sim_scores. „recommendations“ koristi iloc za dohvaćanje redaka iz df koristeći movie_indices. Dohvaća movie_id vrijednosti tih redaka i pretvara ih u popis (tolist). Na kraju „return recommendations“ vraća popis preporučenih movie_id vrijednosti.

Funkcija get_movie_recommendations koristi se za dobivanje preporuka filmova na temelju kosinusne sličnosti između filmova. Funkcija prvo provjerava jesu li matrica sličnosti i DataFrame predani kao argumenti, a ako nisu, izgrađuje ih. Zatim dohvaća indeks zadanog filma, izračunava sličnosti s ostalim filmovima, sortira rezultate, filtrira prvih 5 preporuka (isključujući zadani film), i vraća movie_id vrijednosti preporučenih filmova. Ovaj pristup omogućuje efikasno preporučivanje filmova na temelju njihovih sličnosti, što je korisno za personalizaciju sadržaja i poboljšanje korisničkog iskustva u sustavima za preporuke. Prikazano na isječku ispod.

```
def get_movie_recommendations(movie_id, cosine_sim=None, df=None):
    if cosine_sim is None or df is None:
        df, cosine_sim = build_cosine_similarity()

    indices = pd.Series(df.index, index=df["movie_id"]).drop_duplicates()
    idx = indices.get(movie_id, None)

    if idx is None:
        return []

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(
        sim_scores, key=lambda x: x[1] if np.isscalar(x[1]) else x[1][0], reverse=True
    )

    sim_scores = sim_scores[1:6]
    movie_indices = [i[0] for i in sim_scores]
    recommendations = df.iloc[movie_indices]["movie_id"].tolist()
    return recommendations
```

Na slici 9 su prikazani rezultati unutar aplike za content-based preporuku filmova. Film za koji je korisnik htio preporuke je „Bad Boys: Ride or Die“ te za njega su se preporučilo 10 filmova koji su mu najsličniji. Na slici je vidljivo kako je čak ponuđeno prijašnje filmove u franšizi što je dobro jer se na taj način vizualno može vidjeti kako uzima u obzir parametre i dobro preporučuje. Također na kraju su prikazane metrike koje je algoritam dao gdje je preciznost 1.0 što znači da je rezultat sigurno dobar, ali „recall“ i „f1:score“ nisu dobri što znači

da je potrebno povećati obujam parametra, na primjer dodati glumce, redatelja i slično u preporuku.

```
Original Movie ID: 573435, Title: Bad Boys: Ride or Die, Genres: Action, Crime, Thriller, Comedy

-------------------------NEW CONTENT BASED-------------------------
Movie ID: 447200, Title: Skyscraper, Genres: Action, Thriller, Adventure
Movie ID: 160885, Title: Tel chi el telùn, Genres: Comedy
Movie ID: 1105407, Title: Damaged, Genres: Action, Crime, Thriller
Movie ID: 996154, Title: Black Lotus, Genres: Action, Crime, Mystery, Thriller
Movie ID: 9737, Title: Bad Boys, Genres: Action, Comedy, Crime, Thriller
Movie ID: 955555, Title: The Roundup: No Way Out, Genres: Action, Crime, Comedy, Thriller
Movie ID: 475557, Title: Joker, Genres: Crime, Thriller, Drama
Movie ID: 395990, Title: Death Wish, Genres: Action, Crime, Drama, Thriller
Movie ID: 339403, Title: Baby Driver, Genres: Action, Crime
Movie ID: 866398, Title: The Beekeeper, Genres: Action, Crime, Thriller
{'precision': 1.0, 'recall': 0.022321428571428572, 'f1_score': 0.043668122270742356}
```

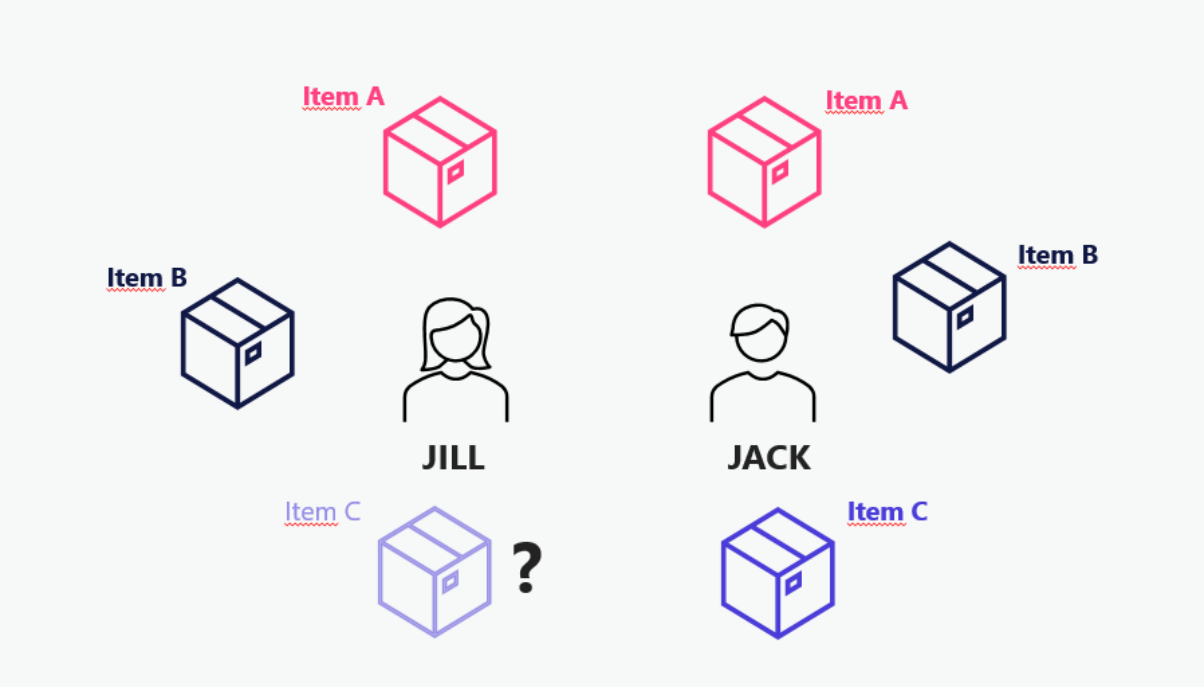
Slika 9. Rezultat algoritma

1.1.10. recommendations.py

Kolaborativno filtriranje(engl. Collaborative filtering) [17] (CF) je metoda preporučivanja koja se temelji na kolektivnom ponašanju i preferencijama korisnika. Ova metoda koristi podatke o interakcijama korisnika sa stavkama (na primjer filmovima, knjigama, proizvodima) kako bi preporučila nove stavke koje bi korisnicima mogle biti zanimljive. Drugim riječima, algoritmi kolaborativnog filtriranja grupiraju korisnike na temelju ponašanja i koriste opće karakteristike grupe za preporuku stavki ciljanom korisniku. Kolaborativni sustavi za preporučivanje djeluju na principu da slični korisnici (po ponašanju) dijele slične interese i slične ukuse.

Collaborative filtering je jedna od dviju glavnih vrsta sustava za preporučivanje, dok je druga filtriranje temeljeno na sadržaju(engl. Content-based). Ova potonja metoda koristi značajke stavki kako bi preporučila slične stavke onima s kojima je određeni korisnik pozitivno interagira u prošlosti. Dok kolaborativno filtriranje fokus stavlja na sličnost korisnika kako bi preporučilo stavke, filtriranje temeljeno na sadržaju preporučuje stavke isključivo prema značajkama profila stavki. Filtriranje temeljeno na sadržaju cilja preporuke na preferencije jednog specifičnog korisnika, a ne na grupu ili tip korisnika kao kod kolaborativnog filtriranja.

Primjer collaborative filtering-a prikazan je na slici 10.



Slika 10. Collaborative filtering primjer

Obje metode su u posljednjih nekoliko godina imale mnoge stvarne primjene, od e-trgovine poput Amazona do društvenih mreža i usluga streaminga. Zajedno, kolaborativni i sustavi temeljen na sadržaju tvore hibridne sustave za preporučivanje. Zapravo, Netflix je 2009. godine usvojio hibridni sustav za preporučivanje kroz svoj Netflix prize natječaj.

Collaborative filtering funkcionira tako da analizira obrasce ponašanja korisnika i pronalazi sličnosti između njih ili između stavki koje su korisnici ocijenili. Postoje dvije glavne vrste collaborative filtering-a: User-Based Collaborative Filtering i Item-Based Collaborative Filtering. Koristi matricu za mapiranje ponašanja korisnika za svaku stavku u svom sustavu. Sustav zatim izvlači vrijednosti iz te matrice kako bi ih prikazao kao podatkovne točke u vektorskom prostoru. Različite metrike zatim mjere udaljenost između točaka kao sredstvo za izračunavanje sličnosti između user-user te item-item.

