

Razvoj web aplikacije "Moji najdraži filmovi"

Nezić, Tihor

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **The Polytechnic of Rijeka / Veleučilište u Rijeci**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:125:593084>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-08-24**



Repository / Repozitorij:

[Polytechnic of Rijeka Digital Repository - DR PolyRi](#)



VELEUČILIŠTE U RIJECI

Tihor Nezić

RAZVOJ WEB APLIKACIJE „MOJI NAJDRAŽI FILMOVI“

(završni rad)

Rijeka, 2020.

VELEUČILIŠTE U RIJECI

Poslovni odjel

Preddiplomski stručni studij Informatika

RAZVOJ WEB APLIKACIJE „MOJI NAJDRAŽI FILMOVI“

(završni rad)

MENTOR

Izv. prof. dr. sc. Alen Jakupović, prof. v.š.

STUDENT

Tihor Nezić

MBS: 2422000053/16

Rijeka, studeni 2020.

VEUČILIŠTE U RIJECI

Poslovni odjel

Rijeka, 1.10.2020.

ZADATAK za završni rad

Pristupnik Tihor Nezić, MBS: 2422000053/16.

Studentu preddiplomskog stručnog studija Informatika izdaje se zadatak za završni rad – tema završnog rada pod nazivom:

RAZVOJ WEB APLIKACIJE „MOJI NAJDRAŽI FILMOVI“

Sadržaj zadatka:

Opisati alate, programske jezike i razvojne okvire korištene u razvoju web aplikacije: IDE Visual Studio Code, HTML, CSS, Node.js, Express.js, MongoDB, EJS i NPM. Opisati softversku arhitekturu MODEL-VIEW-CONTROLLER. Detaljno opisati uspostavu poslužiteljske i klijentske strane aplikacije. Opisati postavljanje NoSQL baze podataka. Opisati postavljanje aplikacije na Heroku i GitHub. Prikazati osnovno korištenje gotove web aplikacije.

Preporuka:

Rad obraditi sukladno odredbama Pravilnika o završnom radu Veleučilišta u Rijeci.

Zadano: 1.10.2020.

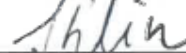
Predati do: 15.9.2021.

Mentor:



(izv.prof.dr.sc. Alen Jakupović, prof.v.š.)

Pročelnik odjela:



(mr.sc. Anita Štilin, v. pred.)

Zadatak primio dana: 1.10.2020.

Pristupnik:



(Tihor Nezić)

Dostavlja se:

- mentoru

- pristupniku

IZJAVA

Izjavljujem da sam završni rad pod naslovom „**RAZVOJ WEB APLIKACIJE „MOJI NAJDRAŽI FILMOVI“** izradio samostalno pod nadzorom i uz stručnu pomoć mentora Alena Jakupovića.

Ime i prezime

Tihomir Nežić
(potpis studenta)

SAŽETAK

U ovom završnom radu opisana je cjelokupna procedura izrade web aplikacije „Moji najdraži filmovi.“ Aplikacija je namijenjena za osobnu uporabu te kroz učitavanje i spremanje podataka o redateljima, glumcima i filmovima omogućuje pregled, uređivanje, brisanje korisnikovih najdražih filmskih naslova, redatelja i glumaca. Unutar aplikacije implementirane su funkcije za pretraživanje već spomenutih entiteta kao i neke dodatne funkcije koje su kao i svi drugi koraci vidljivi na priloženim slikama, te su uz obrazloženje korištenih alata i programskih jezika objašnjeni.

Aplikacija je nakon učitavanja na GitHub¹ podignuta na *eng. cloud* platformu Heroku kako bi se mogla koristiti i izvan lokalnoga servera.

KLJUČNE RIJEČI

HTML, JavaScript, Node.js, Express.js, MongoDB, Visual Studio Code, server, baza

¹ Platforma za verzioniranje, spremanje i dijeljenje kôda (GitHub, 2020.)

SADRŽAJ

1. UVOD	1
2. KORIŠTENI ALATI I JEZICI	2
3. NODE.JS, NPM I POSTAVKE PROJEKTA	4
3.1. Node.js	4
3.2. NPM.....	4
3.3. Osnovne postavke projekta	6
4. MODEL-VIEW-CONTROLLER	9
5. SERVER.JS	12
6. BAZA PODATAKA	14
6.1. Kreiranje shema	14
6.2. Izvoženje shemi i spajanje na lokalnu bazu podataka	16
7. LAYOUT.EJS	18
7.1. Header.ejs	19
8. PUTANJE (ROUTES)	21
8.1. Putanja glumaca.....	21
8.1.1. Putanja indeksa svih glumaca	22
8.1.2 Putanja dodavanja novoga glumca (GET)	25
8.1.3 Putanja dodavanja novoga glumca (POST)	26
8.1.4. Putanja za prikaz specifičnoga glumca	28

8.1.5. Putanja za uređivanje glumca.....	30
8.1.6. Putanja za brisanje glumca.....	31
9. MONGODB ATLAS	32
10. HEROKU	35
11. GITHUB I GIT	37
12. APLIKACIJA MOJI NAJDRAŽI FILMOVI	39
12.1. Opis aplikacije	39
12.2. Indeks stranica aplikacije.....	40
12.3. Stranica prikaza filma	41
12.4. Redatelji	42
12.4.1. Indeks stranica redatelja	42
12.4.2. Dodavanje redatelja.....	44
12.5. Glumci	45
12.5.1. Indeks stranica glumaca	45
12.5.2. Dodavanje glumca.....	46
12.6. Filmovi.....	48
12.6.1. Indeks stranica filmova	48
12.6.2. Dodavanje filma	49
12.7. Moji Top Filmovi.....	51
13. ZAKLJUČAK.....	52
14. LITERATURA	53

