FindWatch: web aplikacija za pronalazak filmova i serija

Topčagić, Saud

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:468063

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-25**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike

Saud Topčagić

FindWatch: web aplikacija za pronalazak filmova i serija

Završni rad

Pula, rujan 2021.godine

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike

Saud Topčagić

FindWatch: web aplikacija za pronalazak filmova i serija

Završni rad

JMBAG: 0303075962, redovni student

Studijski smjer: Sveučilišni preddiplomski studij informatika

Kolegij: Programiranje

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Saud Topčagić, ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Saud Topčagić

U Puli, rujan, 2021. godine

IZJAVA

o korištenju autorskog djela

Ja, dolje potpisani Saud Topčagić dajem odobrenje Sveučilištu Jurja Dobrile

u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „FindWatch: web aplikacija za pronalazak filmova i serija“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan 2021. godine

Student

Saud Topčagić

Sažetak

Cilj završnog rada je napraviti dokumentaciju izrade servisa ili API-ja. Zajedno sa tim servisom biti će izrađena i web stranica koja prikazuje zamišljenu upotrebu navedenog servisa. Aplikacija bi trebala korisnicima omogućiti jednostavno praćenje novih i popularnih filmova, pretragu bilo kojeg filma ili serije te ocjenjivanje istog. Korisnici bi imali svoju vlastitu biblioteku u koju spremaju svoje filmove i serije u grupe s obzirom na to da li im se sviđaju ili su ih pogledali i slično. Ujedno bi aplikacija trebala korisnicima predlagati filmove i serije koje bi im se mogle svidjeti na osnovi prethodno danih ocjena svakog korisnika. Glavni dio ovog rada je servis dok web stranica (frontend) ima zadaću prikazati podatke koje joj servis šalje.

Ključne riječi: filmovi, serije, kino, web aplikacija, JavaScript React.js, Node.js, Express.js, MongoDB

Abstract

The aim of this bachelor's thesis is to document a development of a web service or API. Along with the service, a website will be created in order to show the intended use of the said service. The application should allow users to easily follow new and popular movies and or shows, search for any movie or series, and rate it. Users would have their own library where they will be able to store their movies and series in groups depending on whether they like it or have watched it and so on. At the same time, the application should suggest to users movies and series that they might like based on previous user ratings. The main part of this thesis is the service (backend) while the website (frontend) has the task of displaying the data that the service sends.

Keywords: movie, shows, cinema, web application, JavaScript React.js, Node.js, Express.js, MongoDB

Sadržaj

1. Uvod

U današnjim modernim vremenima internet je postao jedna od osnovnih potreba života. Upleo se u gotovo sve aspekte naših života, a životi većine ljudi ne bi bili isti da nema interneta. Svakodnevni životi većine ljudi i njihove aktivnosti su primarno fokusirane na internet. Osnovne stvari kao gledanje recepata, filmova, serija, spajanje jedno s drugima, kupovina raznih stvari te ponajviše dolaska do potrebnih informacija se danas dohvaća preko interneta. Tu tvrdnji dodatno podupire činjenica da mnogi planiraju ili već imaju karijere koje se baziraju primarno na internetu. Pod time se misli da se ljudi bave poslovima koje uključuju davanje usluga preko interneta, oglašavanje, izradu web stranica, servisa i slično. Informacije na internetu se nalaze na web stranicama koje su izrađene sa strane firmi ili pojedinaca. Web stranice mogu bili vrlo jednostavne kao što su stranice sa samo jednom „stranicom“ (Single page Website) ili napredne web stranice koje se sastoje od mnogo stranica, baza, naprednih algoritama i još mnogo drugih aspekata. Dosta popularne su, kao što su prije navedene web stranice koje se sastoje od nekoliko stranica te su povezane sa bazom podataka u koju spremaju podatke potrebne za normalan rad stranice i pružanje određene usluge korisnicima. U nastavku ćemo prikazati jednu od takvih web stranica. Specifično, web stranica koja se bavi pružanjem usluga za pronalazak filmova i serija, spremanje istih, ocjenjivanje te pomoć pri pronalasku mogućeg željenog filma.

Ideja za izradu ovog servisa dolazi iz želje da se na jednom mjestu može pronaći željeni film, ocijeniti pogledani, ali i spremiti film koji bi se pogledao u budućnosti te za asistenciju pri pronalasku filmova koji bi nam se mogli svidjeti, a to sve zahvaljujući podacima koje smo dali servisu. Trenutno postoje web aplikacije nalik našoj, kao što su: JustWatch [15], IMDb [17], Rotten Tomatoes [16] te još mnogo drugih manje poznatih. Svaka od navedenih aplikacija ima većinu mogućnosti koje su prethodno navedene, ali njihova količina informacija može biti i njihova mana. Imaju mnogo kompleksnosti da korisnici nisu ni u stanju pronaći veliki broj mogućnosti koje one nude. Naš servis je fokusiran na jednostavnost i učinkovitost, kao što su pretraživanje filmova, vođenje svoje privatne male biblioteke svih odabranih, pogledanih, ocijenjenih filmova i serija. Zajedno sa prikazom preporuka koji film ili serija bi se korisniku mogli svidjeti.

U sljedećem poglavlju biti će navedene tehnologije koje su se koristile tijekom razvoja servisa te koraci koji su se morali napraviti prilikom razvoja stranice. Nakon toga će biti priloženo objašnjenje implementacije funkcionalnosti u kombinaciji sa slikama koda. Prvo će biti objašnjen servis zato što je on kompliciraniji, navesti će se korištena baza podataka, biblioteke te njihova funkcionalnost. Zatim slijedi frontend dio koji je jednostavniji te će isto opisivati korištene tehnologije i biblioteke. Na kraju ćemo navesti nekoliko stvari koje su bile vezane za projekt, ali ne i direktno za implementaciju funkcionalnosti kao što su moguće nadogradnje, problemi te nekoliko završnih riječi.

2. Web tehnologije i alati

Prilikom izrade web servisa korištene su sljedeće tehnologije. Za izradu web servisa i dizajna je korišten JavaScript, React.js i JSX te Express.js. Za bazu se koristila nerelacijska baza MongoDB, dok se za tekstualni editor koristio popularan Visual Studio Code [18] te Git [20] i GitHub [19] za verzioniranje koda u projektu. U sljedećim poglavljima opisat će se navedene korištene tehnologije. Pomoć servisu za dohvaćanje podataka o filmovima i serijama su korištena dva API[1]-ja. Neslužbeni imdbAPI [12] i JustWatchAPI [13].

2.1. JavaScript

JavaScript je vrlo popularan, dinamičan skriptni programski jezik. No, iako je skriptni jezik JS(JavaScript) podržava i objektno-orijentirani stil. Najčešće se koristi pri izradi web stranica iako ima i mnogo drugih mogućnosti. Klijenti naših web preglednika koriste JavaScript najčešće za prikaz dizajna web stranice, ali isto tako JavaScript govori web stranicama kako da se ponašaju. Jednostavan je jezik za početnike, a ujedno i jedan od najmoćnijih skriptnih jezika. Na slici 1. prikazan je primjer JavaScript koda iz ovog projekta.

```
const changes = diff(oldDoc, movieUserData)._doc
delete changes._id
delete changes.genres
if (changes == undefined) return
let db = await connect();

let result = await db.collection("movie_user_data").updateOne(
  { jw_id: jwId, user_id: ObjectId(userId) }, { $set: changes });

if (result.modifiedCount === 1) {
  await updateUserSimilarityScores(userId)

  return result
```

Slika 1. JavaScript kod

[1] Aplikacijsko Programsko Sučelje (Application Programming Interface)

2.2. React.js i JSX

U ovom projektu je korišten React.js okvir koji je primarno namijenjen za izradu dizajna web stranica. Pisan je u JavaScriptu koji se koristi i unutar Reacta za izgradnju dizajna. Specifično se koristi JSX. JSX nije prezentacijski jezik već je to zapravo JavaScript-ova sintaksna ekstenzija. Koristi se najčešće u React okviru, a služi za opisivanje ili izgradnju dizajna web stranice. Prednosti JSX-a nad HTML[2]-om je to što unutar JSX postoji mogućnost pisati JavaScript jezik. To nam daje dodatnu dinamičnost i olakšanu izgradnju web stranica. JSX se kompilira u normalan JavaScript koji se pokreće na web stranici te prikazuje njen izgled. Na slici 2. se vidi primjer JSX koda koji izgleda kao HTML. React.js održava poznata kompanija Facebook i velika zajednica individualnih programera. To je moguće zato što React izvorni kod može vidjeti tko god poželi te isti može pomoći u poboljšavanju Reacta. Dosta često se koristi sa pripadnim bibliotekama koje daju opširnije funkcionalnosti i više mogućnosti programerima.

```
function App() {
  return (
    <div className="App" type="movie">
      <Home />
    </div>
  );
}
```

Slika 2. JSX kod

[2] HyperText Markup Language

2.3. Express.js

Express.js je najpopularniji web okvir za JavaScript koji se koristi u mnogim drugim popularnim JavaScript web okvirima. Express.js nudi mehanizme pisanja obrađivača za HTTP zahtjeve na različitim URL-ovima koje programer može sam odrediti. Namještanje uobičajenih postavki kao što je port koji se koristi za spajanje na servis. Koristeći Express programeri mogu u cijelosti dizajnirati svoj server ili web servis. Express je napravljen kao minimalistički web okvir, ali zato postoji mnogo paketa koji mogu riješiti skoro svaki problem sa kojim se osoba koja izrađuje servis može susresti. Koriste se paketi za rad sa kolačićima, sesijama, prijavama korisnika i slično. Express kod se sastoji primarno od ruta koje se definiraju, na koje dolaze pozivi kao što su POST ili GET, te se nakon dolaska, ti zahtjevi obrađuju i eventualno šalju nazad web stranicama ili drugim servisima.

2.4. MongoDB

MongoDB je nerelacijska baza podatka koja u sebe pohranjuje fleksibilne dokumente za razliku od normalnih baza podatka. Dizajnirana je sa ciljem da bude skalabilna i fleksibilna te kao i većina baza podataka podržava pretraživanje i indeksiranje. Sastoji se baza, kolekcija i dokumenata. U bazama se nalaze kolekcije, što bi bilo slično tablicama u relacijskim bazama podataka. Unutar kolekcija se nalaze dokumenti, što su podaci koji se spremaju unutar baze te koji se dohvaćaju po potrebi. Dokumenti se mogu spremati u popularnom JSON obliku (JavaScript Object Notation) što olakšava integraciju sa servisima koji se pišu u JavaScriptu dok se svejedno može koristiti sa bilo kojim drugim servisima zbog popularnosti JSON-a. slika 3. prikazuje oblik dokumenata unutar MongoDB kolekcije. MongoDB je jednostavan za korištenje, te se kolekcijama može pristupiti iz web preglednika. Iako je jednostavan, MongoDB ima mogućnost zadovoljiti kompleksne zahtjeve velikih servisa.