U standardnom postavljanju collaborative filtering, imamo skup od n korisnika i skup od x stavki. Individualna preferencija svakog korisnika za svaku stavku prikazana je u user-item matrix (ponekad nazvana i matrica ocjena korisnika). Ovdje su korisnici predstavljeni u redovima, a stavke u stupcima. U matrici Rij, određena vrijednost predstavlja ponašanje korisnika u prema stavci i. Te vrijednosti mogu biti kontinuirani brojevi koje su korisnici pružili (na primjer ocjene) ili binarne vrijednosti koje označavaju je li određeni korisnik pogledao ili kupio stavku.

Na dijagramu ispod slike 11, svaka kvačica označava film koji je određeni korisnik pogledao. Treći i četvrti korisnik imaju preferencije koje su dobro objašnjene ovim značajkama - treći korisnik preferira filmove za djecu, dok četvrti korisnik preferira filmove za odrasle. Međutim, preferencije prvog i drugog korisnika nisu dobro objašnjene ovim jednim značajkama.

Slika 11. Primjer preporuke filmova

U ovakvom scenariju(vidljivo u tablici 1), možemo vidjeti da su ocjene Korisnika 1 i Korisnika 2 gotovo slične za filmove, pa možemo zaključiti da će Film 3 također biti prosječno preporučen Korisniku 1, ali će Film 4 biti dobra preporuka za Korisnika 2. Na sličan način, moguće je vidjeti da postoje korisnici s različitim izborima, poput Korisnika 1 i Korisnika 3 koji su suprotni jedno drugome. Također se može vidjeti da Korisnik 3 i Korisnik 4 imaju zajednički interes za filmove, pa na temelju toga možemo reći da se Film 4 također neće svidjeti Korisniku 4. Ovo je primjer collaborative filtering-a, gdje se preporučuje korisnicima stavke koje su se svidjele korisnicima sličnih interesnih domena.

Tablica 1. Collaborative filtering preporuka

Users	Movie 1	Movie 2	Movie 3	Movie 4
User 1	5	4		5
User 2	4		3	
User 3	1			2
User 4		1		

U procesu normalizacije, uzima se prosječnu ocjenu korisnika i oduzimamo sve dane ocjene od nje, tako se dobije pozitivne ili negativne vrijednosti kao ocjene, koje se mogu jednostavno dalje klasificirati u slične skupine. Normalizacijom podataka može se formirati klastere korisnika koji daju slične ocjene sličnim stavkama, a zatim se može koristiti te klastere za preporuku stavki korisnicima.

Item-based filtering [18] preporučuje nove stavke ciljanom korisniku na temelju ponašanja tog korisnika prema sličnim stavkama. Na primjer, u sustavu za preporuku filmova, algoritam može identificirati slične filmove na temelju korelacija između ocjena svih korisnika za svaki film. Sustav će zatim preporučiti novi film ciljanom korisniku na temelju koreliranih ocjena. Ako je ciljani korisnik visoko ocijenio film a i film b, ali nije gledao film c, a drugi korisnici koji su visoko ocijenili prva dva filma također su visoko ocijenili film c, sustav će preporučiti film c ciljanom korisniku. Na ovaj način, item-based filtering izračunava sličnost stavki kroz ponašanje korisnika. Funkcije sličnosti temeljene na stavkama izračunavaju se između stupaca u user-item matrici. Prednost je lakše skaliranje za velike skupove stavki i može biti učinkovitiji kod rijetkih matrica korisnika-stavki. Mana je što može biti manje personaliziran u usporedbi s User-Based pristupom i preporučivati previše sličnih stavki, što smanjuje raznolikost preporuka.

Collaborative filtering se koristi u sustavima preporuka gdje je važno personalizirati sadržaj za korisnike na temelju njihovih prethodnih interakcija. Najčešće se koristi u:

- Preporučivanje filmova i serija
- Preporučivanje glazbe
- Preporučivanje proizvoda
- Preporučivanje knjiga

Koristi se zbog toga što pruža personalizirane preporuke na temelju stvarnog ponašanja korisnika. Koristi podatke mnogih korisnika kako bi donijela preciznije preporuke. Može se implementirati koristeći razne algoritme i tehnike strojnog učenja.

Prednosti su je ovaj algoritam vrlo učinkovit u personaliziranju preporuka na temelju ponašanja korisnika. Ne zahtijeva detaljne informacije o stavkama, samo podatke o interakcijama korisnika. Interakcije mogu biti kupovina artikla, ocjenjivanje stavka pa čak i ponašanje korisnika prilikom korištenja aplikacije, kao na primjer na koje stavke klikne i koliko vremenski provodi na pojedinoj stranici. Može se skalirati za velike skupove podataka koristeći distribuirane sustave.

Mane je to što je teško je preporučiti stavke novim korisnicima ili nove stavke jer nema dovoljno podataka o njima. Može favorizirati popularne stavke i manje preporučivati manje popularne stavke. Za učinkovite preporuke potrebna je velika količina povijesnih podataka o korisničkim interakcijama.

Kada se koristi User-Based Collaborative Filtering i kada Item-Based Collaborative Filtering je prikazano u tablici 2.

Tablica 2. Usporedba collaborative filtering metoda

User-Based Collaborative Filtering	Item-Based Collaborative Filtering
Kada ima relativno manji broj korisnika, a veliki broj stavki.	Kada ima veliki broj korisnika i veliki broj stavki.

Kada ima bogate podatke o korisnicima i njihovim interakcijama sa stavkama.	Kada ima rijetku matricu korisnika-stavki i želite izbjegavati problem sparsnosti.
Kada je potrebno imati visoko personalizirane preporuke.	Kada je skalabilnost ključna i potrebne su brze preporuke.

Collaborative filtering je moćan alat za preporučivanje stavki na temelju kolektivnog ponašanja korisnika. User-Based Collaborative Filtering fokusira se na sličnosti između korisnika, dok Item-Based Collaborative Filtering fokusira se na sličnosti između stavki. Svaki pristup ima svoje prednosti i mane, a izbor između njih ovisi o specifičnim zahtjevima sustava preporuka, dostupnim podacima i potrebama za skalabilnost i personalizaciju.

ALS (Alternating Least Squares) je modul iz PySpark-a, koristi se za kreiranje modela preporuka baziranih na matrici korisnika i stavki. ALS je popularan algoritam za collaborative filtering koji se temelji na faktorizaciji matrice.

SparkSession je modul koji se koristi za kreiranje Spark sesije, koja je ulazna točka za sve funkcionalnosti Spark SQL-a. SparkSession zamjenjuje starije ulazne točke kao što su SQLContext i HiveContext.

Modul apps koristi se za dohvaćanje aplikacija i njihovih modela unutar Django projekta. Omogućava dinamički pristup modelima bez direktnog uvoza.

models je generički uvoz svih modela definiranih u lokalnom modulu models.py. Omogućava pristup svim Django modelima u aplikaciji.

Funkcija cosine_similarity iz sklearn.metrics.pairwise modula koristi se za izračunavanje kosinusne sličnosti između redaka matrice. Koristi se u sustavima preporuke za mjerenje sličnosti između korisnika ili stavki.

numpy je popularna biblioteka za znanstveno računarstvo u Pythonu. Koristi se za rad s višedimenzionalnim nizovima i matricama, kao i za provođenje matematičkih operacija nad tim nizovima. Ova biblioteka je jedna od bitnijih u Python-u općenito.

prepare_data_scikit je funkcija koja priprema podatke za korisnički orijentirane preporuke koristeći PySpark i Pandas. Funkcija koristi Spark za manipulaciju podacima i stvaranje pivot tablice, te pretvara rezultat u Pandas DataFrame za daljnju obradu s bibliotekama kao što su scikit-learn. Kreira Spark sesiju koristeći SparkSession.builder. appName("User Based Recommendations") postavlja naziv aplikacije u Spark UI-u. getOrCreate() kreira novu Spark sesiju ili dohvaća postojeću ako je već kreirana.

Dohvaća Django model MovieRating iz aplikacije moviesApp koristeći apps.get_model metodu. Ovo omogućava dinamički pristup modelima bez direktnog uvoza. Zatim dohvaća sve zapise iz modela MovieRating i pretvara ih u listu rječnika. Svaki rječnik sadrži ključeve user_id, movie_id i rating. Kreira Spark DataFrame iz liste rječnika data koristeći createDataFrame metodu. Ovaj DataFrame sadrži tri stupca: user_id, movie_id i rating.

Na varijabli „pivot_df“ grupira podatke prema user_id koristeći groupBy metodu. koristi pivot("movie_id") kako bi transformirao movie_id stupac u više stupaca, gdje svaki stupac predstavlja jedan film. koristi avg("rating") kako bi izračunao prosječnu ocjenu za svaki film po korisniku. koristi fillna(0) kako bi zamijenio NaN vrijednosti s 0. Ovo je korisno jer mnogi korisnici neće imati ocjene za sve filmove, a preporučiteljski sustavi obično zahtijevaju popunjenu matricu.