15. POPIS SLIKA.....	54
-----------------------------	-----------

1. UVOD

Ponukan idejom o vlastitom repozitoriju u kojemu bih imao pregled nad osobnim najdražim filmskim naslovima, kojima se mogu dodavati ocjene po vlastitome mišljenju, rangirati ih od najbolje ocijenjenih pa nadalje, motiviralo me kako bih implementirao svoje ideje i izradio web aplikaciju „Moji najdraži filmovi“. Aplikacija je izrađena (uz naravno HTML i CSS) unutar tzv. *eng. full-stack*² okruženja, specifičnije *stack*³-a koji uključuje tehnologije Node.js, Express.js koji su bazirani na JavaScript-u (za izradu *eng. backend* - serverske strane), baze podataka MongoDB koja je nerelacijska baza podataka, te korisničkog sučelja za čiju je izradu korišten *vanilla*⁴ CSS.

² Full-stack podrazumijeva razvoj i serverske strane i klijentske strane softvera (medium.com, 2020.)

³ Stack je kolekcija - okvir nezavisnih tehnologija koje rade zajedno kako bi se izradila aplikacija (pcmag.com)

⁴ Vanilla u informacijskoj tehnologiji označava običnu, osnovnu tehnologiju (whatis.techtarget.com, 2005.)

2. KORIŠTENI ALATI I JEZICI

Cjelokupni kod aplikacije pisan je u text-editoru Visual Studio Code koristeći HTML, CSS te spomenuti full-stack okvir kojega čine Node.js, Express.js i MongoDB, te uz to EJS.

Visual Studio Code je besplatni uređivač izvornog koda koji je izradio Microsoft za Windows, Linux i macOS. Prvo izdanje ovoga softvera je 2015. godina i otada postepeno postaje najpopularniji uređivač koda među *eng. developerima*⁵. Značajke uključuju podršku za otklanjanje pogrešaka, isticanje sintakse, inteligentno dovršavanje koda, isječke i ugrađeni Git⁶.

HTML (*eng. HyperText Markup Language*) je osnovni jezik za izradu web stranica koji preglednicima daje podatke o sadržaju i strukturi učitane web stranice, sastoji se od niza elemenata, a preglednik od tih podataka oblikuje stranicu kakvu mi vidimo. Definirao ga je 1990. godine Timothy Berners-Lee, danas ravnatelj *World Wide Web konzorcija (W3C)*. (Uvod u HTML, 2020.)

CSS (*eng. Cascading Style Sheets*) je stilski jezik za oblikovanje HTML elemenata kojima se uređuje svaki od njih. CSS-om se uređuje izgled HTML stranica, a pod time se podrazumijeva definiranje fontova, boja, margina između elemenata, animacija i sl. (CSS, 2020.)

Node.js server ili jednostavnije samo Node, baziran na JavaScript-u, skriptnom jeziku za preglednike uvedenoga još 1995., jest serversko okruženje (kojemu je najveći konkurent npr. PHP) za izgradnju brzih i skalabilnih aplikacija. Razvijen je 2009. g. od strane različitih developera. (Mihovilović, 2014.)

Express.js je *eng. framework*⁷ baziran u okviru Node.js servera, kojemu je svrha olakšati izgradnju web aplikacija. Godina njegove prve pojave je 2010. (Express.js, 2020.)

⁵ Razvijatelj, programer, developer (rječnik.com)

⁶ Git je distribuirani sustav koji služi za upravljanje i verzioniranjem izvornim kodom (Git, 2014.)

⁷ Framework podrazumijeva pojednostavljenje neke tehnologije, pri kojemu se ona fokusira na detalje na visokoj razini, umjesto na niskoj, a koje mogu definirati različiti programeri i developeri; više specificirana razina neke tehnologije (Framework, 2020.)

MongoDB je NoSQL program za manipuliranje bazom podataka. NoSQL podrazumijeva ne-relacijsku bazu pri kojemu su podatci spremljeni u obliku npr. ključ-vrijednost što je slučaj kod MongoDB-a. MongoDB koristi JSON dokumente za oblikovanje tablica, tj. shema. JSON, tj. *eng. JavaScript Object Notation* jest format za čitljivu razmjenu podataka (na već spomenuti princip ključ-vrijednost), koji je uz XML (*eng. Extensible Markup Language*) jedan od najkorištenijih i najpoznatijih. Prvotno se pojavio 2009. g. (MongoDB, 2020.)

EJS, *eng. Embedded JavaScript templates*, je jezik za generiranje HTML predložaka koristeći JavaScript. Kompatibilan je sa Express-om za backend uporabu. Njegovo korištenje omogućava vrlo brz i lak razvoj tzv. *eng. one-page-applications* koji je i razlog korištenja pri izradi ove web aplikacije. (ejs, 2020.)

3. NODE.JS, NPM I POSTAVKE PROJEKTA

3.1. Node.js

Kako bi se Node.js mogao koristiti za izradu serverske strane aplikacije potrebno je prvo preuzeti Node.js server sa njegove službene stranice. Tada je on nakon instalacije na lokalnoj mašini spreman za korištenje. Pošto je Node.js *eng. open-source, cross-platform* back-end okruženje koje je bazirano na JavaScript-u, on zvuči kao idealno rješenje za sve zaljubljenike u sada već najpopularniji jezik web developera, pošto se na bazi toga jezika kôd može izvršavati u web pregledniku, dakle za front-end, i izvan web preglednika što je slučaj kod Node.js-a, za već spomenuti back-end.