Slika 3. Dokument unutar MongoDB kolekcije

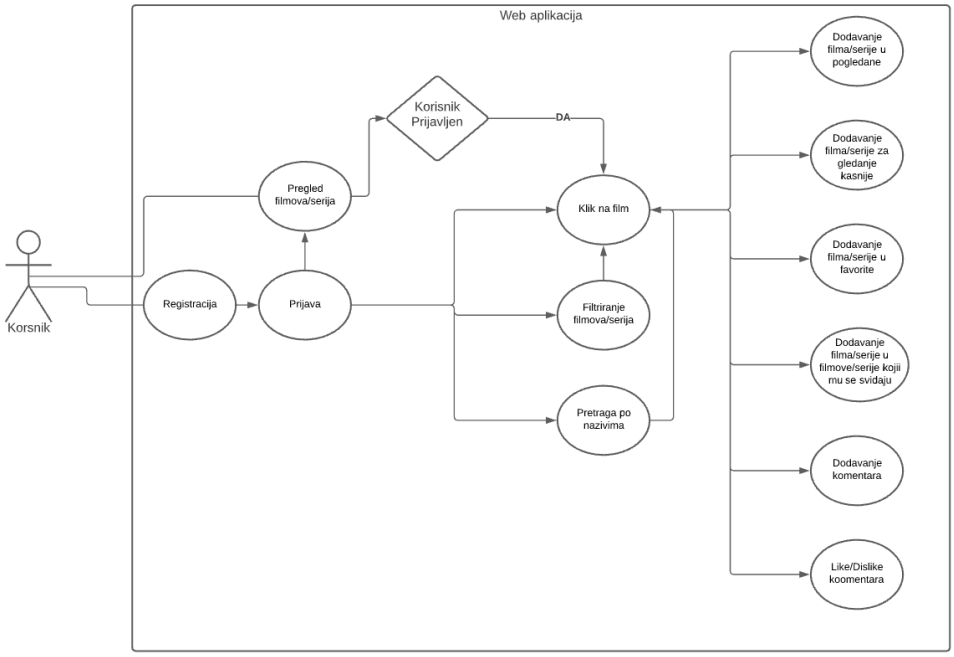
3. Razrada funkcionalnosti

U aplikaciji mogu postojati korisnici koji su posjetili web stranicu kao gosti, drugim riječima nemaju namjeru odraditi registraciju i napraviti račun na stranici. Takvi korisnici imaju limitirane opcije kada dođe do pristupa sadržaju na stranici. Imaju mogućnost samo pregledavati početnu stranicu na kojoj se nalaze poznati filmovi/serije, filmovi koji su u kinima, najpopularniji filmovi/serije... Druga vrsta korisnika bi bili korisnici koji se odluče registrirati i napraviti račun na aplikaciji. Takvi korisnici imaju istu mogućnost pregleda početne stranice zajedno sa nekoliko dodatnih mogućnosti. Neke od tih mogućnosti su pretraživanje filmova/serija po nazivu, mogućnost filtriranja filmova/serija po nekim atributima kao što su žanrovi, imdb ocjene i slično. Jedne od najbitnijih funkcionalnosti koje registrirani korisnici imaju su mogućnost klika na odabrani film/seriju što korisniku prikaže dodatne detalje odabrane stavke, zajedno sa video trailer-om. U slučaju da se radi o seriji, korisnik može vidjeti popis sezona i epizoda odabrane serije. Svaku epizodu može označiti kao pogledanu te za svaku seriju za koju korisnik ima bilo kakve podatke, aplikacija šalje obavijest o novim epizodama. Isto tako postoji opcija da korisnik za svaki film/seriju doda oznaku da mu se sviđa, ne sviđa, da mu je među favoritima, da će pogledati stavku kasnije, da ju je pogledao te ponajbitnije dodati ocjenu za film/seriju. Ocjene pomažu algoritmu koji se nalazi na servisu da

korisniku pobliže predlaže filmove ili serije koje bi mu se mogle svidjeti bazirano na ocjenama drugih sličnih korisnika. Drugim riječima, što više korisnika dodijeli ocjene, to je algoritam precizniji. Na posljetku, registrirani korisnik može pregledati sve filmove i serije koje je označio sa bilo kojom od prijašnje navedene stavke klikom na ikonu na izborniku te ujedno i pregledati svoj profil koji isto tako nudi iste opcionalnosti pregleda označenih filmova ili serija. Algoritam koji predlaže filmove i serije radi samo za registrirane korisnike iz razloga što samo od njih aplikacija smije i može prikupljati podatke, za korisnike koji nisu dodjeljivali ocjene ili korisnike koji nisu izradili korisnički račun, aplikacija vraća samo trenutno popularne filmove.

3.1. Dijagram slučaja uporabe

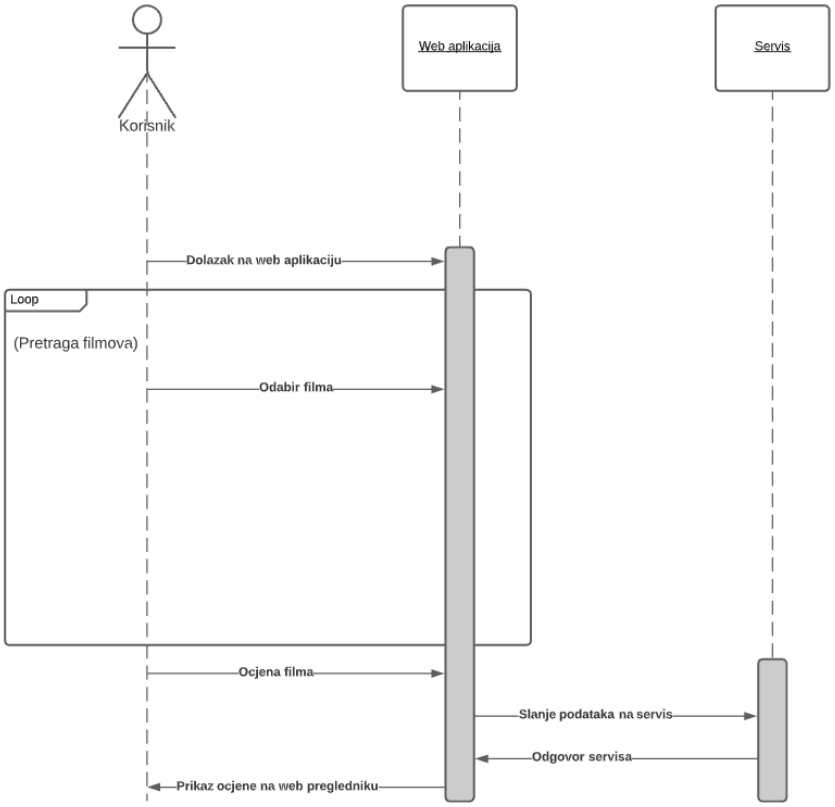
Dijagram slučaja uporabe se koristi kako bi u kratko prikazao aktore(korisnike) i njihovu interakciju sa sustavom. slika 4. prikazuje dijagram slučaja uporabe naše web aplikacije. Korisnik ima mogućnost posjetiti stranicu bez da se registrira što mu daje jednu opciju pregleda filmova na početnoj stranici. Osim normalnog posjeta, korisnik se može registrirati i prijaviti. Nakon prijave korisnik se vraća na početnu stranicu na kojoj ima pregled filmova isto kao i neprijavljeni korisnik. Ostale opcije koje prijavljeni korisnik ima su filtriranje filmova/serija po zadanim kriterijima, pretraga po nazivima filmova te mogućnost odabira filma ili serije koja mu nudi dodatne informacije te opcije za dodavanje filma/serije u pogledane, dodavanje filma/serije za gledanje kasnije, dodavanje filma/serije u favorite, dodavanje filma/serije u filmove/serije koji mu se sviđaju, dodavanje komentara, like ili dislike postojećih komentara.



Slika 4. Dijagram slučaja uporabe

3.2. Dijagram slijeda

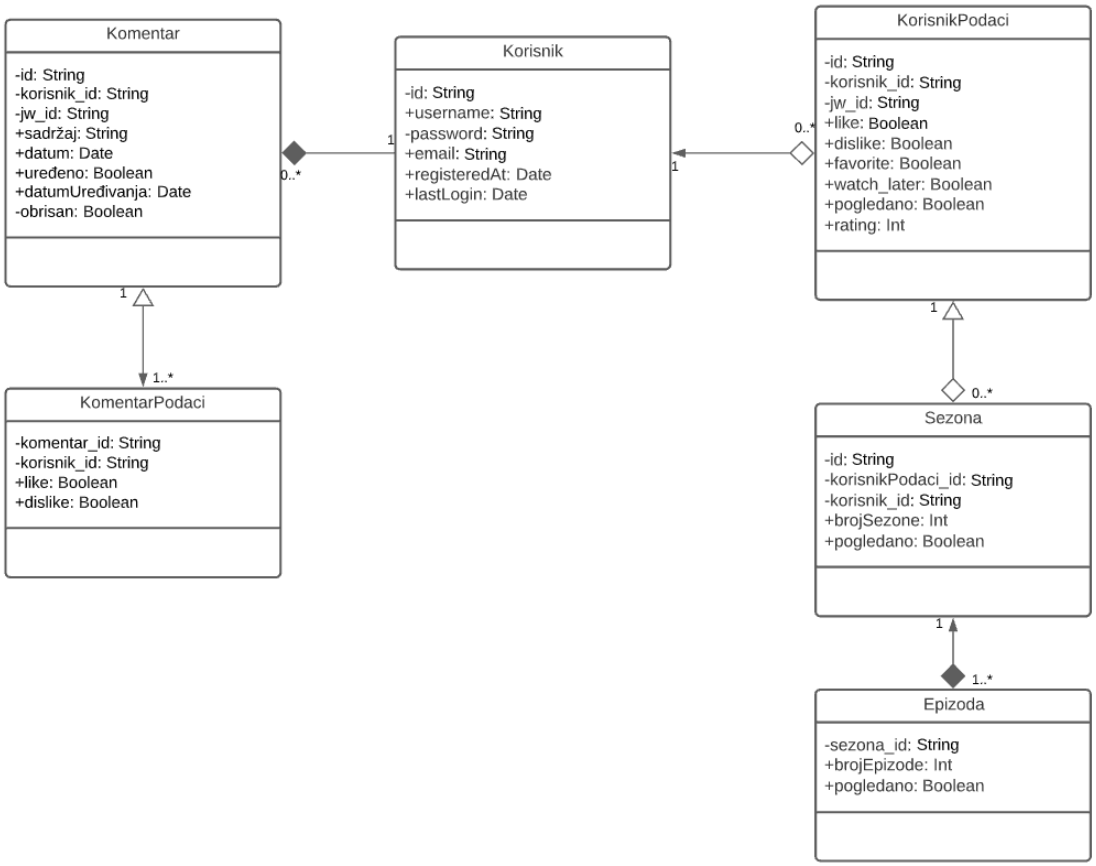
Dijagram slijeda prikazuje tok poruka između objekata tijekom interakcije. Primjer našeg dijagrama slijeda možemo vidjeti na slici 5. Dijagram prikazuje interakciju korisnika sa web aplikacijom u kojoj korisnik prolazi kroz proces dodjele ocjene jednom filmu. Nakon što prijavljeni korisnik dođe na web stranicu, počinje proces odabira filma koji traje sve dok korisnik ne odabere film. Nakon odabira, korisnik dodjeljuje ocjenu te se ta informacija šalje na servis. Servis obrađuje informaciju i šalje ju dalje te pri završetku obrade vraća odgovor web aplikaciji koja prikazuje ocjenu na stranici.