Zatim pretvara Spark DataFrame pivot_df u Pandas DataFrame koristeći toPandas metodu. Ovo omogućava daljnju obradu podataka koristeći Pandas i druge biblioteke u Pythonu kao što je scikit-learn. Nakon čega zaustavlja Spark sesiju koristeći stop metodu. Ovo oslobađa resurse koje koristi Spark. Na kraju vraća pripremljeni Pandas DataFrame pandas_df. Ovaj DataFrame sadrži korisničke ocjene za filmove u obliku pivot tablice, gdje su redci korisnici, stupci filmovi, a vrijednosti su ocjene.

Ovaj pripremljeni Pandas DataFrame može se dalje koristiti za izračunavanje sličnosti, treniranje modela preporuka ili druge analize koristeći biblioteke kao što su scikit-learn. Prednosti korištenja Spark-a u ovom procesu uključuju mogućnost skaliranja obrade podataka na velike skupove podataka, dok Pandas omogućava jednostavniju manipulaciju i analizu podataka u Pythonu. Prikazano na isječku.

```
def prepare_data_scikit():
    spark = SparkSession.builder.appName("User Based Recommendations").getOrCreate()

    MovieRating = apps.get_model(“MoviesApp“, "MovieRating")
    data = list(MovieRating.objects.all().values("user_id", "movie_id", "rating"))
    df = spark.createDataFrame(data)

    pivot_df = df.groupBy("user_id").pivot("movie_id").avg("rating").fillna(0)

    pandas_df = pivot_df.toPandas()
    spark.stop()

    return pandas_df
```

Funkcija user_based_filtering_scikit koja prima dva argumenta: user_id koji je ID korisnika za kojeg se generiraju preporuke i dana što je Pandas DataFrame koji sadrži ocjene korisnika za različite filmove.

Varijabla „similarity_matrix“ koristi cosine_similarity funkciju iz scikit-learn-a za izračunavanje kosinusne sličnosti između svih korisnika u skupu podataka. data bi trebao biti Pandas DataFrame gdje su redci korisnici, stupci filmovi, a vrijednosti su ocjene. Rezultat je kvadratna matrica similarity_matrix gdje je svaki element (i, j) sličnost između korisnika i i korisnika j.

„user_index“ varijabla prvo pronalazi indeks redka u DataFrame-u data koji odgovara zadanom user_id. data["user_id"] == user_id stvara booleovsku seriju koja je True za redak gdje user_id odgovara zadanom user_id. data.index[data["user_id"] == user_id] vraća indeks(e) redka gdje je serija True. tolist() pretvara indeks u listu. [0] dohvaća prvi (i jedini) element liste, što je indeks retka za korisnika user_id.

„similar_users“ služi za pronalaženje najsličnijih korisnika zadanom korisniku. similarity_matrix[user_index] dohvaća redak matrice sličnosti koji odgovara zadanom

korisniku. np.argsort(-similarity_matrix[user_index]) sortira indekse korisnika prema sličnosti u opadajućem redoslijedu (zato koristimo - za negativne vrijednosti). [:5] uzima prvih 5 najsličnijih korisnika.

Na kraju „recommendations“ varijabla generira preporuke na temelju ocjena najsličnijih korisnika. data.iloc[similar_users] dohvaća retke iz DataFrame-a data koji odgovaraju indeksima najsličnijih korisnika. „.sum()“ zbraja ocjene za svaki film preko najsličnijih korisnika. „.sort_values(ascending=False)“ sortira zbrojene ocjene u padajućem redoslijedu, stavljajući filmove s najvećim ocjenama na vrh. Zatim putem return vraća generirane preporuke kao Pandas seriju, gdje su indeksi nazivi filmova, a vrijednosti zbrojene ocjene.

Funkcija user_based_filtering_scikit implementira user-based collaborative filtering koristeći kosinusnu sličnost za izračunavanje sličnosti između korisnika. Funkcija prvo izračunava matricu sličnosti, pronalazi indeks zadanog korisnika, pronalazi najsličnije korisnike, te generira preporuke na temelju ocjena tih korisnika. Rezultat je sortirana serija preporuka, gdje su filmovi s najvećim zbrojenim ocjenama na vrhu. Ova metoda omogućuje personalizirane preporuke na temelju sličnosti korisnika, koristeći scikit-learn za učinkovito izračunavanje sličnosti i Pandas za manipulaciju podacima. Prikazano na isječku.

```
def user_based_filtering_scikit(user_id, data):
    similarity_matrix = cosine_similarity(data)
    user_index = data.index[data["user_id"] == user_id].tolist()[0]

    similar_users = np.argsort(-similarity_matrix[user_index])[:5]

    recommendations = data.iloc[similar_users].sum().sort_values(ascending=False)

    return recommendations
```

1.1.11. fake_data.py

Ovaj kod koristi biblioteku Faker [19] za generiranje lažnih podataka i koristi TMDB API za dohvaćanje popularnih filmova. Uvozi biblioteku Faker koja se koristi za generiranje lažnih podataka, sve modele iz aplikacije moviesApp, biblioteku random koja se koristi za generiranje slučajnih brojeva, model User iz Django-ovog modula za upravljanje korisnicima, klasu date iz modula datetime za rad s datumima te na kraju biblioteku requests koja se koristi za slanje HTTP zahtjeva.

Funkcija generate_users koristi se za generiranje određenog broja lažnih korisnika koristeći biblioteku Faker i Django-ov User model. Ovo se koristi zbog toga što je lakše generirati lažne korisnik nego da se ručno kreira svaki korisnik i smišljaju podaci. Definira funkciju generate_users koja prima jedan argument num, što predstavlja broj korisnika koji će biti generirani. Pokreće petlju koja će se izvršiti num puta. Ovdje num predstavlja broj korisnika koje želimo generirati. Korištenje _ kao varijable unutar petlje označava da nije važna trenutna vrijednost brojila petlje, već samo broj iteracija. Poziva metodu create_user iz Django-ovog User modela za kreiranje novog korisnika s generiranim podacima. Generira lažno korisničko ime koristeći metodu user_name() iz Faker objekta fake. Generira lažnu lozinku koristeći metodu password() iz Faker objekta fake. Generira lažno ime koristeći metodu first_name() iz Faker objekta fake. Generira lažno prezime koristeći metodu last_name() iz

Faker objekta fake. Generira lažnu email adresu koristeći metodu email() iz Faker objekta fake. Poziva metodu save() na objektu user kako bi spremio novog korisnika u bazu podataka. Iako metoda create_user već automatski sprema korisnika, pozivanje save() dodatno osigurava da su svi podaci pohranjeni. Ako je potrebno kreirati 10 lažnih korisnika poziva se funkcija komandom „generate_users(10)“ unutar Python shell. Kod je prikazan na isječku.

```
from faker import Faker
from moviesApp.models import *
import random
from django.contrib.auth.models import User
from datetime import date
import requests

fake = Faker()

def generate_users(num):
    for _ in range(num):
        user = User.objects.create_user(
            username=fake.user_name(),
            password=fake.password(),
            first_name=fake.first_name(),
            last_name=fake.last_name(),
            email=fake.email(),
        )
        user.save()
```

Funkcija generate_ratings generira ocjene za filmove koristeći TMDB API za dohvaćanje podataka o filmovima i Faker biblioteku za generiranje lažnih ocjena i recenzija. generate_ratings koja prima jedan argument num_ratings, što predstavlja broj ocjena koje će biti generirane. Poziva funkciju get_popular_movie_ids s api_key kako bi dobila popis popularnih filmova.