3.2. NPM

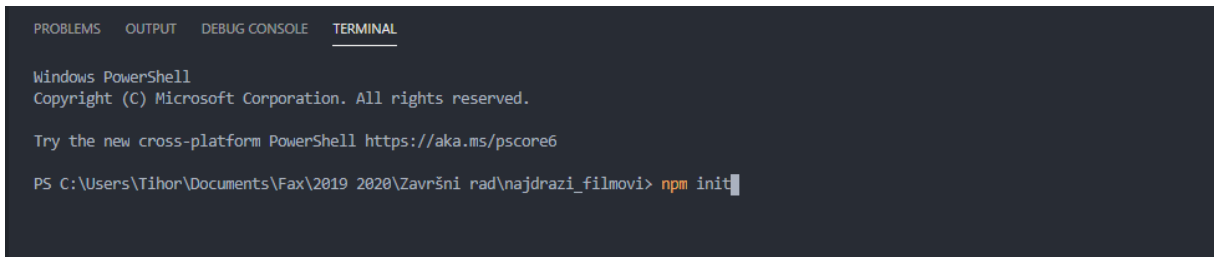
Jedna vrlo učinkovita, snažna, ali isto tako i jednostavna za korištenje značajka koje nudi Node.js je NPM, tj. *eng. node package manager*. NPM je upravitelj paketa za JavaScript koji je integriran sa Git-om i GitHub-om; dakle pomoću njega kôd je vrlo lako moguće spremati na on-line repozitorij, te se nadalje, NPM koristi za instalaciju i manipuliranje raznim dodatcima za razvoj softvera, instalaciju frameworko-va (npr. Express.js) i sl. (What is npm?, 2020.)

Ono što ga čini najpopularnijim upraviteljem paketa je činjenica da NPM nije nužno ovisan o Node-u, drugim riječima moguće je instalirati Node.js kako bi se mogao koristiti NPM i sve njegove mogućnosti koje pruža, ali ne koristiti sâm Node kao serversku stranu za aplikaciju ili sl.

Također, NPM ima mogućnost korištenja neovisno, dakle u vlastitom terminalu ili komandnoj liniji (konzoli) operativnog sustava ili u Visual Studio Code-u, jer je terminal ili komandna linija integrirana unutar samog VS Code-a.

Za početak rada u Node.js back-end okruženju ali i bilo kojem drugom okruženju ukoliko se koristi NPM upravitelj paketa, prvo se koristi naredba `npm init` kojom se stvori i inicijalizira datoteka `package.json` u kojoj se nalaze skripte za pokretanje servera, te su tamo prikazani svi *eng. dependencies*-i. *Dependencies*-i se mogu definirati kao paketi, ili neophodni „dodatci“ (treće partije ili sl.) koji olakšavaju ili su potrebni za razvijanje željenoga projekta; web stranice, aplikacije i sl. Npr., za razvoj ove web aplikacije kao *dependencies*-i su instalirani već spomenuti Node.js *eng. framework* Express.js, nadalje EJS i sl. Sve dosad spomenute naredbe redom su prikazane na slikama 1 i 2, a na slici 3 je prikazan izgled `package.json` datoteke ovoga cjelokupnoga projekta.

Slika 1: inicijalizacija projekta



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

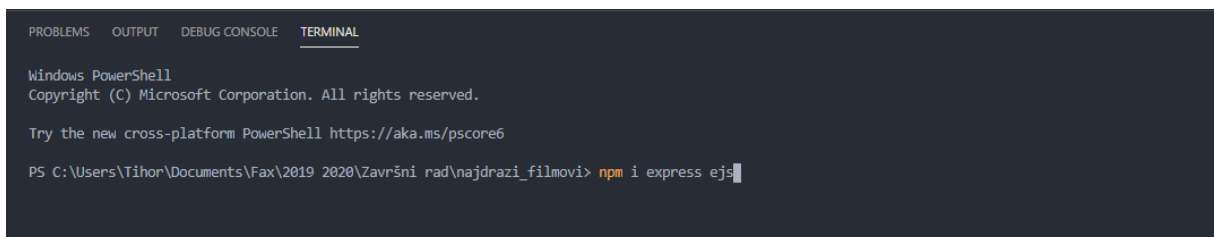
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrži_filmovi> npm init
```

Izvor: izradio autor

U jednom redu moguće je instalirati više različitih *eng. dependencies*-a, sve dok su oni odmaknuti razmakom.

Slika 2: Instalacija Express.js-a te EJS-a



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrži_filmovi> npm i express ejs
```

Izvor: izradio autor

Slika 3: *package.json* datoteka cijelog projekta

```
package.json X
package.json > ...
1 {
2   "name": "najdrazi_filmovi",
3   "version": "1.0.0",
4   "description": "",
5   "main": "server.js",
6   > Debug
7   "scripts": {
8     "start": "node server.js",
9     "devStart": "nodemon server.js"
10  },
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "ejs": "^3.1.5",
15    "express": "^4.17.1",
16    "express-ejs-layouts": "^2.5.0",
17    "method-override": "^3.0.0",
18    "mongoose": "^5.10.6"
19  },
20  "devDependencies": {
21    "dotenv": "^8.2.0",
22    "nodemon": "^2.0.4"
23  }
24 }
```

Izvor: izradio autor

3.3. Osnovne postavke projekta

Iz priložene slike 3 vidljivi su uz već opisane potrebne pakete i neki dosad nespomenuti: Method-override je *eng. library*⁸ koji omogućava korištenje PUT i DELETE zahtjeva, pošto inače, u pravilu, kod web preglednika se mogu koristiti samo GET i POST metode za manipuliranje podacima i komuniciranjem sa serverom.