Slika 5. Dijagram slijeda

3.3. Klasni dijagram

Klasni dijagram se koristi u dizajniranju i modeliranju softvera kako bi prikazao strukturu sistema prikazivanjem klasa u sistemu, njihovih atributa te veza između klasa. Slika 6. prikazuje klasni dijagram web servisa. Promatranjem dijagrama može se zaključiti da je korisnik glavni entitet unutar servisa. Korisnik na sebe ima povezane komentare vezom jedan naprema nula ili više što nam govori da jedan korisnik ne mora imati niti jedan komentar ali može imati i više komentara. S desne strane se primjećuje tablicu KorisnikPodaci koja predstavlja entitet filma ili serije. Razlika između dvoje je ta što film nema dvije dodatne tablice Sezona i Epizoda dok serije imaju i jednu i drugu. Veza između Korisnika i KorisnikPodaci je jedan naprema nula ili više. Drugim riječima, tablica Korisnik ima istu vezu sa KorisnikPodaci i Komentar.



Slika 6. Klasni dijagram

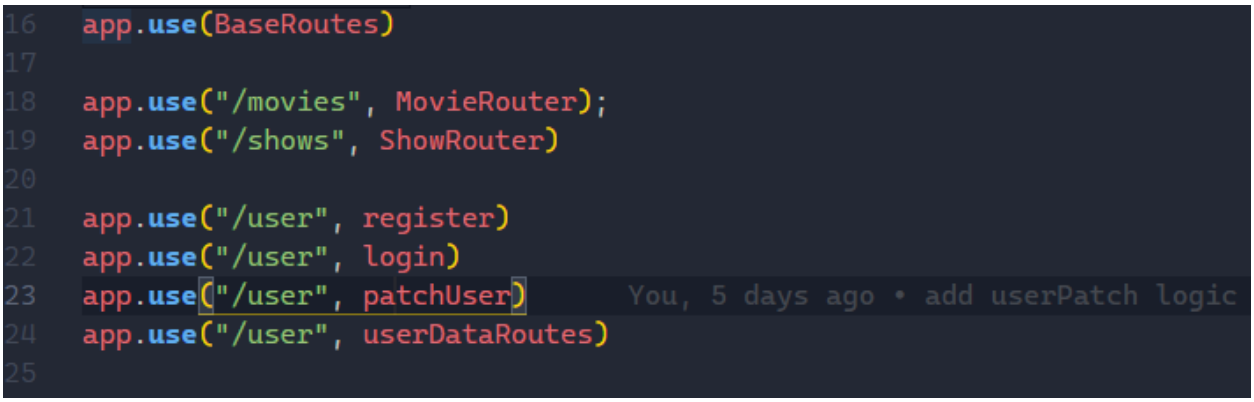
4. Implementacija servisa

U nadolazećem poglavlju biti će detaljnije objašnjen kod Servisa, biti će prikazani ključni dijelovi servisa te će ujedno biti razjašnjeni i objašnjeni u cilju da budu lakše razumljivi. Servis se sastoji od baze podataka, koja je u ovom slučaju MongoDB te web servisa građenog pomoću Express.js okvira koja služi kao srednja veza koja omogućuje komunikaciju web servisa i baze podataka. Uz to se podrazumijeva da servis mora imati logiku koja pomaže obraditi podatke i spremiti ih u bazu sigurno, te isto tako te podatke izvući iz baze i poslati na web preglednik. Način na koji će se prolaziti kroz kod je taj da će biti objašnjen proces koji se odvija kada se pozove određena ruta unutar servisa, pošto je tok koda na servisu uvijek unutar jedne rute koja ponekad poziva dodatne funkcije. Nakon rada tih funkcija, kod se vraća u tu istu rutu te se odgovor šalje nazad. Servis je sastavljen od takvih ruta pa je to najbolji pristup kada se treba objasniti cijeli

servis. Kod koji se ponavlja biti će izostavljen da se izbjegne repetitivnost. Prije nego što se počne prolaziti kroz kod, trebalo bi se razjasniti da je servisni dio glavni dio ovog završnog rada te je dizajn rađen u cilju da se pokaže i opiše funkcionalnost servisa, drugim riječima, drugi programeri mogu koristiti servis bez da koriste web stranicu. Još jedna od solucija bi mogla biti da programeri koji izgrađuju svoj dizajn i korisničko sučelje web stranice mogu koristiti navedeni servis u svrhu dohvaćanja informacija. Način na koji bi trebali koristiti servis mogu vidjeti iz priloženog korisničkog sučelja i koda.

4.1. Rute

Prvo je prikazana ulazna index.js datoteka, ova datoteka je napravljena na način da se ona prva poziva kada korisnik pošalje neki zahtjev na servis, nebitno koju rutu korisnik pozove. Zadatak ove datoteke je da pogleda rutu u pozivu te onda pozove određenu funkciju ovisno o toj ruti. Moglo bi se reći da je zadaća ove datoteke grananje poziva u određene funkcije. Slika 7. prikazuje sve grane u koje se poziv može proslijediti. U slučaju da poziv u svojoj adresi nema prefix „/movies“, „/show“ ili „/user“, poziv se prosljeđuje u funkciju BaseRoutes koja se vidi na slici 7, linija 18. U drugom slučaju ako poziv ima u sebi „/user“, pozivaju se neke od funkcija na linijama 21-24. Koja se točno funkcija poziva ovisi o tome koji su drugi parametri unutar adrese. Niti jedna ruta ne može potpuno istu adresu, osim ako se ne radi o različitim pozivima. Pozivi su najčešće GET, POST i PATCH. GET pozivi se razlikuju u tome što korisnici ne mogu slati podatke u tijelu poruke, jedina opcija slanja podataka je unutar adrese, što može biti nesigurno. Najčešće se u adresi šalju podaci koji nemaju toliku važnosti kada dođe do sigurnosti, dok se u tijelu poruke kod POST poziva šalju bitne informacije. Isto tako se šalje POST poziv i kada se više podatka šalje na servis pa nema smisla slati sve unutar adrese.



Slika 7. Ulazna datoteka index.js

4.2. Bazne rute

Kada korisnik pošalje zahtjev na servis, unutar zahtjeva šalje adresu rute. U slučaju da se u adresi nakon hosta ne nalazi jedna od ruta shows, movies ili user, kod ulazi u funkciju BaseRoutes. U ovom potpoglavlju će biti prikazano mnogo slika iz razloga što će odmah biti prikazane cijele funkcije koje se pozivaju te radi lakšeg razumijevanja nisu razdvojene u nekoliko manjih slika, tijekom pojašnjavanja koda, iste će slike biti nekoliko puta referencirane. Isto tako neke rute neće biti prikazane iz razloga što se slična ili ista funkcionalnost već objasnila u prijašnjoj ruti i da se rad nepotrebno ne oduži i bude previše repetitivan.

4.2.1. Ruta za pretraživanje

Bazna ruta „/search“ je jedna od GET ruta što znači da korisnik ne šalje nikakve podatke u tijelu poziva već se svi podaci koji se šalju nalaze u adresi. Ova ruta ima zadatak dohvaćanja podataka ovisno o izrazu koji joj se pošalje. Proces kroz koji se odvija kod je sljedeći. Nakon što se uspostavi da je poziv usmjeren na navedenu rutu, poziva se jedna posrednička funkcija koja se naziva „tokenVerify“ slika 8., linija 22. prikazuje gdje se funkcija poziva.

```
router.get('/search', [tokenVerify], async (req, res) => {
  let word = req.query.expression;

  const requestBody = filter({ query: word })

  const { items } = await Utils.justWatchAPIfetchData(`/en_US/popular`, requestBody)
      You, a month ago • Patch logic for all tables in the database
  res.json(items);
})
```

Slika 8. Ruta za pretraživanje

Slika 9. prikazuje tokenVerify funkciju. TokenVerify uzima podatke iz glave zahtjeva pod imenom „authorization“. U njemu očekuje string u obliku „Bearer“ plus enkriptirani string, odvojeno zarezom. Ako je string u odgovarajućem formatu, funkcija poziva jednu biblioteku koja se zove JWT. Ta biblioteka olakšava provjeru valjanosti tokena. Ako je token valjan, zahtjev je odobren i pušta se dalje u rutu. Valjanost tokena označava da je korisnik prijavljen na svoj račun i da ima pristup.

```
export default (req, res, next) => {      You, 4 months ago • auth and auth with JWT setu
  try {
    const auth = req.headers.authorization.split(" ");
    if (!auth) {
      res.status(401).send("Access denied.")
    }
    const type = auth[0];
    const token = auth[1];

    if (type !== "Bearer") {
      return res.status(401).send("Access denied.")
    } else {
      req.jwt = jwt.verify(token, process.env.TOKEN_SECRET);
      return next();
    }
  } catch (error) {
    res.status(401).send("Access denied.")
  }

}
```

Slika 9. tokenVerify funkcija

Nakon odobrenja, ruta uzima izraz i gradi poziv prema JustWatchAPI-ju, slika 10. Poveznica za navedeni API se nalazi na kraju rada, u prilogu. Nakon što je poziv spreman, poziva se funkcija „justWatchAPIfetchData“ koja se nalazi u „Utils“ dokumentu, slika 11, linija 23. Nakon poziva, API vraća odgovor koji se modificira na način da se nadoda link za poster filma te se isti taj modificirani odgovor šalje nazad korisniku.

```
export default options => {      You, a month ago • fix filters and add fetting jwId from imdbId
  const body = {
    "age_certifications": null,
    "content_types": null,
    "presentation_types": null,
    "providers": null,
    "genres": null,
    "languages": null,
    "release_year_from": null,
    "release_year_until": null,
    "monetization_types": null,
    "min_price": null,
    "max_price": null,
    "scoring_filter_types": null,
    "cinema_release": null,
    "query": null,
    "page": null,
    "page_size": null
  };
  const paramKeys = Object.keys(body);

  for (const key in options) {
    if (paramKeys.indexOf(key) === -1) {
      throw "invalid option '" + key + "'";
    }
    else {
      body[key] = options[key];
    }
  }

  return body;
}
```

Slika 10. Slika izgradnje poziva prema JustWatchAPI-ju.