Pokreće petlju koja će se izvršiti num_ratings puta. Pokreće beskonačnu petlju koja će se prekinuti kad se uspješno generira ocjena. Nasumično odabire movie_id iz popisa movie_ids. Nasumično generira user_id između 1 i 31, zbog toga što imamo uvid koliko korisnika ima pa je moguće znati gornji limit. Pokušava dohvatiti korisnika s user_id, dohvaća korisnika iz baze podataka i ako korisnik ne postoji, petlja se nastavlja. Zatim provjerava postoji li već ocjena za odabrani film od strane korisnika, ispisuje poruku ako korisnik već ima ocjenu za film i nastavlja petlju s novim movie_id ako ocjena već postoji. Provjerava ima li film sinopsis i žanrove, pretvara popis žanrova u CSV format, pretvara popis ključnih riječi u CSV format, dohvaća ime redatelja i na kraju pretvara popis glavnih glumaca u CSV format. Generira slučajnu ocjenu između 1 i 10 zbog toga što je to potrebno za preporuku filmova kako bi se osiguralo da su ocjene pozitivni brojevi s maksimalnom ocjenom 10. Generira lažni tekst recenzije s najviše 200 znakova. Generira slučajni datum između 1. siječnja 2024. i današnjeg datuma. Kreira ili ažurira ocjenu za film i korisnika zato što ako ocjena postoji neće se nepotrebno kreirati nova za isti film što je moguće da sam korisnik tako napravi, ali za lažne podatke je nepotrebno. Izlazi iz beskonačne petlje nakon uspješnog generiranja ocjene. Ako film nema sinopsis ili žanrove, ispisuje odgovarajuću poruku i ne sprema ocjenu i na isti način radi ostale provjere i ispise poruka. Prikazano na isječku.

```
def generate_ratings(num_ratings):
    api_key = "5db79aa066a2a3d49ea80f0641bdfd67"
```

```
            movie_ids = get_popular_movie_ids(api_key)

            for _ in range(num_ratings):
                while True:
                    movie_id = random.choice(movie_ids)
                    user_id = random.randint(1, 31)

                    try:
                        user = User.objects.get(id=user_id)
                    except User.DoesNotExist:
                        continue

                    if MovieRating.objects.filter(user=user, movie_id=movie_id).exists():
                        print(
                            f"User {user_id} has already rated movie ID {movie_id}, selecting
a new movie."
                        )
                        continue

                    movie_url                                                          =
f"https://api.themoviedb.org/3/movie/{movie_id}?api_key={api_key}&append_to_response=keywor
ds,genres,credits"
                    response = requests.get(movie_url)
                    if response.status_code == 200:
                        movie = response.json()
                        overview = movie.get("overview", "")
                        genres = movie.get("genres", [])
                        if overview and genres:
                            genres_csv = ",".join([genre["name"] for genre in genres])
                            keywords_csv = ",".join(
                                [
                                    keyword["name"]
                                    for keyword in movie.get("keywords", {}).get("keywords",
[])
                                ]
                            )
                            director = next(
                                (
                                    member["name"]
                                    for member in movie.get("credits", {}).get("crew", [])
                                    if member["job"] == "Director"
                                ),
                                None,
                            )
                            actors = ",".join(
                                actor["name"]
                                for actor in movie.get("credits", {}).get("cast", [])[:5]
                            )

                            rating = random.randint(1, 10)
                            review_text = fake.text(max_nb_chars=200)
                            created_at = fake.date_between(
                                start_date=date(2024, 1, 1), end_date=date.today()
                            )

                            MovieRating.objects.update_or_create(
                                user=user,
                                movie_id=str(movie_id),
                                defaults={
                                    "rating": rating,
                                    "review_text": review_text,
                                    "created_at": created_at,
                                    "genres": genres_csv,
                                    "keywords": keywords_csv,
                                    "director": director,
```

```
                                "actors": actors,
                                "overview": overview,
                            },
                        )
                        break
                    else:
                        if not overview:
                            print(
                                f"Movie ID {movie_id} has no overview and will not be
saved."
                            )
                        if not genres:
                            print(
                                f"Movie ID {movie_id} has no genres and will not be saved."
                            )
                else:
                    print(f"Movie ID {movie_id} not found in TMDB or API error.")
                    break
```

3. Korištenje aplikacije

“MoviesApp“ je ime ove Django aplikacije koja korisnicima nudi preporuke filmova i omogućuje ocjenjivanje filmova. Naslovna stranica aplikacije pruža osnovne informacije o aplikaciji i njenim funkcionalnostima te sadrži navigacijske elemente za jednostavan pristup različitim dijelovima aplikacije.

Navigacijska traka prikazuje Naslov (“MoviesApp“) što je oznaka aplikacije smještena u gornjem lijevom kutu, koja pruža identitet aplikacije. Navigacijski linkovi su tekstovi koji vode od pojedine stranice unutar aplikacije sa posebnim dizajnom putem CSS kako ne bi izgledalo kao običan tekst. Naslovna je link koji vodi na glavnu stranicu aplikacije, što je i prikazano na slici 12. Lista Filmova je link koji vodi na stranicu s popisom filmova dostupnih za pregled i ocjenjivanje. Login je link koji omogućuje korisnicima prijavu na njihov korisnički račun. Nakon što se kreira korisnički račun, gumb se pretvori u „Logout“ s korisničkim imenom i dodaju se dodatni gumbovi na sučelje.

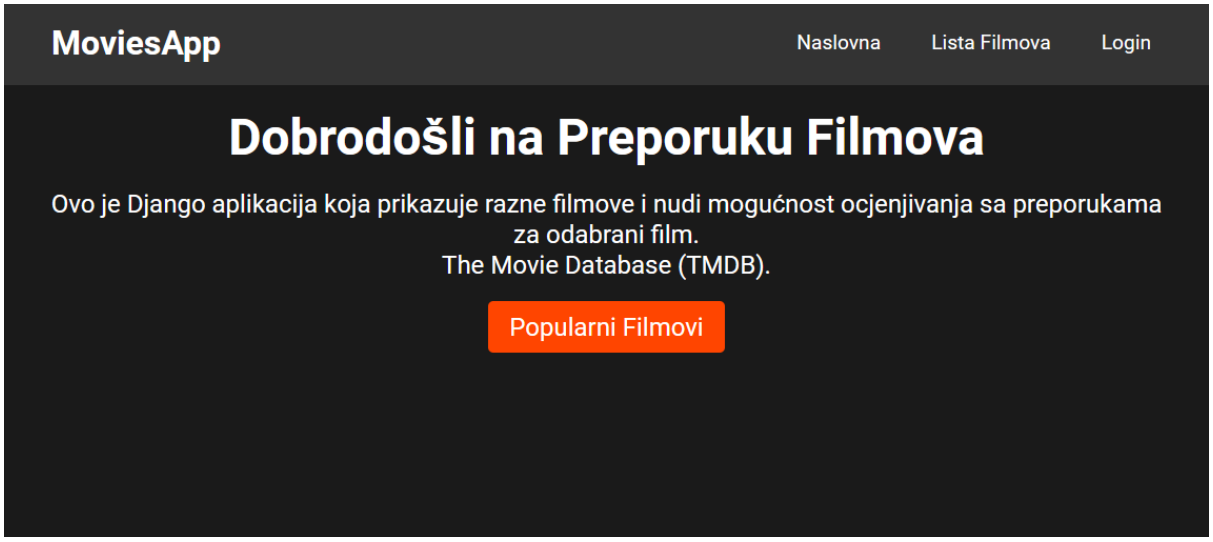
Poruka dobrodošlice gdje veliki naslov "Dobrodošli na Preporuku Filmova" odmah privlači pažnju korisnika.

Informativni paragraf ispod naslova objašnjava svrhu aplikacije, "Ovo je Django aplikacija koja prikazuje razne filmove i nudi mogućnost ocjenjivanja sa preporukama za odabrani film. The Movie Database (TMDB)." Ovo je samo naznaka čemu aplikacija služi pošto nije na produkciji pa se koristi ovaj tekst privremeno za prikaz elemenata.

Istaknuti narančasti gumb vodi korisnike na stranicu s popularnim filmovima. Gumb je centralno pozicioniran kako bi bio lako uočljiv i dostupan za interakciju. Moguće je posjetiti stranicu s popularnim filmovima ovim gumbom, ali također i gumbom u navigacijskoj traci. Izrađeno je na ovaj način kako bi korisnik imao odmah priliku korištenja najbitnije djela aplikacije, kako i rade ostale velike aplikacije na internetu.

Pozadina stranice je tamne boje, što omogućuje bolji kontrast i čitljivost bijelog teksta. Također u današnje doba često se koristi noćna tema(engl. night theme ili night mode) što je bio još jedan od razloga za korištenje ovakvih boja. Tekst je bijele boje, što ga čini lako čitljivim na tamnoj pozadini. Gumb za popularne filmove je narančaste boje, što dodaje vizualni kontrast i privlači pažnju korisnika.

Naslovna stranica je implementirana pomoću HTML-a i CSS-a unutar Django frameworka. Ključni elementi dizajna i funkcionalnosti definirani su unutar Django template-a, dok su stilovi definirani u zasebnoj CSS datoteci kako bi se postigla konzistentnost i preglednost dizajna. Neke od funkcija kako da se elementi ponašaju interakcijom korisnika ili u pozadini što se dešava, koristi se Javascript i Python za implementaciju. Naslovna stranica prikazan je na slici 12.