Mongoose je *eng. Object Data Modeling (ODM) eng. library* za MongoDB i Node.js. Upravlja povezanošću među podacima, te se koristi za prevođenje JSON objekata u kôd i prikaz tih objekata u MongoDB-u.

Što se tiče u slici 3, *devDependencies*-a, to su oni *dependencies*-i koji se koriste samo lokalno, a njihovo takvo instaliranje može se postići korištenjem naredbe npr. `npm i --save-dev nodemon`.

⁸ Skup podataka i programskog koda koji se koristi kao pomoć za programere/developere za izradu softverskih programa i aplikacija (techopedia, 2016.)

Nodemon je značajka koja omogućava automatsko osvježavanje servera, pri čemu nije potrebno automatski „*refreshati*“ stranicu kako bi promjene na stranici bile vidljive.

Na kraju, instalacija `dotenv`-a omogućava korištenje `.env` datoteke u koju se upisuje kôd za kojega želimo da je privatna, i da nitko drugi osim na lokalnoj mašini nema njemu pristup.

Također, vrlo korisna odlika NPM-a je ta da se bez velike muke svi instalirani *dependencies*-i sa lokalne mašine mogu pokretati na nekoj drugoj, korištenjem naredbe `npm install` u konzoli, kojom se izvodi instalacija svih već ranije instaliranih *dependencies*-a sa lokalne mašine što omogućuje nesmetano pokretanje servera tj. cijele aplikacije.

Naredba kojom se zapravo pokreće server se nalazi pod „*scripts*“. Tamo se definira naziv varijable koju želimo upotrebljavati za spomenutu svrhu te zatim u nastavku što zapravo želimo da se pokreće unutar toga naziva. U ovom slučaju to je `nodemon` te `server.js`. Primjer toga je prikazan na slici 4.

Slika 4: definiranje varijable za pokretanje servera

```
"scripts": {  
  "start": "node server.js",  
  "devStart": "nodemon server.js"  
},
```

Izvor: izradio autor

Nakon toga se može pokrenuti server, naredbom `npm run devStart`, što je prikazano na slici 5:

Slika 5: pokretanje node.js servera

```
PS C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrazi_filmovi> npm run devStart
> najdrazi_filmovi@1.0.0 devStart C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrazi_filmovi
> nodemon server.js

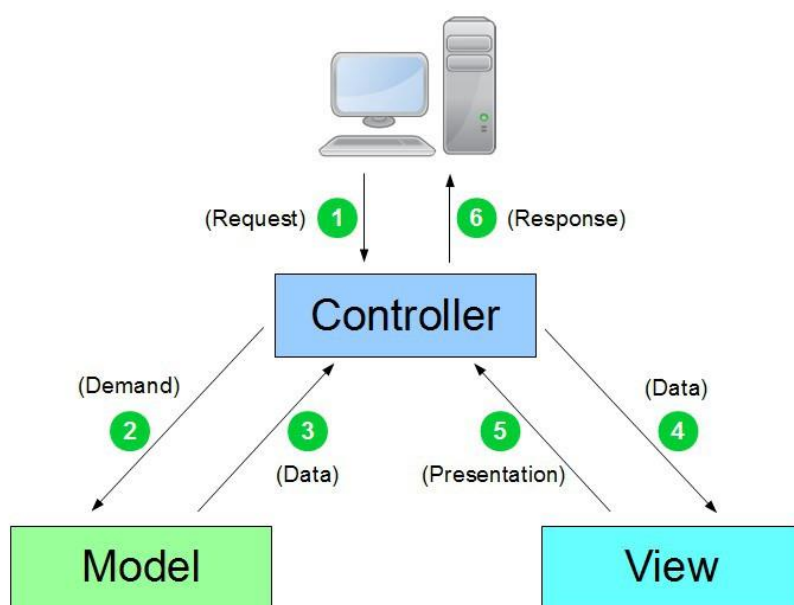
[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
```

Izvor: izradio autor

4. MODEL-VIEW-CONTROLLER

Da bi se nadalje mogla razumjeti struktura samoga projekta, potrebno je prvo razumjeti *eng. model-view-controller* arhitekturu na kojemu je bazirana izrada ove web aplikacije. MVC se može definirati kao obrazac za razvoj softverske arhitekture po kojemu se pojedini dijelovi aplikacije odvajaju u komponente ovisno o njihovoj namjeni. Provođenje definicije u grafički prikaz, pokazano je ukratko na slici 6:

Slika 6: MVC arhitektura



Izvor: https://miro.medium.com/max/875/1*Yc-MITc516auVOUz8EtGLw.jpeg

Glavna odlika MVC arhitekture, tj. modela je ta da su sve funkcionalnosti *eng. software-*a podijeljene na tri dijela koja se nalaze u određenom odnosu i komuniciraju jedan sa drugim na određeni način preko metoda. Razlog za korištenje takvoga načina izgradnje *eng. software-*a je radi preglednosti i pojednostavljivanja u snalaženju među (potencijalno mogućim) velikim brojem datoteka. MVC arhitektura implementirana je u gotovo svakoj modernoj web aplikaciji te je najpopularnija arhitektura korištena za izgradnju kompleksnih servera. (Veledar, 2019.)

Model upravlja manipulacijom nad podacima, dakle njihovim kreiranjem, modificiranjem te brisanjem. Iz toga, model vrši interakciju sa bazom podataka, a kontroler iz nje dobiva informacije upitom o određenim podacima.