Utils datoteka ima funkciju komunikacije sa aplikacijskim programskim sučeljima (API). Ova datoteka se poziva iz različitih dijelova servisa, ovisno gdje zatrebaju podaci koje API-ji nude. Osim normalnog dohvaćanja podataka, datoteka ima zadaću nadopunjavati i mijenjati dobivene podatke prije nego što ih proslijedi u servis. Primjer modifikacije podataka je da se nadopunjuje link za sliku svaki put kada se dohvate podaci o filmu ili seriji. Slika 11, linija 44. prikazuje nadopunjavanje podataka o određenom filmu ili seriji na način da se pregledaju kodovi dobivenih žanrova te se nadodaju puna imena žanrova.

```
const Utils = {

  async imdbAPIfetchData(params) {
    let request = await axios.get(`${imdbBaseUrl}${params}`);
    return request.data
  },

  async justWatchAPIfetchData(params, body) {

    let request = await axios.post(`${jwBaseUrl}${params}`, JSON.stringify(body));
    request.data.items.forEach(el => el.poster = generatePosterUrl(el.poster));
    return request.data
  },

  async fetchID(name) {
    let SeriesTitle = await axios.get(`${imdbBaseUrl}/Search/${APikey}/${name}`);
    return SeriesTitle.data.results[0].id;
  },

  async fetchInfo(IMDbId, queryFilters) {
    let request = await axios.get(`${imdbBaseUrl}/Title/${APikey}/${IMDbId}/${queryFilters}`);
    return request.data;
  },

  async fetchJWInfo(type, jw_id) {
    let request = await axios.get(`${jwBaseUrl}/${type}/${jw_id}/locale/en_US`);
    request.data.poster = generatePosterUrl(request.data.poster)
    let genreList = []
    request.data.genre_ids.map(x => genreList.push({
      short_name: genres[x].short_name,
      full_name: genres[x].translation
    }))
    request.data.genres = genreList
    return request.data;
  },

  async fetchJWSeasonInfo(show_id) {
    let request = await axios.get(`${jwBaseUrl}/show_season/${show_id}/locale/en_US`);
    request.data.poster = generatePosterUrl(request.data.poster)
    return request.data;
  }
}
```

Slika 11. Utils datoteka

4.2.2. Jw_info ruta

Bazna ruta „/jw_info“ isto je GET ruta koja se primarno poziva od korisnika kada on klikne na jedan od filmova. Funkcija vraća dodatne informacije o filmu te korisnik ima mogućnosti ocijeniti film, dodati ga u favorite i još nekoliko ocjena koje su već navedene u radu. Pošto je ovo ruta koja zahtjeva od korisnika da bude prijavljen sa valjanim računom, prije nego što kod uđe u rutu, ulazi u posredničku funkciju tokenVerify, slika 9. Funkcionalnost tokenVerify funkcije je opisana u potpoglavlju (**Ruta za pretraživanje**). Nakon ulaska u rutu, izvlače se podaci o tipu filma ili serije, jw_id i/ili imdb_id te korisnikov id. S obzirom na to da servis kao primarni id filma koristi jw_id, prvo se provjerava da li je poslan jw_id, ukoliko nije provjerava se da li postoji imdb_id te ako postoji, pomoću njega se izvlači jw_id, slika 12, linija 69. Nakon toga se poziva Utils datoteka, specifično fetchJWInfo funkcija koja poziva JustWatchAPI te se odgovor

dodatno modificira dodavanjem žanrova. Prikaz funkcije se nalazi na slici 11, linija 40. Nakon toga se pozivaju podaci o korisniku iz baze podataka, slika 13. Taj se poziv unutar sastoji od nekoliko poziva prema bazi podataka, referenca na slici 14. kao primjer jednog od tih poziva, u navedenom primjeru se poziva tablica sa korisnikovim podacima o filmovima/serijama. Ti pozivi se nadodaju u jedan odgovor koji se šalje nazad u rutu. Zatim se ti podaci dodaju na odgovor iz JustWatchAPI-ja te se takav šalje nazad korisnicima.

```
62  router.get('/jw_info', [tokenVerify], async (req, res) => {
63    try {
64      const type = req.query.type;
65      const jw_id = req.query.jw_id;
66      const imdb_id = req.query.imdb_id
67      const user_id = req.query.user_id
68
69      let id = jw_id
70      if (type == undefined) throw "Missing type."
71      if (id == undefined && imdb_id != undefined) id = await fetchJwId(type, imdb_id)
72      if (id == 0) throw "Unable to resolve id"
73
74      const data = await Utils.fetchJWInfo(type, id)
75
76      const userData = await GetUserData({ userId: user_id, jwId: id, type: type })
77
78      data.userData = userData
79
80      res.json(data);
81
82    } catch (error) {
83      res.status(400).send(error)
84    }
85  })
86
```

Slika 12. Jw_info ruta.

4.2.3. Ruta za dohvat podataka po atributima

Ova ruta nema specifično ime po kojem se poziva, umjesto toga se unutar adrese šalju samo podaci, specifično korisnikov id i ime atributa po kojemu se filmovi i serije žele dohvatiti, slika 13, linija 127. Prvo se kao uvijek izvlače podaci koje korisnik šalje iz adrese, zatim se šalje upit na bazu po atributu koji je dobijen. Nakon što se dobije popis svih tih filmova i serija, radi se iteracija nad tom listom i za svaki film/seriju se poziva fetchJWInfo funkcija, za referencu pogledati sliku 11, linija 40. Nakon što se dohvate podaci za svaki film i seriju, lista se vraća korisniku.

```
127  router.get("/:userid/:property", async (req, res) => {
128
129    const userId = req.params.userid;
130    const property = req.params.property;
131
132    try {
133      const response = await GetDataByProperty({ userId: userId, property: property });
134
135      const data = []
136      for (const x of response) {
137        data.push(await Utils.fetchJWInfo(x.type, x.id))
138      }
139
140      if (response == null) return res.status(422).send("Data cannot be empty.")
141      if (response.lenght == 0) return res.status(200);
142
143      res.json(data)
144
145    } catch (error) {
146      res.status(500).send(error)
147    }
148
149  })
150
```

Slika 13. Ruta za dohvat podataka po atributima.

4.2.4. Ruta preporuka (Recommendations ruta)

Ruta preporuka je jedna od najsloženija ruta u servisu. To je GET poziv u kojem se šalje samo korisnikov id. Nakon što se korisnikov id izvuče iz adrese, poziva se funkcija GetRecommendedMovies čiji je kod prikazan na slikama 14 i 15. Unutar te funkcije se prvo poziva " justWatchAPIfetchData" funkcija koja vraća sve popularne filmove i serije, referenca na slici 11, linija 23. Nakon toga se radi manipulacija podataka da se izdvoji imdb ocjena filma. Nakon toga se funkcija spaja na bazu podataka i iz tablice „user_similarity" izvlači sve podatke u kojima se nalazi korisnikov id. Ti se podaci sortiraju po polju „pcc" što označava Pearsonov koeficijent korelacije te se uzima top pet dokumenata iz razloga što nas zanimaju podaci od korisnika koji su najsličniji našem korisniku (Pearsonov koeficijent korelacije i implementacija). Nakon toga se dohvaćaju podaci o filmovima i serijama od trenutnog korisnika u cilju da se isti izdvoje tijekom predloška filmova. Nakon filtriranja filmova dobivenih od najsličnijih korisnika i trenutnog korisnika počinje se graditi lista filmova koji se žele predložiti te se dodatno ocjena predloška filma pomnoži sa dobivenom ocjenom filma i zbroji sa imdb ocjenom u slučaju da se film nalazi na popularnoj listi u cilju da se prvo predlože popularni filmovi. Kada se dobije lista id filmova poredanih po ocjeni, za svaki se poziva fetchJWInfo funkcija, slika 11., linija 40. Na kraju se odrađuje zadnja provjera da se vidi da li je dobiveno minimalno 10 filmova i serija. U slučaju da nije, u listu se stavlja lista popularnih filmova i serija koja je dobivena na početku funkcije. Takva lista se vraća nazad u rutu preporuka koja ujedno istu takvu listu vraća korisniku.

```
export default async userId => {
  try {
    let data = []
    const requestBody = filter()
    let apiMovies = await Utils.justWatchAPIfetchData(`/en_US/popular`, requestBody)
    let apiMovieScores = [];

    apiMovies.items.forEach(x => {
      let apiMovie = {};
      apiMovie.id = x.id
      apiMovie.imdb_rating = x.scoring.find(y => y.provider_type === "imdb:score")?.value

      if (apiMovie.imdb_rating !== undefined) {
        apiMovieScores.push(apiMovie)
      }
    })

    if (userId == null) return apiMovies

    const db = await connect()

    const cursor = await db.collection("user_similarity").find({ users: userId })

    const userData = await cursor.toArray()

    if (userData.length === 0) throw "User does not exist"

    const similarityData = _(userData)
      .orderBy('pcc', 'desc').filter(x => x.pcc > 0).take(5).value()

    if (similarityData.length == 0) return apiMovies;

    const users = _(similarityData).map(similarity => {
      return {
        ..._.omit(similarity, 'users'),
        userId: _.remove(similarity.users, user => user !== userId)[0],
      }
    }).value()

```

Slika 14. Get RecommendedMovies funkcija

```
const mainUser = await GetAllMovieUserData({ user_id: ObjectId(userId) })

const mainUserMovies = _.map(mainUser, 'jw_id')
const allMovieUserData = await GetAllMovieUserData()

await Promise.all(
  users.map(user => {
    user.reviews = _(_.filter(allMovieUserData, review => user.userId == review.user_id && !_.includes(mainUserMovies, review.jw_id)))
      .orderBy('rating', 'desc').take(6).value()
  }))

let movieRecommendations = []

users.forEach(user => user.reviews.forEach(review => {
  let apiRating = apiMovieScores.find(x => x.id == review.jw_id)?.imdb_rating
  movieRecommendations.push({
    movieId: review.jw_id,
    rating: apiRating != undefined ? review.rating * user.pcc + apiRating : review.rating * user.pcc,
    type: review.type
  })
}))

_(movieRecommendations)
  .orderBy('rating', 'desc')
  .unionBy('movieId')
  .value()

for (const x of movieRecommendations) {
  data.push(await Utils.fetchJWInfo(x.type, x.movieId))
}

if (data < 10) return apiMovies;

return data;
```