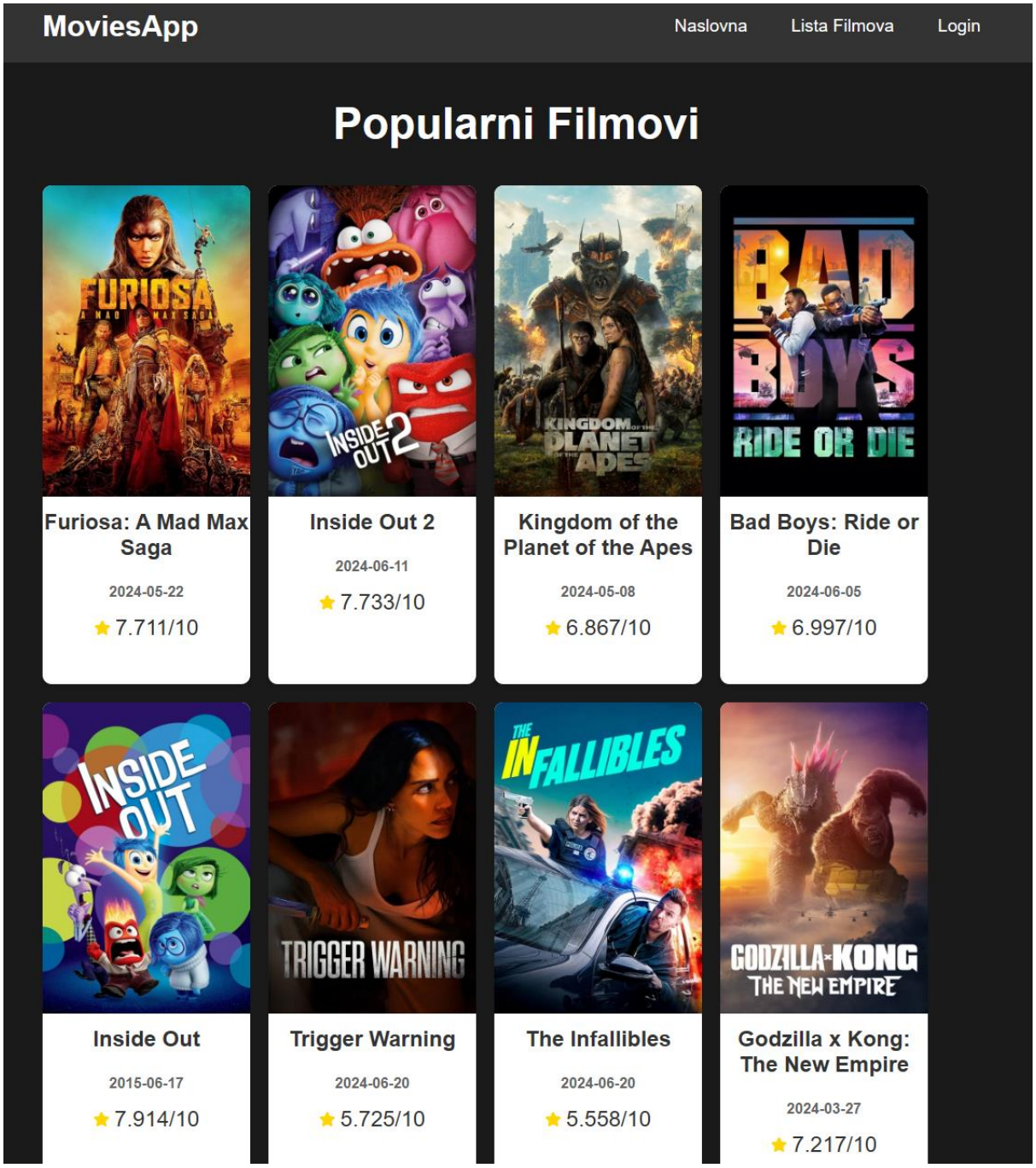
Slika 12. Naslovna stranica aplikacije

Ova stranica unutar aplikacije “MoviesApp“ prikazuje popularne filmove. Svi filmovi s informacijama dohvaćeni su putem API poziva i prikazani na stranici. Navigacijska traka ostaje ista kao i na naslovnoj stranici.

Veliki naslov "Popularni Filmovi" jasno prikazuje svrhu stranice kako bi korisnik znao koja se vrsta filmova prikazuje. Na prvu moguće je vidjeti filmove koji su aktualni, bilo to u kinima ili najgledaniji općenito, na primjer putem nekih od streaming platforma.

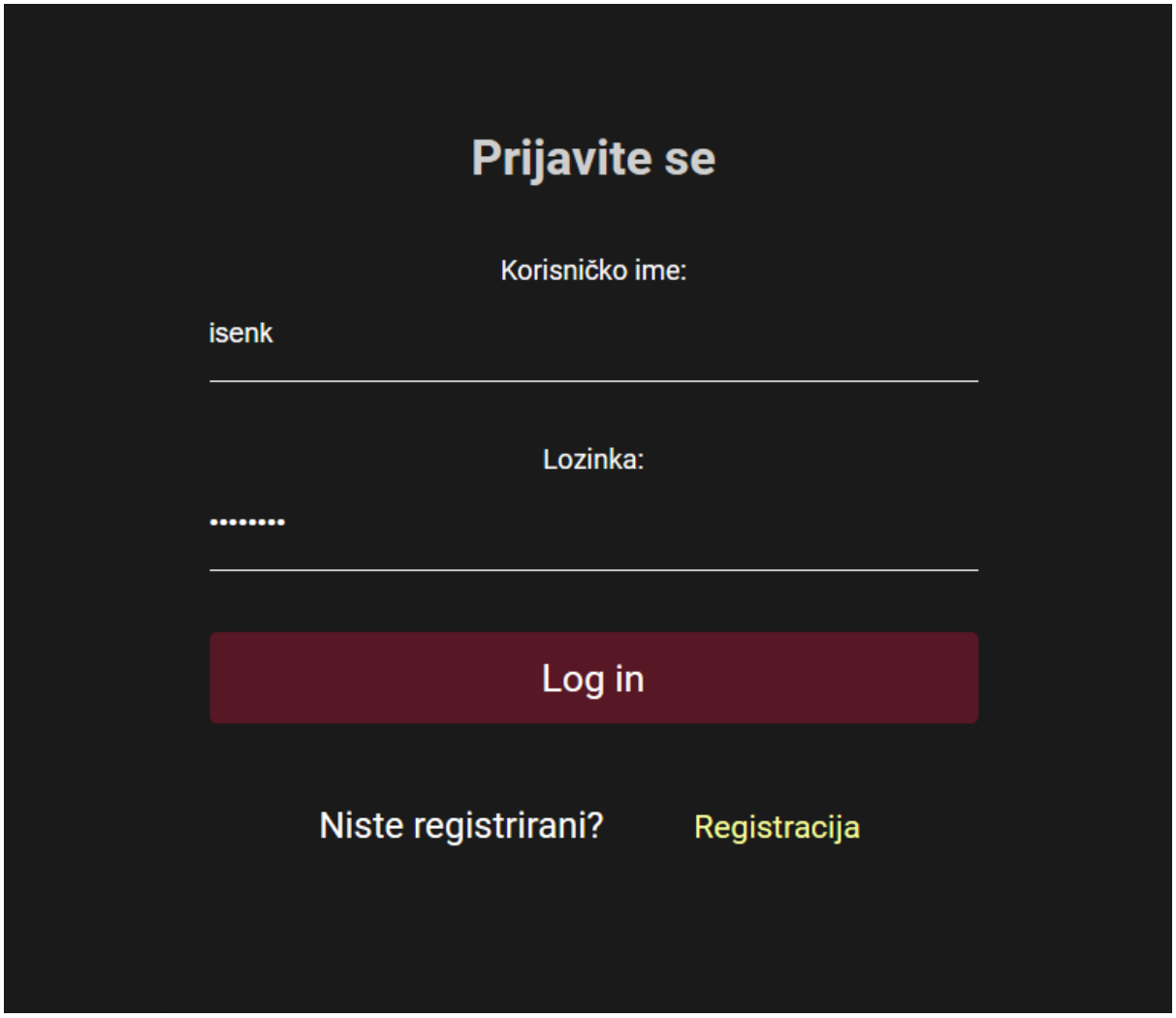
Filmske kartice prikazuju vizualni prikaz svakog filma. Prikazuju vizualni prikaz svakog filma. Ispod svakog plakata nalazi se naziv filma. Prikazuje također datum izlaska filma kako bi korisnik imao dojam ako je film popularan jer je nov ili jednostavno ima dobar broj gledatelja. Prikazuje prosječnu ocjenu filma, koristeći zlatnu zvjezdicu za vizualnu prezentaciju. Ova se ocjena prikuplja putem API-ja te nije prosječna ocjena korisnika aplikacije jer je ta ocjena rezervirana za pojedinog korisnika i njihove preporuke filmova.

Pozadina i ostali elementi su isti kao i na naslovnoj stranici s time da pozadine kartica su bijele boje kako bi se pojedina kartica zasebno istaknula, kao što je moguće vidjeti na slici 13. "Lista Filmova" pruža korisnicima pregled popularnih filmova s ključnim informacijama kao što su naslov, datum izlaska i ocjena. Korisnik može pristupiti pojedinom filmu klikom na sliku ili naslov kako bi lakše navigirao, što se također inače koristi na ostalim aplikacijama.