View obavlja posao prezentiranja informacija, tj. predstavlja front-end stranu software-a. On najčešće (u modernim aplikacijama) dinamički kreira HTML stranice na osnovu podataka koje mu model dostavlja.

Za kontroler je bitno istaknuti da je zadužen za interakciju podataka između modela i view-a, tako da view i model nikada nisu u izravnoj komunikaciji jedan sa drugim. Kontroler, također, upravlja korisničkim upitima te razmjenjuje informacije između view-a i modela. On uvijek radi sa upitima, a nikad izravno sa podacima ili reprezentacijom sadržaja.

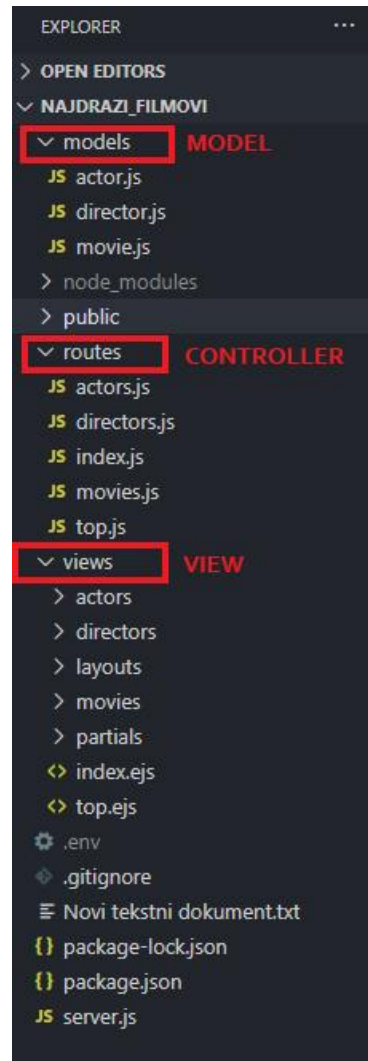
Primjer toka na koji način se ostvaruje MVC model koji je prikazan na slici 6: Korisnik šalje upit za određenu stranicu na serveru, dok server šalje sve potrebne informacije o upitu specifičnom kontroleru. Taj kontroler je zadužen za obrađivanje korisničkog upita i govori ostatku servera što točno činiti sa upitom. Prva stvar koja se događa kada kontroler primi upit je da šalje upit modelu za podatke bazirano na korisničkom upitu. Model tada vrši interakciju sa bazom podataka te povlači one podatke koje su tražene. Model zatim vraća kontroleru tražene podatke, a zatim vrši interakciju na view-om kako bi prikazao tražene informacije korisniku. View dinamički generira HTML predložak u kojemu se nalaze traženi podatci dobiveni od kontrolera. View tada vraća taj predložak nazad kontroleru, koji tada napokon odgovara korisniku na traženi upit.

Iz toga je lako zaključiti da kontroler uistinu ima ulogu kontrolera, odnosno kontrolira radom ostalih elemenata u MVC arhitekturi, interakcijom tj. međusobnom razmjenom podataka.

Konkretno, na primjeru na slici 7. je prikazan izgled MVC strukture ovoga projekta: Po uzoru na MVC, projekt se sastoji od modela u kojemu su modeli za glumca, redatelje i film; view-ovi sadrže prikaz za svaki model, a mapa *eng. routes* je u ulozi kontrolera. *Route* je sastavni dio svakog Node.js servera te je usko vezan uz rad servera, a predstavlja putanju kojoj će se moći pristupiti klikom na neku akciju (gumb, *eng. link*, sl.). Npr., aplikacija između ostalih, ima putanju (na primjeru *eng. localhost*⁹servera) <http://localhost:3000/directors>.

⁹ Označava pokretanje servera na lokalnoj mašini, zadani port Node.js localhost-a je 3000

Slika 7: MVC model primijenjen na projektu



Izvor: izradio autor

Nakon razumijevanja MVC modela po kojemu uzoru je razdijeljena aplikacija, druga vrlo važna stavka nakon postavljanja osnovnih postavki projekta je samo pokretanje servera; Već je spomenuto koja se naredba koristi za njegovo pokretanje, međutim server će ispravno raditi kada se prvotno postavi sami Express.js server te već spomenuti *eng. routes*-i. Vidljivo je na slici 7. kako postoji datoteka *server.js* koja je direktno povezana sa svim *eng. routovima*.

5. SERVER.JS

Pošto server mora imati *eng. routes-e*, njihov način povezivanja sa datotekom `server.js` je prikazano na slikama 8 i 9.:

Slika 8: Uključivanje *routes-a* u `server.js` datoteku

```
const indexRouter = require('./routes/index')
const directorRouter = require('./routes/directors')
const movieRouter = require('./routes/movies')
const actorRouter = require('./routes/actors')
const topRouter = require('./routes/top')
```

Izvor: izradio autor

Tipično za sintaksu Node.js servera i Express.js-a, koristi se *eng. „require“* pomoću kojih se može koristiti neka druga datoteka iz druge mape na projektu uz navođenje njegove putanje što je prikazano na slici 8.