Slika 15. Get RecommendedMovies funkcija

4.2.5. Ruta obavijesti (Nofitications ruta)

Ruta obavijesti je dizajnirana da se poziva prilikom svakog učitavanja stranice. Cilj ove rute nije samo poslati korisnicima obavijesti nego i usporediti podatke od korisnika unutar baze i podatke sa JustWatchAPI-ja. To se događa iz razloga što je cilj da podaci koje korisnik ima za određene serije u bazi budu što točniji. Pod time se misli da je broj epizoda od određene serije uvijek točan da se ne bi moglo dogoditi da korisnik označi jednu epizodu kao pogledanu, a da ta epizoda ne bude u bazi. Ruta prvo izvlači korisnikov id koji je on poslao na servis, nakon toga poziva „PatchUserData“ funkciju koja se može vidjeti na slikama 16 i 17. Unutar te funkcije se dohvaćaju svi podaci vezani za jednog korisnika te se isto tako poziva justWatchAPI za dohvat serija koje se nalaze u korisnikovim podacima, za svaki se podatak provjerava da li je serija, razlog te provjere je što film nema smisla provjeravati zato što mu ne mogu stići nove epizode ni sezone. U slučaju da je podatak serija, uspoređuje se sa podatkom izvučenim iz API-ja. Prvo se provjerava da li je došla nova sezona te se id-ovi od novih sezona spremaju u listu. Zatim se iterira po sezonama dobivenim iz API-ja i za svaku sezonu se uspoređuje da li se nalazi u listi novih sezona, ako se dogodi da se pronađe, ta se sezona doda u bazu. Zatim se u toj istoj iteraciji pokreće još jedna iteracija koja provjerava nove epizode sličnom tehnikom. Svaki put kada se pronađe nova sezona ili epizoda, ista se dodaje u objekt koji se zove „notifications“. Ta se lista napuni nadolazećim sezonama i epizodama te se vraća kao odgovor web stranici. Obavijesti se ne spremaju u bazu iz razloga što bi dodale velik nivo kompleksnosti, a ne smatraju se bitnim dijelom servisa. Iz tog razloga se ostavlja na web servisu da sprema podatke o notifikacijama korisnika i to je ujedno obrazloženje razloga zašto bi se ova ruta trebala često pozivati od strane web stranice.

```
export default async userId => {
  try {

    const userData = await GetAllMovieUserData({ user_id: ObjectId(userId) })
    let notifications = {
      user_id: userId,
      notifications: []
    }

    if (userData == null) throw "Invalid user Id"

    for (const item of userData) {

      if (item.type == "show") {
        const show = await Utils.fetchJWInfo(item.type, item.jw_id);

        const db_show = await GetUserSeasonData({ userId: userId, movieUserDataId: item._id })

        const newSeasonIds = show.seasons.filter(x => !db_show.some(y => x.id.toString() === y.jw_id.toString())).map(x => x.id)

        for (const season of show.seasons) {
          const seasonData = Utils.fetchJWSeasonInfo(season.id)
          let seasonResponse = null

          if (newSeasonIds.includes(season.id)) {
            notifications.notifications.push({
              jw_id: seasonData.show_id,
              name: seasonData.show_title,
              sesason_number: seasonData.season_number,
              poster: seasonData.poster
            })

            seasonResponse = await PostUserSeasonData({
              userId: userId,
              movieUserDataId: item._id.toString(),
              seasonJwId: seasonData.show_id, seasonNumber: seasonData.season_number
            });
          }
          const seasonUserDataId = db_show.filter(x => x.season_jw_id == season.id)
          const db_eps = await GetUserEpisodeData(seasonUserDataId._id)
```

Slika 16. PatchUserData funkcija.

```
          const newEpisodes = seasonData.episodes.filter(x => !db_eps.some(y => x.episode_number === y.episode_number))

          for (const eps of newEpisodes) {
            notifications.notifications.push({
              jw_id: seasonData.show_id,
              name: seasonData.show_title,
              sesason_number: seasonData.season_number,
              poster: seasonData.poster,
              episode_number: eps
            })

            await PostUserEpisodeData({
              seasonDataId: seasonResponse != null ? seasonResponse.season_id : seasonUserDataId._id,
              episodeNumber: eps,
              seasonJwId: season.id
            });
          }
        }
      }
    }

    return notifications;
```

Slika 17. PatchUserData funkcija.

4.3. Ruta za filmove

Rute za filmove imaju jednostavnu zadaću, pozivanje imdbAPI-ja. ImdbAPI je servis koji nudi podatke o filmovima kao što su: najpopularniji filmovi, filmovi u kinima, nadolazeći filmovi... Ona je jedna od grana koja se nalazi u index.js datoteci, slika 7, linija 18. Ako korisnik želi dobiti podatke od jedne od tih ruta, mora u adresu nakon hosta dodati „/movies“ ili „/shows“. Unutar projekta web servisa, ove rute se pozivaju prilikom učitavanje početne stranice. Za ovaj projekt je izbjegavano preveliko korištenje ovog API-ja iz razloga što postoji limit na broj poziva u danu za besplatnu verziju. Kako je ovo projekt za fakultet i nije u planu da bude korišten za komercijalnu upotrebu, taj limit bi trebao biti dovoljan. Postoji nekoliko ruta za filmove i serije i isto tako za filmove i serije zajedno, oni se mogu vidjeti na prije spomenutoj poveznici na kraju rada. Ovdje će kao i do sada biti spomenuta samo jedna ruta radi izbjegavanja repetitivnosti. Na slici 19. se može vidjeti lista funkcija koje se mogu pozvati iz grane „/movies“. Svaka od priloženih funkcija poziva Imdb API te vraća odgovor nazad u rutu, koji se na posljetku vraća korisnicima.

```
export default {
  UpcommingMovies: async () => await Utils.imdbAPIfetchData(`/ComingSoon/${APikey}`),

  MoviesInTheaters: async () => await Utils.imdbAPIfetchData(`/InTheaters/${APikey}`),

  Top250Movies: async () => await Utils.imdbAPIfetchData(`/Top250Movies/${APikey}`),

  MostPopularMovies: async () => await Utils.imdbAPIfetchData(`/MostPopularMovies/${APikey}`),

  SearchMovies: async name => await Utils.imdbAPIfetchData(`/SearchMovie/${APikey}/${name}`),

  BoxOffice: async time => {
    if (time == 'allTime') {
      return await Utils.imdbAPIfetchData(`/BoxOfficeAllTime/${APikey}`);
    }
    else if (time == 'week') {
      return await Utils.imdbAPIfetchData(`/BoxOffice/${APikey}`);
    }
    else throw "Supported times are allTime and week.";
  }
}
```

Slika 18. Funkcija koja poziva imdbAPI.

4.4. Rute za korisnike

Rute za korisnike se odnose na korisnika i njegove podatke. Dijele se na registraciju korisnika, prijavu, mijenjanje korisnikove lozinke ili korisničkog imena te upravljanje korisničkim podacima o filmovima i serijama. Ovo poglavlje će objasniti samo neke od ruta koje se nalaze na grani „userDataRoutes“, za podsjetnik oblika grana pogledati sliku 7. Razlog tome je što bi ovaj rad bio predug, ali i na izostavljenim rutama nema drugačijih načina spremanja i dohvaćanja podataka koje se još nije ili neće vidjeti. Rute koje će se obraditi su zadužene za nadopunjavanje korisnikovih podataka u tablice, mijenjanje tih podataka te izračunavanje i spremanje Pearsonove koeficijent korelacije koja se koristi za prije spomenuto predlaganje filmova unutar „Recommendations rute“, slike 14 i 15. Sliku svih ruta unutar navedenog grananja može se vidjeti na slici 19.Od svih ruta na slici, u ovom radu će biti objašnjene rute koje se nalaze unutar „movieUserData“ funkcije prikazane na liniji 12.

```
12  router.use("/data", [tokenVerify], movieUserData)
13  router.use("/data", [tokenVerify], userSeasonData)
14  router.use("/data", [tokenVerify], userEpisodeData)
15  router.use("/data", [tokenVerify], userCommentsData)
16  router.use("/data", [tokenVerify], userCommentLikes)
```

Slika 19. Rute unutar userDataRoutes grane.

MovieUserData funkcija sadrži dvije rute, jedna za dohvaćanje podataka korisnika, druga za mijenjanje podataka korisnika. Ne postoji ruta za kreiranje podataka, razlog tome je što se unutar rute za mijenjanje, nadalje zvane PATCH rute prvo provjerava da li postoje korisnikovi podaci za odabrani film. U slučaju da podaci ne postoje, poziva se funkcija koja kreira nove podatke za korisnika. Prikaz ruta se može vidjeti na slici 20. Ruta GET se odmah poziva „GetMovieUserData“ koja dohvaća sve podatke iz baze. Nakon što ta funkcija vrati svoj odgovor, taj se odgovor prosljeđuje korisnicima. PATCH ruta prvo uzima sve podatke iz adrese koji su joj potrebni za odrađivanje, zajedno sa podacima iz tijela upita. Nakon toga se poziva funkcija „PatchMovieUserData“ te joj se prosljeđuju svi podaci. Nadalje će biti opisana navedena funkcija u detalje.

```
router.get("/userData", async (req, res) => {

  const data = req.query;

  try {
    const response = await GetMovieUserData(data);

    if (response == null) return res.status(422).send("Data cannot be empty.")
    if (response == {}) return res.status(400);

    res.json(response)

  } catch (error) {
    res.status(500).send(error)
  }

})

router.patch("/userData/:userid/:jwid/:type", async (req, res) => {

  const jwid = req.params.jwid
  const userId = req.params.userid
  const type = req.params.type
  const data = req.body;

  try {
    await PatchMovieUserData(jwid, userId, type, data)

    res.json({ "success": true })
  } catch (err) {
    res.status(500).send(err)
  }
})
```

Slika 20. Prikaz ruta unutar movieUserDataRoutes.