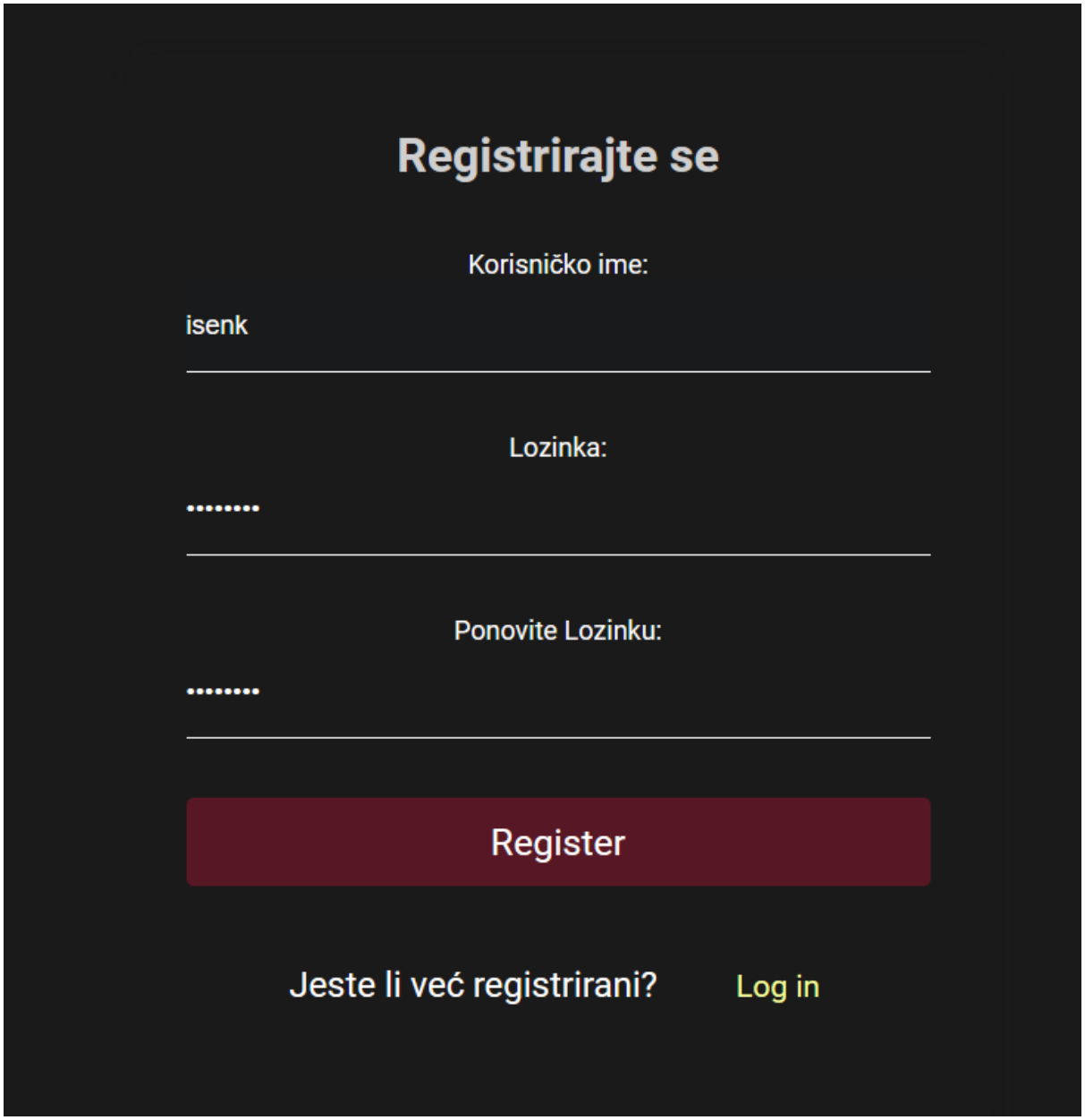
Slika 13. Prikaz popularnih filmova

Ova stranica unutar aplikacije “MoviesApp“ omogućuje korisnicima prijavu i registraciju na njihov korisnički račun. "Korisničko ime:“ je input polje gdje korisnik unosi svoje korisničko ime koje je kreirao tijekom registracije. "Lozinka:" je input polje gdje korisnik unosi svoju lozinku. Lozinka je prikazana kao skriveni tekst (točkice) radi sigurnosti. Na kraju kad je korisnik siguran da je unio točne podatke pruža se crveni gumb "Log in" za potvrdu unosa korisničkog imena i lozinke te prijavu na račun. "Niste registrirani? Registracija" je pitanje i link koji omogućuje korisnicima koji nemaju račun da se registriraju. Link je označen žutom bojom kako bi bio uočljiv i različit od prijave. Prozor za prijavi prikazan je na slici 14.



Slika 14. Prozor prijave korisnika

Na isti način je prikazana i stranica koja nudi registraciju korisnika. Izgled je identičan onome za prijavu samo što se razlikuje dodatno polje za ponovno upisivanje lozinke kako bi se potvrdilo i osiguralo da je korisnik unio točnu željenu lozinku. Na ovome se prozoru za lozinku gleda ako je jedno slovo veliko, ako ima ostalih znakova i slično radi sigurnosti. Ova provjera je dio Djanga te ju je moguće isključiti ili čak učiti još kompleksnijom ako je potrebno. Na kraju klikom na gumb „Register“ korisnik kreira svoj korisnički profil ili se vratiti na prethodni „Logi in“ prozor. Proces registracije prikazan je na slici 15.

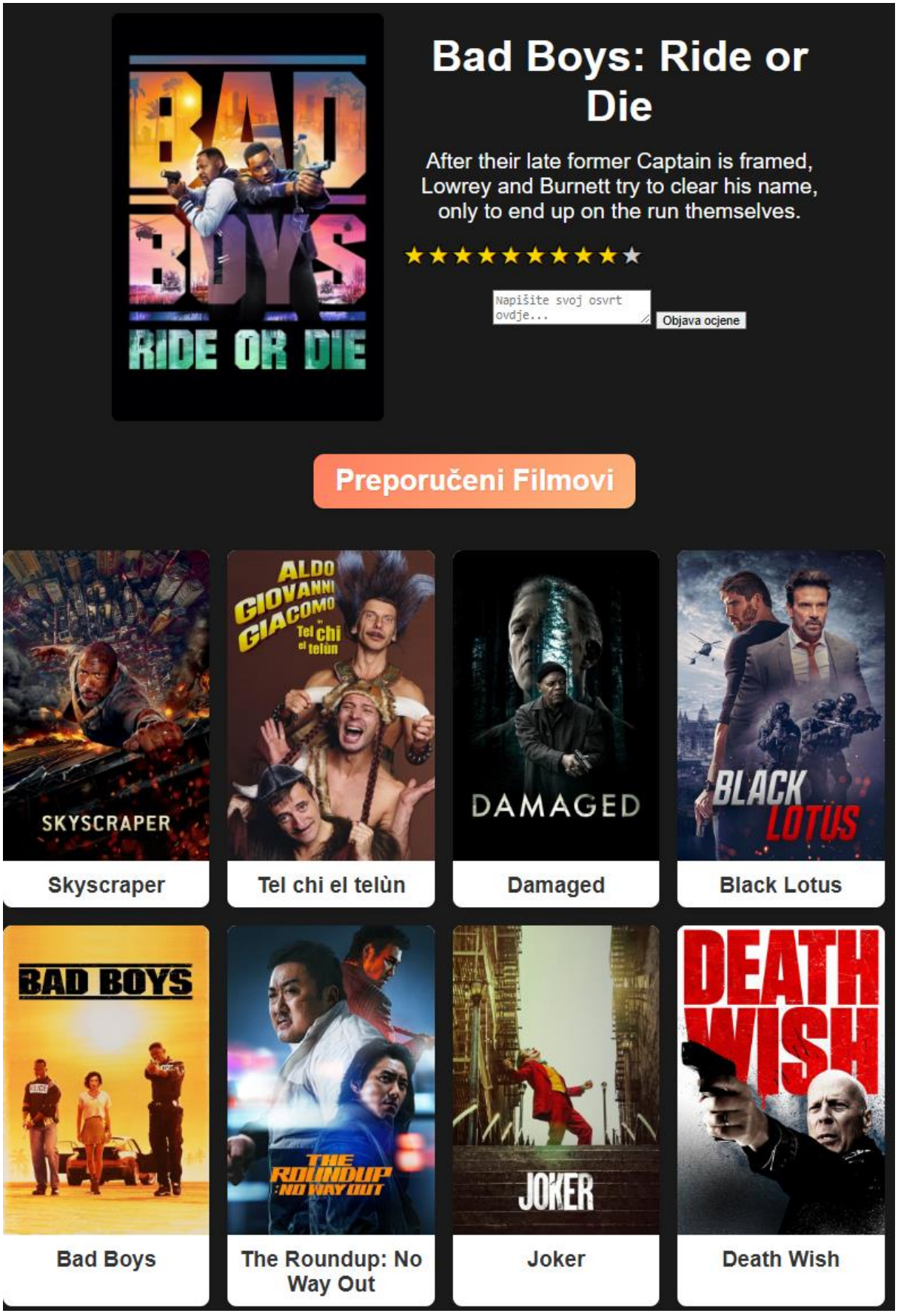


Slika 15. Prozor registracije korisnika

Ova stranica unutar aplikacije “MoviesApp“ pruža detalje o pojedinom filmu, omogućuje ocjenjivanje filma i prikazuje preporučene filmove na temelju odabranog filma. Vizualni prikaz plakata filma s lijeve strane. „Bad Boys: Ride or Die“: Veliki naslov filma koji je podebljan kako bi se istaknuo. Kratki opis radnje filma kako bi korisnik imao više informacija o filmu te mogao otprilike znati ako po opisu ima sličan preporučeni film koji bi ga zanimao. Ocjena film je prikazana kao niz zlatnih zvjezdica s ocjenom pored. Ocjenjivanje se radi tako što korisnik pomiče zvjezdice od 1 do 10 kako bi imao interaktivno iskustvo umjesto da samo upiše brojku. Tekstualno polje za osvrt je ponuđeno gdje korisnici mogu napisati svoj osvrt na film, ali nije potrebno da se uvijek unese vrijednost. Na kraju gumb za objavu ocjene omogućava korisnicima da pošalju svoju ocjenu i osvrt.