Tada je potrebno koristiti te uvezene datoteke, koristeći tipične naredbe za osnovno postavljanje servera:

Slika 9: korištenje *use* naredbe za korištenje uvezenih *routes-a*

```
app.use('/', indexRouter)
app.use('/directors', directorRouter)
app.use('/movies', movieRouter)
app.use('/actors', actorRouter)
app.use('/top', topRouter)
```

Izvor: izradio autor

I na kraju, pošto svaki server mora imati port, kao kod konfiguriranja svakoga Express.js servera, koristi se linija kôda prikazana na slici 10.:

Slika 10: naredba kojom se postavlja port servera

```
app.listen(process.env.PORT || 3000)
```

Izvor: izradio autor

Kao što je moguće primijetiti, u server.js datoteci je vidljivo pojavljivanje *app* varijable. Ona je dio osnovnoga postavljanja samoga Express.js servera, koje je kao do sad već mnogo potrebnih ostalih spomenutih stvari nužno imati na samom početku postavljanja servera, a to je prikazano na slici 11.:

Slika 11: deklariranje korištenja express.js frameworka i ostalih prethodno instaliranih dependencies-a

```
const express = require('express')
const app = express()
const expressLayouts = require('express-ejs-layouts')
const bodyParser = require('body-parser')
const methodOverride = require('method-override')
```

Izvor: izradio autor

Nepisano pravilo (dobra praksa) je da se kod pozivanja korištenja Express.js-a ta varijabla i nazove *express*, a varijabla preko koje ćemo ga pozivati *app*. Uz to, isto tako potrebno je serveru dati do znanja da želimo koristiti sve prethodno instalirane *eng. dependencies-e*, a prikazano je na slici 12.:

Slika 12: set i use naredbe za korištenje i postavljanje prethodno instaliranih dependencies-a

```
app.set('view engine', 'ejs')
app.set('views', __dirname + '/views')
app.set('layout', 'layouts/layout')
app.use(expressLayouts)
app.use(methodOverride('_method'))
app.use(express.static('public'))
app.use(bodyParser.urlencoded({limit: '10mb', extended: false}))
```

Izvor: izradio autor

6. BAZA PODATAKA

6.1. Kreiranje shema

Kako bi se aplikacija mogla postaviti na već spomenuti Heroku, potrebno je prvo izraditi lokalnu bazu podataka, a zatim je izrađena *eng. cloud* baza podataka koristeći MongoDB Atlas koja će biti povezana sa postavljenom aplikacijom na Heroku-u.

Prvi korak je izgradnja same logike podataka, tj. sheme (koja je ekvivalent tablice u npr. MySQL-u), a bazirana je na JSON objektima, čiji su primjeri sva tri modela (za glumce, filmove i redatelje) izgrađeni koristeći spomenuti *eng. mongoose library*, vidljivi na slikama 13, 14 i 15.

Svaki JSON objekt je jedan stupac za spremanje određenih podataka, a odmaknuti su zarezom, a cjelokupna shema je spremljena u varijablu koja će se koristiti za izvoženje.

Slika 13: Mongoose JSON model glumaca

```
const mongoose = require('mongoose')

const actorSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },

  coverImage: {
    type: Buffer,
    required: true
  },

  coverImageType: {
    type: String,
    required: true
  },

  description: {
    type: String
  }
})
```

Izvor: izradio autor

Slika 14: Mongoose JSON model redatelj

```
const mongoose = require('mongoose')

const Movie = require('./movie')

const directorSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },

  description: {
    type: String
  }
})
```

Izvor: izradio autor

Slika 15: Mongoose JSON model filmova

```
const movieSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  // žanr
  genre: {
    type: String,
    required: true
  },
  // komentar/opis
  description: {
    type: String
  },
  // godina izlaska
  releaseYear: {
    type: Date,
    required: true
  },
  // trajanje
  duration: {
    type: Number,
    required: true
  },
  createdAt: {
    type: Date,
    required: true,
    default: Date.now
  },
  coverImage: {
    // buffer of the data representing our entire image, no longer string
    type: Buffer,
    required: true
  },
  // for indentifying image type
  coverImageType: {
    type: String,
    required: true
  },
  // redatelj
  // referencing director from our director collection model
  director: {
    // referencing another object inside of our collections
    // id of the director object
    type: mongoose.Schema.Types.ObjectId,
    // every movie has its director
    required: true,
    // since we're referencing something else (ObjectId), we need to tell mongoose what
    // we're referencing
    ref: 'Director'
  },
  actor: [
    {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Actor'
    }
  ],
  rating: {
    type: Number,
    required: true
  }
})
```

Izvor: izradio autor

6.2. Izvoženje shemi i spajanje na lokalnu bazu podataka

Za njihovo korištenje izvan datoteke u kojemu su definirani, tj. kako bi se mogli koristiti u server.js datoteci te tada spajati na lokalnu bazu podataka, potrebno ih je *eng. exportat*-i. Ta se radnja ostvaruje naredbom prikazanom na slici 16.:

Slika 16: primjer "exportanja" modela redatelja

```
module.exports = mongoose.model('Director', directorSchema)
```

Izvor: izradio autor

Prvi parametar metode `mongoose.model` je ime koje će biti kreirano u bazi podataka za određenu shemu. Isti se postupak izvodi za svaki model.