4.4.1. Funkcija za mijenjanje korisničkih podataka

Glavna namjera funkcije je da mijenja podatke korisnika o filmovima i serijama. Slika 21 prikazuje kod funkcije. Prvobitno se dohvaćaju postojeći podatci iz baze podataka te odmah nakon se provjerava da li postoje dohvaćeni podaci, drugim riječima da li postoje ti podaci u bazi. U slučaju da ne postoje, podaci se odmah kreiraju sa poslanim podacima iz rute. Nakon izrade se ponovno poziva ista funkcija da se dohvate novokreirani podaci. Nakon dohvata se podaci mijenjaju sa poslanim promjenama iz rute. Nakon što su promjene uspješno obavljane poziva se „updateUserSimilarityScore“ funkcija koja mijenja Pearsonovu koeficijent korelaciju na noviju verziju. To se poziva svaki put kada se pozove PATCH metoda od usera iz razloga što jedna od promjena može biti ocjenjivanje filma ili serije što je potrebno za davanje prijedloga filmova i serija korisnicima. Detaljno o Pearsonovoj koeficijent korelaciji i njegovoj implementaciji se govori na sljedećem potpoglavlju.

```
export default async (jwId, userId, type, data) => {
  try {
    let oldDoc = await GetMovieUserData({ jwId, userId, type })

    if (Object.keys(oldDoc).length === 0 && oldDoc.constructor === Object) {
      try {
        await CreateUserData({ jwId: jwId, userId: userId, type: type })
        oldDoc = await GetMovieUserData({ jwId, userId, type })

      } catch (error) {
        throw "Error: Failed To create user data"
      }
    }

    const movieUserData = createSchemas.MovieUserSchema(oldDoc, data)

    const changes = diff(oldDoc, movieUserData)._doc
    delete changes._id
    delete changes.genres
    if (changes == undefined) return
    let db = await connect();

    let result = await db.collection("movie_user_data").updateOne(
      { jw_id: jwId, user_id: ObjectId(userId) }, { $set: changes });

    if (result.modifiedCount === 1) {
      await updateUserSimilarityScores(userId)

      return result
```

Slika 21. PatchMovieUserData funkcija

4.4.2. Pearsonov koeficijent korelacije i implementacija

Pearsonova koeficijent korelacija se može opisati kao mjera linearne korelacije između dvije varijable. Rezultati koje daje su između -1 i 1. Rezultat -1 govori da postoji totalna negativna korelacija između dvije točke, u našem slučaju bi se to reklo da ako je korelacija između dva korisnika -1, ti korisnici imaju totalno različite ukuse u filmove i ne bi trebali razmatrati jednog kada imamo želju predložiti filmove drugom. U drugu ruku, ako je korelacije između dva korisnika 1, to nam govori da ta dva korisnika imaju jako slične ukuse, u većim bazama podataka, rezultati skoro nikad nisu strogo 1 ili -1. Formulu za Pearsonov koeficijent korelacije se može vidjeti na slici 22. [11]

$$\rho = \frac{pSum - \frac{user1MovieScoreSum * user2MovieScoreSum}{n}}{\sqrt{(user1MovieScoreSqSum - \frac{user1MovieScoreSum^2}{n}) \cdot (user2MovieScoreSqSum - \frac{user2MovieScoreSum^2}{n})}}$$

Slika 22. Pearsonov koeficijent korelacije [11]

Implementacija Pearsonove koeficijent korelacije je poprilično jednostavna kada postoji formula, prikaz implementacije u kodu se može vidjeti na slici 23. Tu funkciju poziva funkcija „updateUserSimilarityScore“ koja ujedno sprema i dobivene podatke u bazu.

```
const pcc = (user1, user2) => {
  const n = _.size(user1)

  if (n === 0) {
    return n
  }

  const user1MovieScoreSum = _.sumBy(user1, 'rating')
  const user2MovieScoreSum = _.sumBy(user2, 'rating')
  const user1MovieScoreSqSum = _.sumBy(user1, squaredNum)
  const user2MovieScoreSqSum = _.sumBy(user2, squaredNum)

  let pSum = 0

  for (let i = 0; i < n; i++) {
    pSum += (user1[i].rating * user2[i].rating)
  }

  const num = pSum - ((user1MovieScoreSum * user2MovieScoreSum) / n)

  const den = Math.sqrt(
    (user1MovieScoreSqSum - (Math.pow(user1MovieScoreSum, 2) / n)) *
    (user2MovieScoreSqSum - (Math.pow(user2MovieScoreSum, 2) / n))
  )

  if (den === 0) {
    return 0
  }

  return num / den
}
```

Slika 23. Implementacija Pearsonove koeficijent korelacije

5. Implementacija web stranice

Ovo poglavlja opisuje razradu funkcionalnosti web stranice. Stranica je građena pomoću React okvira i Material UI [21] biblioteke za prikazivanje ikona na stranici te joj je svrha prikaz rada servisa. Za komunikaciju sa servisom je korištena biblioteka Axios [22]. U nastavku će biti prikazane slike sučelja stranice zajedno sa opisom i kodom koji je potreban za normalan rad stranice. Neke funkcionalnosti servisa nisu implementirane na ovoj stranici, ali kao što je ranije navedeno, stranica je samo prikaz kako bi se servis mogao koristiti.

5.1. Rute

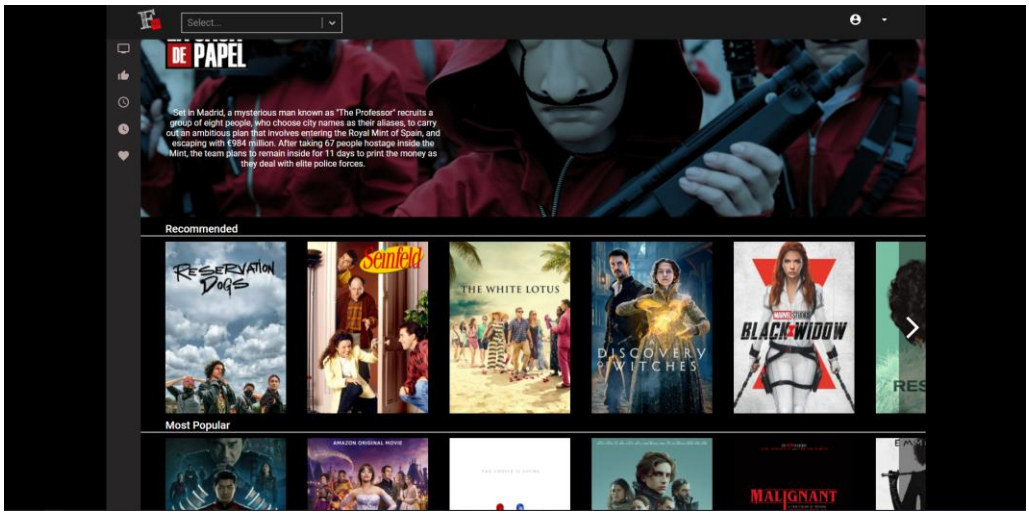
Folder App je ulazna točka stranice, to je prva stvar koju React učitava prilikom pozivanja stranice. Na slici 24. se može vidjeti svaka ruta unutar aplikacije. Vidi se da se rute „Navbar“ i „Sidebar“ učitavaju samo jednom prilikom prvog učitavanja stranice te se ostale rute učitavaju ovisno po potrebi. Neke od ruta koriste metode render iz razloga što imaju dodatnu logiku za učitavanje ruta. U ovom slučaju zahtijevaju od korisnika da bude prijavljen, drugim riječima da ima JWT token koji mu je servis poslao. Taj JWT token se nalazi u lokalnom skladištu preglednika, a dobiva se prilikom prijavljivanja ili registriranja. U slučaju da korisnik ne posjeduje JWT token, stranica se ponaša kao da korisnik nije logiran te mu limitira funkcionalnosti.

```
return (
  <div className="app_wrapper">
    <BrowserRouter>
      <div className="App" type="movie">
        <Navbar />
        <Sidebar />
        <Switch>
          <Route path="/" exact component={Home} ></Route>
          <Route path="/register" exact component={Register}></Route>
          <Route path="/login" exact component={Login}></Route>
          <Route path="/profile" render={() => requireAuth() ? <Route exact component={Profile} /> : <Redirect to="/login" />}></Route>
          <Route path="/movie" render={() => requireAuth() ? <Route exact component={Movie} /> : <Redirect to="/login" />} ></Route>
          <Route path="/movieList" render={() => requireAuth() ? <Route exact component={MovieList} /> : <Redirect to="/login" />} ></Route>
          <Route path="/serie" render={() => requireAuth() ? <Route exact component={Serie} /> : <Redirect to="/login" />} ></Route>
        </Switch>
      </div>
    </BrowserRouter>
  </div>
);
```

Slika 24. Browser router

5.2. Početna stranica

Početna stanica započinje unutar Home komponente koja poziva razne druge komponente. Sama Home komponenta ima vrlo malo koda pa se neće ulaziti u nju previše. Početna stranica se može vidjeti na slici 25. Stranice se razlikuju ovisno o tome da li je korisnik prijavljen ili ne. Slika ispod prikazuje stranicu prijavljenog korisnika iz razloga što neprijavljenim korisnicima stranica ne nudi preveliku funkcionalnost.



Slika 25. Početna stranica

Prilikom učitavanja početne stranice, poziva se funkcija useEffect koja je dio React okvira. Njezina funkcionalnost je da provjeri za koju od opcija je pozvana komponenta. Razlog tome je što se ista komponenta koristi za poziv svake od opcija, bilo to za najpopularnije filmove, filmove u kinima i slično. To se postiže tako što se u nju proslijedi naziv od opcije te ona uspoređuje nazive i priprema metode koje dohvaćaju popis filmova. Na posljetku se poziva funkcija fetchData, slika 26, koja prvo provjerava da li je korisnik prijavljen, u slučaju da je, poziva proslijeđenu metodu te sprema podatke u data varijablu.

```
const fetchData = async (call) => {
  const userId = auth.getUserId()
  let data;

  if (userId === "User not logged In.") data = await call();
  else data = await call(userId);

  if (data.message !== null) {
    if (data.message !== undefined) {
      if (data.message.items === undefined) {
        setData(data.message);
      }
      else {
        setData(data.message.items);
      }
    }
  }
};


useEffect(() => {
  setIsLoggedIn(auth.getAuthenticated())
  let mostPopular = type === "Most Popular" ? movies.mostPopular : null;
  let commingSoon = type === "Comming Soon" ? movies.commingSoon : null;
  let inTheaters = type === "In Theaters" ? movies.inTheaters : null;
  let recommeded = type === "Recommended" ? fetch.getRecommended : null;
  if (mostPopular && !commingSoon && !inTheaters && !recommeded) fetchData(mostPopular);
  if (!mostPopular && commingSoon && !inTheaters && !recommeded) fetchData(commingSoon);
  if (!mostPopular && !commingSoon && inTheaters && !recommeded) fetchData(inTheaters);
  if (!mostPopular && !commingSoon && !inTheaters && recommeded) fetchData(recommeded);

}, []);
```