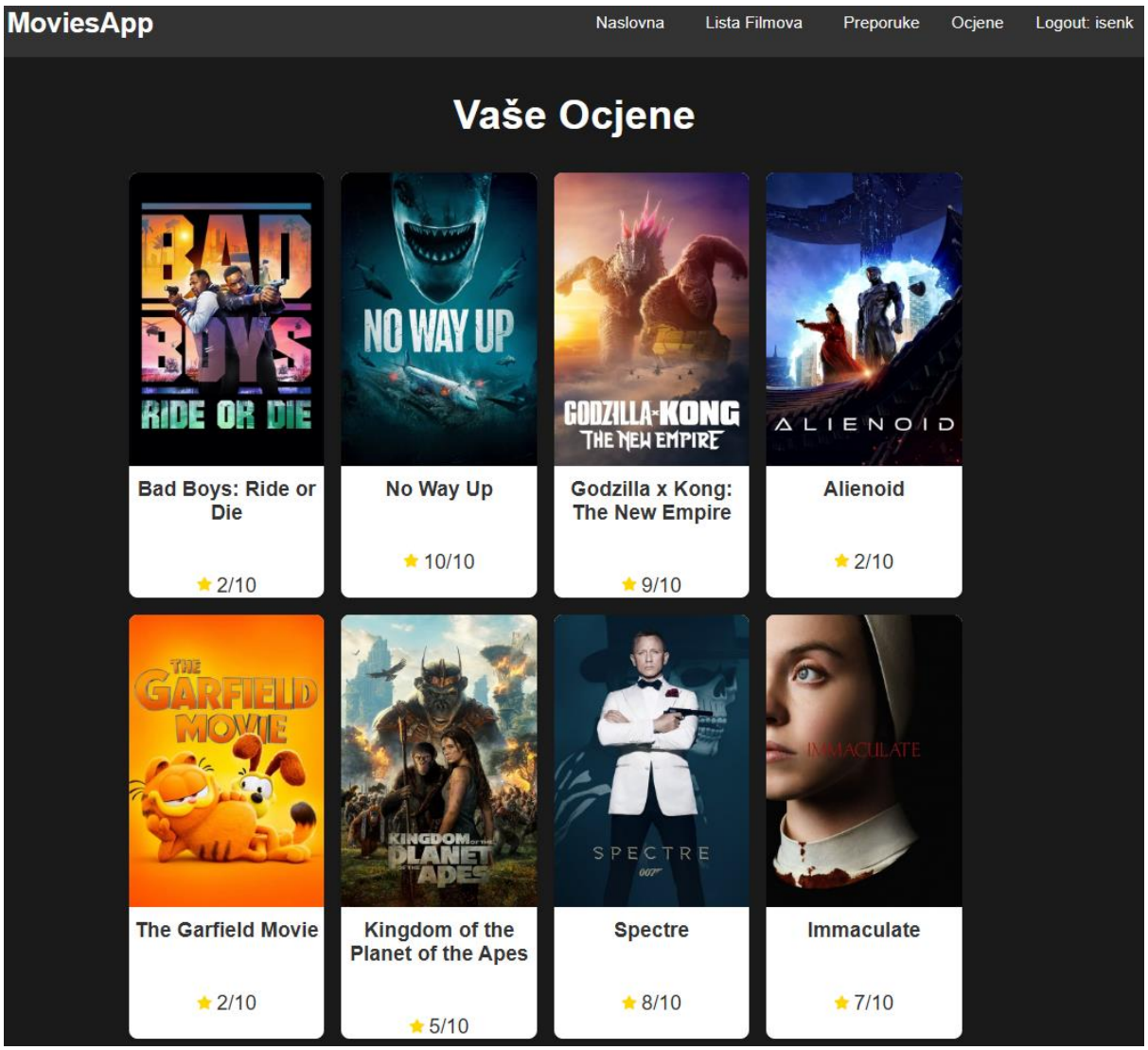
Veliki naslov sa narančastom pozadinom „Preporučeni Filmovi“ prikazuje korisniku sve filmove koji su slični ovome koji je odabrao. Ova preporuka koristi Contet-Based filtering

algoritam. Popis filmova preporučenih na temelju odabranog filma, uključujući plakate filmova, naslove gdje je moguće dodati ostale informacije o svakom filmu naknadno. Ovdje već vizualno je moguće vidjet da je preporučen film „Bad Boys“ koji je bio prvi u serijalu te samim time se može zaključiti da je preporuka dobra. Prikazano na slici 16.



Slika 16. Prikaz filma i preporuke

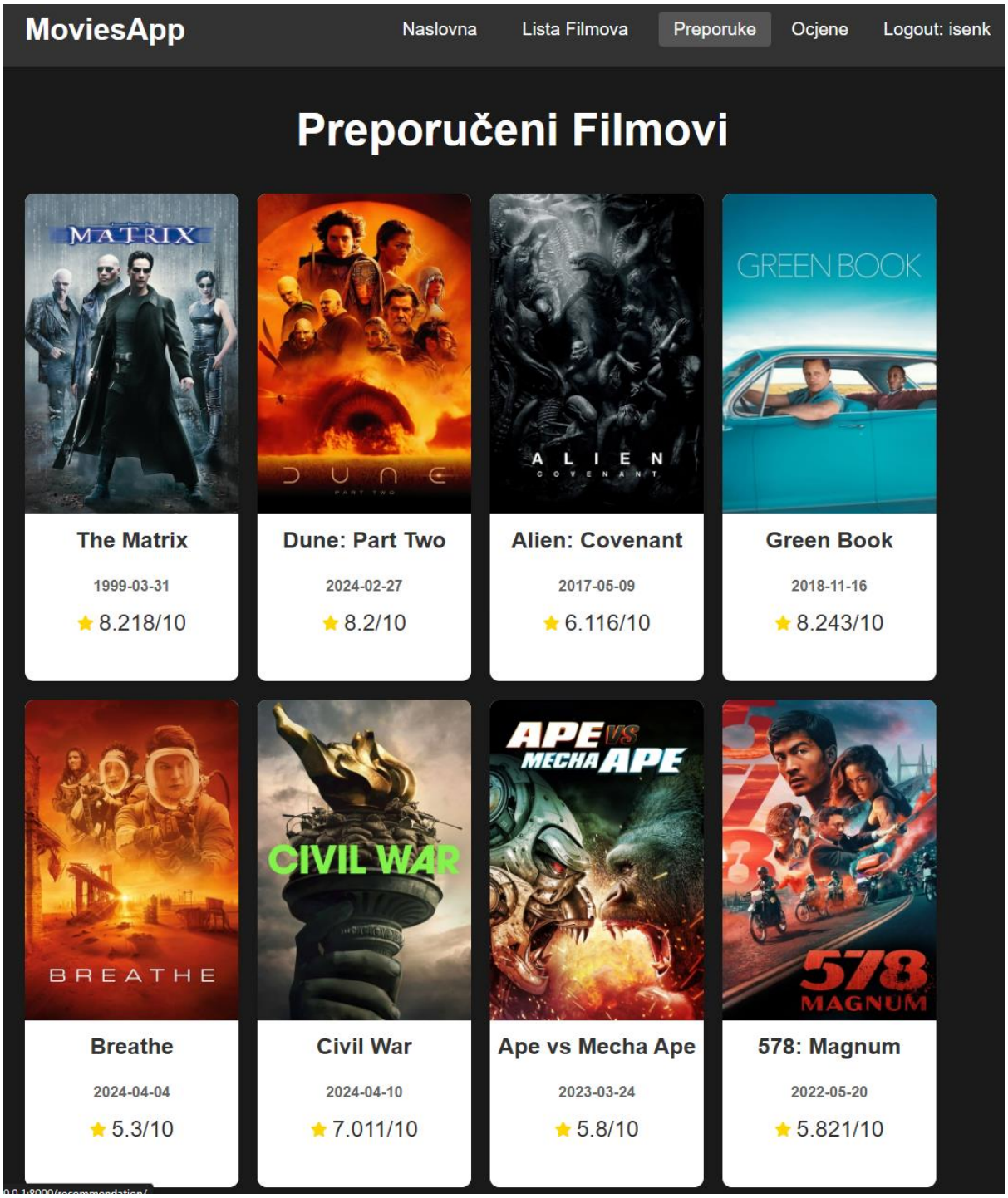
Ova stranica unutar aplikacije “MoviesApp“ prikazuje sve filmove koje je korisnik ocijenio. Izgled stranice je vrlo sličan onome na „Listi filmova“ s time da se ovdje prikazuju ocjene koje je korisnik sam dao za svaki film. Ove se ocjene koriste kasnije kod preporuke filmova. Ova stranica omogućava korisnicima pregled njihovih aktivnosti unutar aplikacije i jednostavan povratak na prethodno ocijenjene filmove. Ovdje je također moguće vidjeti u navigacijskoj traci kako su se dodala dva nova linka: „Preporuke“ i „Ocjene“ nakon što se korisnik uspješno prijavio. Prijašnji link „Login“ sada je „Logout:“ i korisničko ime. Prikazano na slici 17.