Nakon što su izrađeni i *eng. exportani* svi modeli, potrebno je spojiti ih na lokalnu bazu podataka. To se ostvaruje naredbama prikazanim na slici 17.:

Slika 17: spajanje na mongoose lokalnu bazu podataka

```
// mongodb setup
const mongoose = require('mongoose')
const { request } = require('express')
// connection to the DB
mongoose.connect(process.env.DATABASE_URL, {useNewUrlParser: true, useUnifiedTopology: true})
const db = mongoose.connection
db.on('error', error => console.error(error))
db.once('open', () => console.log('Connected to Mongoose'))
//
```

Izvor: izradio autor

Najvažnije je za istaknuti da se spajanje na lokalnu bazu ostvaruje preko datoteke `.env`, koju smo prethodno definirali kao datoteku koju ne želimo dijeliti zbog očuvanja privatnosti podataka, da ne dozvolimo drugim korisnicima da pristupe našoj bazi podataka. Ono što se nalazi u `.env` datoteci je prikazano na slici 18.:

Slika 18: prikaz sadržaja .env datoteke

```
DATABASE_URL = mongodb://localhost/mojifilmovi
```

Izvor: izradio autor

U ovom slučaju, pošto je riječ o spajanju na lokalnu bazu, pa nema problema pri prikazu na koju se bazu spaja, međutim, da je projekt spajan izravno na *eng. cloud* MongoDB bazu, nije dobra zamisao prikazivati URL baze. /mojifilmovi definira ime baze podataka.

Kako bismo se uvjerali (netom nakon pokretanja servera izvođenjem spomenute naredbe `npm run devStart`) da smo spojeni na lokalnu bazu, na slici 17 u zadnjoj liniji je naznačeno da se na uspješno spajanje u konzoli ispiše „Connected to Mongoose“, i to je vidljivo na slici 19.:

Slika 19: uspješno spajanje na lokalnu MongoDB bazu podataka

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrazi_filmovi> npm run devStart

> najdrazi_filmovi@1.0.0 devStart C:\Users\Tihor\Documents\Fax\2019 2020\Završni rad\najdrazi_filmovi
> nodemon server.js

[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Connected to Mongoose
```

Izvor: izradio autor

7. LAYOUT.EJS

Kako je za projekt korišten EJS, eng. *Embedded JavaScript templates*, koje u suštini naznačuje da će neki elementi biti prisutni na svim stranicama aplikacije, odnosno dinamički će se generirati HTML kôd, bez potreba kopiranja kôda na sve različite stranice na kojima želimo koristiti te određene elemente, njegovo je korištenje i sama lokacija datoteke naznačena na slici 12, konkretnije naredbe `app.set('views', __dirname + '/views')` te `app.set('layouts', 'layouts/layout')` koje definiraju da će se datoteka `layout` u mapi `layouts` koristiti kao `layout` (predložak) za cijelu aplikaciju, dakle sav kôd u toj datoteci će biti prisutan na svakoj stranici, odnosno putanji servera. Cijela `layout.ejs` datoteka projekta je prikazana na slici 20.:

Slika 20: `layout.ejs` datoteka

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link href="https://unpkg.com/filepond/dist/filepond.css" rel="stylesheet">
8   <link href="https://unpkg.com/filepond-plugin-image-preview/dist/filepond-plugin-image-preview.css"
9     rel="stylesheet">
10  <!-- defer = to load and run after the entire body is done rendering -->
11  <script defer src="https://unpkg.com/filepond-plugin-image-preview/dist/filepond-plugin-image-preview.js"></script>
12  <script defer src="https://unpkg.com/filepond-plugin-file-encode/dist/filepond-plugin-file-encode.js"></script>
13  <script defer src="https://unpkg.com/filepond-plugin-image-resize/dist/filepond-plugin-image-resize.js"></script>
14  <script defer src="https://unpkg.com/filepond/dist/filepond.js"></script>
15  <!-- -->
16
17  <script defer src="/javascripts/fileUpload.js"></script>
18  <link rel="stylesheet" href="/stylesheets/shared/fonts.css">
19  <link rel="stylesheet" href="/stylesheets/shared/variables.css">
20  <link rel="stylesheet" href="/stylesheets/shared/header.css">
21  <link rel="stylesheet" href="/stylesheets/shared/buttons.css">
22  <link rel="stylesheet" href="/stylesheets/shared/form.css">
23  <link rel="stylesheet" href="/stylesheets/shared/top.css">
24  <link rel="stylesheet" href="/stylesheets/shared/director.css">
25  <link rel="stylesheet" href="/stylesheets/shared/movies.css">
26  <link rel="stylesheet" href="/stylesheets/shared/actors.css">
27  <link rel="stylesheet" href="/stylesheets/main.css">
28  <title>Moji filmovi</title>
29 </head>
30
31 <body>
32   <div class="top-nav">
33     <%- include('../partials/header.ejs') %>
34   </div>
35   <div class="inner-container">
36     <%- include('../partials/errorMessage.ejs') %>
37     <%- body %>
38   </div>
39 </body>
40
41 </html>
```

Izvor: izradio autor

7.1. Header.ejs

Kao kod svake generičke HTML datoteke, u *eng. head* elementu se nalaze putanje na CSS i JavaScript datoteke, u svrhu navigiranja kroz aplikaciju, međutim najvažniji dio je onaj u *eng. body*-ju, jer se pomoću naredbe `<%- body %>` generira tijelo, tj. sâmi sadržaj svake *eng. view* (.ejs) određene datoteke, ovisno o kojoj putanji je riječ. Dakle kod EJS sintakse, ukoliko se želi uključiti neka vanjska datoteka, to se ostvaruje načinom prikazanim na slici 21 te na slici 22.:

Slika 21: uključenje header.ejs datoteke u layout.ejs datoteci

```
<%- include('../partials/header.ejs') %>
```

Izvor: izradio autor

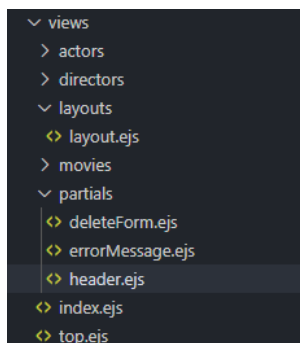
Slika 22: uključenje body-ja unutar layout.ejs datoteke

```
</div>  
<div class="inner-container">  
  <%- include('../partials/errorMessage.ejs') %>  
  <%- body %>  
</div>
```

Izvor: izradio autor

Kao što je vidljivo na slici 21, uključena je datoteka *eng. header* tj. zaglavlje stranice u kojoj se nalaze svi potrebni *eng. link*-ovi potrebni za navigiranje kroz aplikaciju. Struktura spomenutih datoteka i mapa je prikazana na slici 23, a na slici 24 je prikazana sama header.ejs datoteka.