Slika 26. useEffect i fetchData

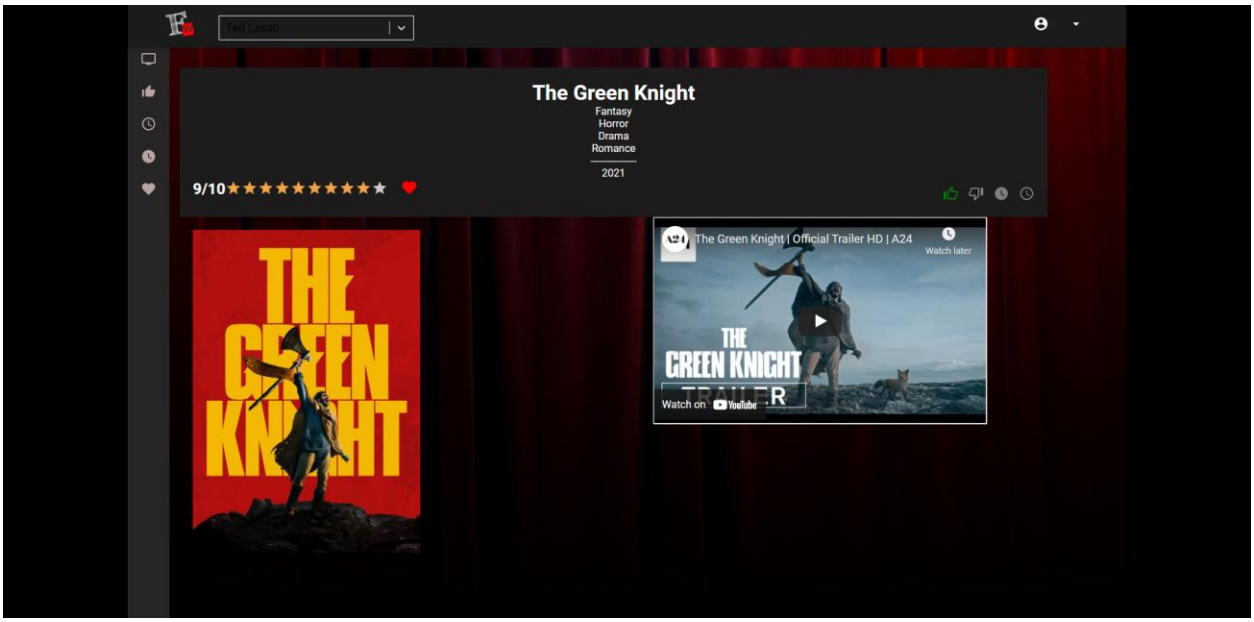
Nakon što je servis vratio podatke, provjerava se ponovno da li je korisnik prijavljen te ako je, dinamički se učitavaju popisi filmova, slika 27. Zajedno sa Link komponentom koja je isto dio React-a. U slučaju da korisnik klikne na jedan od filmova ili serija, Link komponenta ga prosljeđuje na stranicu o filmu ili seriji, ovisno o tome što je korisnik kliknuo.

```
type === "Recommended" ? !data.length ? <CircularProgress /> :
  data.map((x) => {
    let item = x.object_type === "movie" ?
      <div className="item" key={x.jw_entity_id
      }>
        {
          !isLoggedIn ? <div><ListItem key={x.jw_entity_id} title={x.title} image={x.poster} /></div> :
            <Link to={{
              pathname: "/movie",
              data: {
                id: x.id,
                type: "movie"
              }
            }} >
              <ListItem key={x.jw_entity_id} title={x.title} image={x.poster} />
            </Link>
        }

      </div>
      :
      <div className="item" key={x.jw_entity_id
      }>
        {
          !isLoggedIn ? <div><ListItem key={x.jw_entity_id} title={x.title} image={x.poster} /></div> :
            <Link to={{
              pathname: "/serie",
              data: {
                id: x.id,
                type: "show"
              }
            }}>
              <ListItem key={x.jw_entity_id} title={x.title} image={x.poster} />
            </Link>
```

Slika 27. Učitavanje filmova i serija

5.3. Stranica filma

Do movie stranice se dolazi klikom na film negdje na stranici. Slika 28. prikazuje izgled stranice na kojoj se može vidjeti jedan film te dodatni podatci o tom filmu. Isto tako se može vidjeti da odabrani film ima ocjenu 9 od 10 te da je stavljen u favorite i označen sa sviđa mi se. Svi ti podaci se spremaju u bazu podatka na servisu te se neki od tih koriste kako bi se gradili predlošci filmova u recommended opciji na početnoj stranici. Osim osnovnih podataka prisutan je i video o filmu koji se dobavlja sa youtube-a [21] te ako se radi o serijama, prikazane su i sezone te sve epizode koje korisnik može označiti da je pogledao što će se opet spremiti u bazu.



Slika 28. Movie stranica

Izvršavanje koda movie stranice započinje useEffect funkcijom koja je prethodno opisana. Slika koda se može vidjeti na slici 29. Unutar nje se dohvaćaju podaci koji su potrebni da bi se naknadno moglo dohvatiti i svi ostali podaci o filmu te podaci koje korisnik ima za taj film ili seriju. Isto tako se podaci spremaju u lokalno skladište pretraživača u slučaju da korisnik ponovno učita stranicu filmova ili direktno ode na stranicu upisujući u url nastavak movie. Kada se dobiju početni podaci, oni se provjeravaju da da li postoje, ako postoje poziva se funkcija fetchItemData.

```
156  useEffect(() => {
157
158    let data = props.location.data
159
160    data = data === undefined ? props.location?.state?.location?.data : data
161    localStorage.setItem("data", JSON.stringify(data));
162
163    if (data !== undefined) {
164      fetchItemData(data)
165    }
166    else {
167      setItem(JSON.parse(localStorage.getItem("item")))
168    }
169
170  }, []);
```

Slika 29. useEffect od movie komponente

Unutar fetchItemData funkcije se prvo dohvaća korisnikov id iz razloga što je on potreban da se napravi poziv prema servisu i dohvati ostale podatke. Nakon toga se provjerava da li id filma počinje sa „tt“, slika 30, linija 135. To je iz razloga što se koristi dva API-ja te oni imaju različite id-ove za filmove. Ovisno o id-u poziva se jedna ili druga funkcija, obje na kraju vraćaju iste podatke. Nakon što se dohvate podaci, poziva se druga funkcija koja mijenja boje ikona koje je korisnik označio te se ostali podaci koriste kako bi se prikazao sadržaj filma, slika i videa na stranici.

```
129   const fetchItemData = async data => {
130     const userId = auth.getUserId();
131
132     if (userId !== "User not logged In.") {
133       let response;
134       let id = data.id.toString()
135       if (id.startsWith("tt")) {
136         response = await fetch.getMovieImdbInfo(data.id, data.type, userId)
137       }
138       else {
139         response = await fetch.getMovieInfo(data.id, data.type, userId)
140       }
141       if (response.status === 200) {
142         const userData = response.message.userData.baseData
143         setItem(response.message);
144         if (userData !== null) {
145           updateColors(userData)
146         }
147
148         localStorage.setItem("item", JSON.stringify(response.message));
149       }
150       else {
151         setError(response.status);
152       }
153     }
154   }
```

Slika 30. fetchItemData funkcija

Ikone koje su korištene na stranici su dio material UI biblioteke. Uz pomoć te biblioteke potrebno je samo importati željenu ikonu i spremna je za korištenje. Prikaz korištenja jedne od ikona je na slici 31. Ovdje se koristi ikona srca čija je funkcija da korisnik klikne na nju i tako označi da mu je film među favoritima. Ikona mijenja boju u crvenu ili sivu ovisno o varijabli favoriteColor. Klikom na tu ikonu se poziva funkcija handleFavorite.

```
<FavoriteIcon
  className="favourite_icon"
  style={{ color: favoriteColor ? "red" : "grey" }}
  onClick={() => handleFavorite()}
/>
```

Slika 31. handleFavorite funckija

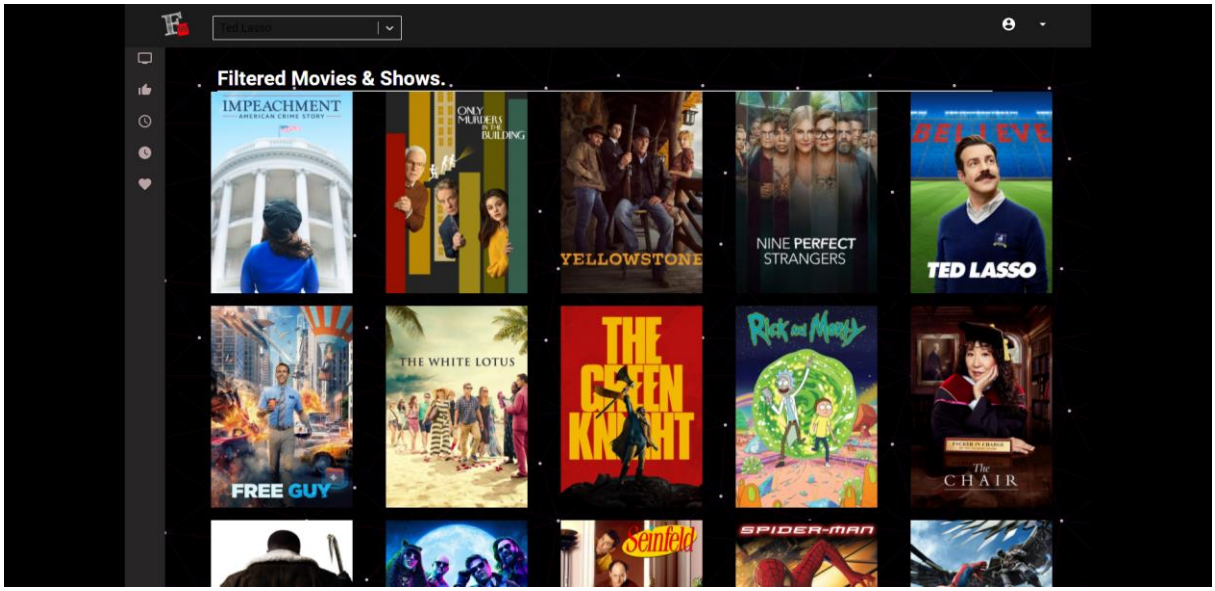
Funkcija handleFavorite mijenjanja vrijednosti varijable favoriteColor te dohvaća korisnikov id iz lokalnom skladišta pretraživača. Nakon toga poziva funkciju patchMovieuserData koja poziva servis na kojem se spremaju promjene o omiljenom filmu tog korisnika u bazu podataka. Slika koda handleFavorite funkcije se može vidjeti na slici 32.

```
const handleFavorite = async () => {
  const userId = auth.getUserId()
  if (userId !== "User not logged In.") {
    const fav = !favoriteColor
    setFavoriteColor(fav);
    const data = JSON.parse(localStorage.getItem("data"))
    const response = await patchUserData.patchMovieUserData(userId, data.id, data.type, {
      favorite: fav
    })
    if (response.status !== 200) {
      console.log(response.message);
    }
  }
};
```

Slika 32. handleFavorite funkcija

5.4. Lista filmova

Na lijevoj strani stranice nalazi se sidebar koji ima nekoliko ikona. Te ikone su isto dio materijal UI-a te klikom na njih, osim prve, korisnika se vodi na stranicu movie list koja prikazuje popis filmova ovisno o tome što je korisnik kliknuo. Prikazuju se filmovi i serije, slika 33. koje je korisnik označio sa sviđa mi se, pogledati ću kasnije, pogledano i favoriti. Kada se klikne na prvu ikonu, otvara se prozor u kojem korisnici mogu odabrati filtere za filmove koje žele pretraživati, filteri mogu biti po godinama, ocjeni, žanrovi te u konačnici da li želi filmove ili serije. Nakon što se klikne na submit, korisnika se vodi na movie list stranicu koja prikazuje filmove koji zadovoljavaju unesene filtere.