Slika 17. Ocijenjeni filmovi korisnika

Stranica "Preporučeni Filmovi" unutar "MoviesApp" pruža korisnicima personalizirane preporuke filmova na temelju user-based collaborative filtering metode. Na slici se može također vidjeti izgled kako izgleda link prije nego što se klikne na njega kako bi korisniku bio odabir očit. Izgled je identičan kao i prošle liste filmova kako bi korisnik imao prepoznatljivu stranicu umjesto da izgleda kao da svaki put otvara različitu aplikaciju. Ovaj prikaz funkcionira na način da se nakon pokretanja user-based filtering algoritma, dobije rezultat u obliku id

filmova nakon čega se pojedini id prosljeđuje API-ju kako bi se dobile sve potrebne informacije. Prikazano na slici 18.



Slika 18. Preporuka filmova korisniku

4. Zaključak

U ovom diplomskom radu obrađena je izrada web aplikacije za preporuku filmova koristeći Apache Spark, Django, i različite tehnike strojnog učenja. Cilj je bio stvoriti platformu koja korisnicima pruža personalizirane preporuke filmova temeljenih na njihovim prethodnim ocjenama, koristeći metode "Content-Based Filtering" i "Collaborative Filtering". Aplikacija je osmišljena tako da koristi prednosti više naprednih tehnologija, omogućavajući precizne i relevantne preporuke filmova, te time poboljšava korisničko iskustvo.

Izrada aplikacije započela je definiranjem ključnih tehnologija i okvira potrebnih za razvoj. Django, Pythonov web framework, odabran je za izgradnju aplikacije zbog svoje brzine, fleksibilnosti i opsežne zajednice podrške. PostgreSQL je korišten za upravljanje podacima, dok je Apache Spark služio za obradu velikih količina podataka i pripremu za strojno učenje. The Movie Database (TMDB) API korišten je za prikupljanje informacija o filmovima, što je omogućilo bogatu bazu podataka za preporuke.

Kroz rad je detaljno opisan proces implementacije i integracije različitih komponenti sustava. Od inicijalizacije Django projekta, do implementacije različitih modela i pogleda, te konfiguracije PostgreSQL baze podataka. Opisana je i upotreba Apache Sparka za pripremu podataka, uključujući čišćenje, transformaciju i oblikovanje podataka za strojno učenje. Jedna od ključnih komponenti bila je implementacija algoritama za preporuku. Koristeći "Content-Based Filtering" algoritme, aplikacija analizira karakteristike filmova i korisničke preferencije kako bi preporučila slične filmove. "Collaborative Filtering" koristi povijest ocjena sličnih korisnika za generiranje preporuka, čime se postiže viša razina personalizacije. Kombinacijom ovih metoda postignuta je visoka točnost preporuka, što je ključno za korisničko zadovoljstvo.

Kroz razvoj aplikacije identificirani su brojni izazovi, uključujući upravljanje velikim količinama podataka, optimizaciju performansi i osiguravanje sigurnosti. Rješenja su implementirana korištenjem najbolji praksi u razvoju web aplikacija, kao i korištenjem modernih tehnologija i alata za obradu podataka.

U konačnici, aplikacija je razvijena kao lokalna, s mogućnostima daljnje ekspanzije i integracije u produkcijsko okruženje. Korištenje lokalne baze podataka i API-ja omogućilo je brz razvoj i testiranje funkcionalnosti bez potrebe za vanjskim servisima.

Kao zaključak, ovaj rad pokazuje kako se moderne tehnologije mogu koristiti za stvaranje naprednih sustava za preporuku koji mogu značajno poboljšati korisničko iskustvo. Aplikacija pruža solidnu osnovu za daljnji razvoj i proširenje, uključujući mogućnost dodavanja novih funkcionalnosti kao što su preporuke televizijskih serija ili integracija s drugim API-ima. Rad također demonstrira važnost kombiniranja različitih tehnika strojnog učenja za postizanje što preciznijih i relevantnijih preporuka.

5. Reference

[1] S. Salloum, R. Dautov, X. Chen i P. X. Peng, »Big data analytics on Apache Spark,« Springer International Publishing Switzerland, 2016.

[2] D. Roy i M. Dutta, »A systematic review and research perspective on recommender systems,« 2022.

[3] Y. Li, K. Liu, R. Satapathy, S. Wang i E. Cambria, »Recent Developments in Recommender Systems: A Survey,« IEEE Publication Technology Group, 2021.

[4] A. Sun i Y. Peng, »A Survey on Modern Recommendation System based on Big Data,« Department of Electrical and Computer Engineering University of Miami Coral Gables, 2024.

[5] J. L. Qian Zhang i Y. Jin, »Artificial intelligence in recommender systems,« 2021.

[6] D. Rubio, »Beginning Django,« 2017.

[7] H. Gore, R. K. Singh, A. Singh, A. P. Singh, M. Shabaz, B. K. Singh i V. Jagota, »Django: Web Development Simple & Fast,« 2021.

[8] J. Martini, »Using Django and JavaScript to implement,« 2019.

[9] W. S. Vincent, »Django for APIs: Build web APIs with Python and Django«.

[10] K.-S. Nenova, »Modeling a Dialog System for Movie,« 2016.

[11] B. Dias, »Decision Support System to Predict,« 2019.

[12] B. Momjia, »PostgreSQL: Introduction and Concepts,« 2004.

[13] S. Juba, A. Vannahme i A. Volkov, Learning PostgreSQL, 2015.

[14] X. Gong, F. Erwee i V. Rautenbach, »GEOMETRY VIEWER FOR PGADMIN4: A PROCESS GUIDED BY THE GOOGLE,« 2019.

[15] O. A. Mohamad, »New Virtual Environment Based on Python Programming,« 2022.

[16] S. Reddy, S. Nalluri, S. Kunisetti, S. Ashok i B. Venkatesh, »Content-Based Movie Recommendation System Using Genre Correlation,« 2019.

[17] S. Sen, »Collaborative Filtering Recommender Systems,« Springer, Berlin, Heidelberg, 2007.

[18] G. Suganeshwari i S. S. Ibrahim, »A Survey on Collaborative Filtering Based Recommendation System,« Springer International Publishing Switzerland, 2016.

[19] A. Carrola, »Synthesizing Realistic Substitute Data for a Law Enforcement,« 2022.

6. Literatura

https://developer.themoviedb.org/docs/getting-started

https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv

https://image.tmdb.org/t/p/w500/kPcHuPYqzkSo4bmPHtH82GaeEgX.jpg

https://stackoverflow.com/

https://faker.readthedocs.io/en/master/index.html

https://www.scaler.com/topics/django/generating-dummy-data-in-django/

https://medium.com/@deepapandithu/recommender-system-user-collaborative-filtering-37613f0c6a9

https://realpython.com/build-recommendation-engine-collaborative-filtering/#user-based-vs-item-based-collaborative-filtering

https://www.freecodecamp.org/news/how-to-build-a-movie-recommendation-system-based-on-collaborative-filtering/

https://spark.apache.org/

https://www.djangoproject.com/start/overview/

https://djangostars.com/blog/10-popular-sites-made-on-django/

https://towardsdatascience.com/hands-on-content-based-recommender-system-using-python-1d643bf314e4

https://www.ibm.com/topics/collaborative-filtering

https://www.be-terna.com/insights/recommendation-systems-in-e-commerce-whats-the-thing-youve-never-known-but-always-wanted-to

7. Popis slika

Slika 1. PgAdmin prijava 9
Slika 2. PgAdmin sučelje 10
Slika 3. Kreiranje servera 11
Slika 4. Kreiranje tablice 11
Slika 5. Pregled podataka tablice 14
Slika 6. Upit na bazi podataka 14
Slika 7. Django administrator sučelje 19
Slika 8. Content-Based filtering primjer 47
Slika 9. Rezultat algoritma 51
Slika 10. Collaborative filtering primjer 52
Slika 11. Primjer preporuke filmova 53
Slika 12. Naslovna stranica aplikacije 62
Slika 13. Prikaz popularnih filmova 63
Slika 14. Prozor prijave korisnika 64
Slika 15. Prozor registracije korisnika 65
Slika 16. Prikaz filma i preporuke 67
Slika 17. Ocijenjeni filmovi korisnika 68
Slika 18. Preporuka filmova korisniku 69

8. Popis tablica

TABLICA 1. COLLABORATIVE FILTERING PREPORUKA 53
TABLICA 2. USPOREDBA COLLABORATIVE FILTERING METODA 54