Slika 33. Movie list

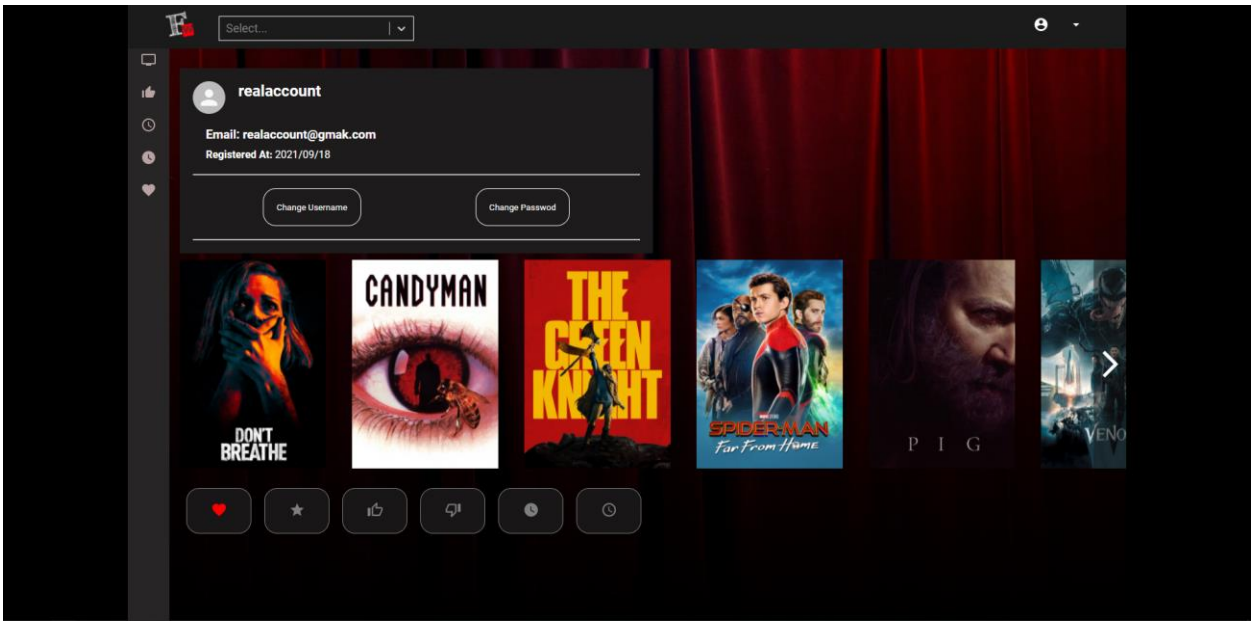
Slika 34. prikazuje implementaciju koda koji koristi filter, može se vidjeti da se unutar funkcije stvaraju nove varijable u koje se spremaju vrijednosti ovisno o tome da li je promijenjeno početno stanje varijabli koje su dio komponente. Te sve varijable se šalju u filter funkciju, neovisno da li imaju neku vrijednost ili im vrijednost ne postoji (null).Nakon poziva se svi podaci vraćaju na početno stanje.

```
const filterMovies = async () => {
  setShowDiv(!showDiv)

  let sendType = type.movie === true ? "movie" : type.show === true ? "show" : null
  let sendGenres = genres.length !== 0 ? genres : null
  let sendRatingMin = ratingMin !== "" ? ratingMin : null
  let sendRatingMax = ratingMax !== "" ? ratingMax : null
  let sendReleaseYearFrom = releaseYearFrom !== "" ? releaseYearFrom : null
  let sendReleaseYearTo = releaseYearTo !== "" ? releaseYearTo : null

  const response = await filters.filter(sendType, sendGenres, sendRatingMin, sendRatingMax, sendReleaseYearFrom, sendReleaseYearTo);
  let data = {};
  data.title = "Filtered Movies & Shows."
  data.items = response
  history.push("/movieList", data);

  setGenere([])
  setType({
    movie: false,
    show: false
  })
  setRatingMin(null)
  setRatingMax(null)
  setReleaseYearFrom(null)
  setReleaseYearTo(null)
}
```

Slika 34. Implementacija filtera u kodu

5.5. Profil

Klikom na ikonu profila na vrhu stranice desno, korisnika se vodi na stranicu njegovog profila, slika 35. Na profilu su prikazane neke osnovne informacije o korisniku kao što je njegovo korisničko ime, njegov e-mail i kada se prvi put registrirao. Isto tako prikazan je popis filmova te opcije iste kao što su na sidebar-u lijevo. Osim toga korisnik ima opcije mijenjanja korisničkog imena te lozinke. To su podaci koji se koriste za prijavu na stranicu pa se zato samo oni mogu mijenjati. Korisnik sa profila može kliknuti na film ili seriju što ga vodi na prijašnje prikazanu stranicu movie ili serije. U slučaju da korisnik posjećuje stranicu kao gost ili je odjavljen, korisnik neće imati pristup stranici profila.



Slika 35. Slika korisničkog profila

6. Zaključak

U svrhu završnog rada napravljena je web aplikacija koristeći poznati MERN stack, drugim riječima za izradu aplikacije su se koristile sljedeće glavne biblioteke i servisi: MongoDB [7], Express.js [25], React.js [4] i Node.js [24]. Primarni cilj rada je bila izrada besplatnog servisa koji ima mogućnost dohvaćanja filmova i serija te koji ima mogućnost praćenja korisničkih podataka vezanih za te serije i filmove. Za prikaz funkcionalnosti tog servisa izgradila se relativno jednostavna web stranica koja konzumira navedeni servis i pruža uvid u njegove mogućnosti. Korisnici mogu pratiti popise svojim omiljenih filmova i serija na jednom mjestu, zajedno sa uslugom predloška filmova koji je bio glavni fokus ovog rada. Osim korisnika, programeri mogu izrađivati svoje web stranice konzumirajući navedeni servis.

Pojedine funkcionalnosti aplikacije su ograničene kao što je dohvaćanje najpopularnijih filmova, filmova u kinima i nadolazećih filmova iz razloga što se koristi API koji je limitiran na 100 poziva dnevno. Postoji nekoliko točaka koje bi se mogle poboljšati kada dođe do aplikacije. Brzina učitavanja predloška filmova je relativno spora iz razloga što se za svaki film moraju dohvaćati svi podaci i zato što API nema opciju dohvaćanja samo slike i još nekoliko podataka, kao kad se filtrira. Nadogradnja za stranicu bi mogla biti implementacija komentara koje servis podržava i notifikacije koje u ovom primjeru nisu bile implementirane. Kod predloška filmova bi se možda mogla koristiti umjetna inteligencija za točnije predviđanje filmova koji bi se mogli svidjeti korisniku te bi se mogla implementirati još neka funkcionalnost koja bi se vrtjela oko korisničkog profila kao posjećivanje drugih profila da se vidi što drugi ljudi prate i vole, što bi indirektno isto služilo kao vrsta predloška filmova.

7. Literatura

[1] Mozilla.org, About JavaScript, dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, [pristupljeno 4.9.2021.]

[3] React Dokumentacija, Introducing JSX, dostupno na: https://reactjs.org/docs/introducing-jsx.html, [pristupljeno 4.9.2021.]

[4] React, Facebook Inc., 2021, dostupno na: https://reactjs.org , [pristupljeno: 10.9.2021]

[5] Cody Lindley, React JS - React Enlightenment, dostupno na: https://www.reactenlightenment.com/react-jsx/5.1.html, [pristupljeno 4.9.2021.]

[6] IBM, What is MongoDB?, dostupno na: https://www.ibm.com/cloud/learn/mongodb, [pristupljeno 4.9.2021.]

[7] MongoDB, MongoDB, dostupno na: https://www.mongodb.com/, [pristupljeno 4.9.2021.]

[8] MongoDB, What is MongoDB?, dostupno na: https://www.mongodb.com/what-is-mongodb, [pristupljeno 4.9.2021.]

[9] Express Dokumentacija, verzija 4.x, dostupno na: https://expressjs.com/en/4x/api.html , [Pristupljeno 1.09.2021.]

[10] Sass Dokumentacija, Sass team and numerous contributors dostupno na: https://sass-lang.com/documentation, [pristupljeno 12.9.2021.]

[11] Petar Ivanicevic, Movie Recommendation using JavaScript, dostupno na: https://blazingedge.io/blog/movie-recommendation-javascript/, pristupljeno [3.8.2021.]

[12] IMDb-API, IMDb-API, dostupno na: https://imdb-api.com/

[13] dawoudt, JustWatchAPI, dostupno na: https://github.com/dawoudt/JustWatchAPI

[14] Lama Dev, React Netflix Movie App Design Tutorial | React UI Full Course for Beginners, dostupno na: https://www.youtube.com/watch?v=FzWG8jiw4XM, pristupljeno [12.9.2021.]

[15] JustWatch, JustWatch, dostupno na: https://www.justwatch.com/, [pristupljeno 15.9.2021.]

[16] RottenTomatoes, RottenTomatoes, dostupno na https://www.rottentomatoes.com/, [pristupljeno 15.9.2021.]

[17] IMDb, IMDb, dostupno na: https://www.imdb.com/, [pristupljeno 15.9.2021.]

[18] Microsoft, Visual Studio Code, dostupno na: https://code.visualstudio.com/, [pristupljeno 15.9.2021.]

[19] GitHub, Inc., GitHub, dostupno na: https://github.com/, [pristupljeno 15.9.2021.]

[20] Software Freedom Conservancy, Git, dostupno na: https://git-scm.com/, [pristupljeno 15.9.2021.]

[21] MUI, Material UI, dostupno na: https://mui.com/, [pristupljeno 15.9.2021.]

[22] Axios Media, Axios, dostupno na: https://axios-http.com/, [pristupljeno 15.9.2021.]

[23] Google, YouTube, dostupno na: https://www.youtube.com/, [pristupljeno 15.9.2021.]

[24] OpenJS Foundation, NodeJS, dostupno na: https://nodejs.org/en/, [pristupljeno 15.9.2021.]

[25] OpenJS Foundation, ExpressJS, dostupno na: https://expressjs.com/, [pristupljeno 15.9.2021.]

Prilozi

Izvorni kod backend dijela aplikacije:
https://github.com/stopcagic/FindWatch/tree/master

Izvorni kod frontend dijela aplikacije: https://github.com/stopcagic/FindWatchFrontend

Popis slika