Slika 23: struktura layout.ejs i header.ejs datoteka i mapa



Izvor: izradio autor

Slika 24: sadržaj header.ejs datoteke

```
<header>
  <nav class="header-nav">
    <a class="header-title" href="/">Moji najdraži filmovi</a>
    <ul>
      <li><a href="/directors">Redatelji</a></li>
      <li><a href="/directors/new">Dodaj Redatelja</a></li>
      <li><a href="/movies">Filmovi</a></li>
      <li><a href="/movies/new">Dodaj Film</a></li>
      <li><a href="/actors">Glumci</a></li>
      <li><a href="/actors/new">Dodaj Glumca</a></li>
      <li><a href="/top">Moji Top Filmovi</a></li>
      <hr class="line">
    </ul>
  </nav>
</header>
```

Izvor: izradio autor

Header.ejs datoteka sadržava *eng. header* HTML element koji sadrži *eng. link*-ove na sve različite stranice aplikacije, a oni su sačinjeni od spomenutih *eng. routes*-a na koje u nastavku slijedi fokus.

8. PUTANJE (ROUTES)

Netom prethodno spomenute putanje, tj. *eng routes*, bez kojih bi svaki Express.js server bio beskoristan, definiraju na kojim odredištima unutar aplikacije se nalaze specifične EJS izgenerirane HTML stranice u kojima se zapravo nalazi cijeli sadržaj.

Na primjeru jednog *eng. route*-a, zbog izbjegavanja ponavljanja i redundantnosti pošto je svaki *eng. route* napisan na isti način, biti će opisan rad putanje glumaca.

8.1. Putanja glumaca

Kako je bilo potrebno naznačiti korištenje Express.js-a kod postavljanja servera, tako je to potrebno učiniti i kod *eng. routes*-a. Uz to, putanje trebaju i posebnu metodu za omogućavanje njihovoga samog korištenja, a to je naredba `express.Router()` koja je u projektu spremljena u varijablu `router` preko koje će se izvršavati sve naredbe vezane za njega. A uz dodatak na to, uveženi su svi potrebni modeli neophodni za izvršavanje zamišljenih funkcija. To je prikazano na slici 25.:

Slika 25: korištenje Express.js-a te spremanje Router() varijable uz uvoženje potrebnih modela

```
1  const express = require('express')
2  const router = express.Router()
3  const Actor = require('../models/actor')
4  const director = require('../models/director')
5  const Movie = require('../models/movie')
```

Izvor: izradio autor

8.1.1. Putanja indeksa svih glumaca

Slika 26: Putanja index stranice glumaca

```
9 // all actors route
10 router.get('/', async (req, res) => {
11   let searchOptions = {}
12   if (req.query.name != null && req.query.name !== '') {
13     searchOptions.name = new RegExp(req.query.name, 'i')
14   }
15   try {
16     const actors = await Actor.find(searchOptions)
17     res.render('actors/index', {
18       actors: actors,
19       searchOptions: req.query
20     })
21   } catch {
22     res.redirect('/')
23   }
24 })
```

Izvor: izradio autor

Na slici 26 prikazana je putanja koja dovodi do indeks stranice glumaca. Nakon prethodno deklarirane varijable router, uz dodavanje točke, navodi se koja metoda se želi izvršiti, a postoje *eng. get, post, put* i *update* metode. U ovom slučaju u kojemu želimo samo dohvatiti informacije sa servera, korištena je *eng. get* metoda. Kao prvi parametar metode se u navodnike unosi sama putanja kojom ćemo moći pristupiti određenoj EJS datoteci koja se definira u nastavku, a pošto je na prethodnom prikazanom primjeru na slici 9, između ostalih putanja, definirano korištenje `/actors` putanje, daje značenje da će svakoj putanji unutar datoteke `actors.js`, (putanja glumaca) biti unaprijed nadodana ta vrijednost `/actors`, što bi na kraju značilo da se klikom unutar prethodno definirane `header.ejs` datoteke, klikom na 'Glumci' otvara u ovom konkretnom primjeru stranica `http://localhost:3000/actors/`.

Nadalje, drugi parametar, koji se nalazi u zagradi su `req` i `res`, tj. *eng. request* i *eng. response* servera preko kojih se mogu izvoditi određene potrebne metode.

Definiranje svake putanje, zbog ljepšeg izgleda kôda, ostvareno je korištenjem asinkronog pristupa koji koristi *eng. try catch statement*-e; unutar *eng. try*-ja se izvodi željeni kôd, a u slučaju pojavljivanja greške, odvija se *eng. catch* blok naredbi.

Slanjem zahtjeva serveru, odnosno slanjem link-a `http://localhost:3000/actors/`, server odgovara kroz spomenuti *eng. request* parametar. Na slici 27 metodom `res.render` je definirano koju stranicu želimo generirati kada korisnik želi pristupiti indeks stranici glumaca.

Slika 27: *render* metoda za prikaz index EJS stranice svih glumaca

```
17     res.render('actors/index', {
18         actors: actors,
19         searchOptions: req.query
20     })
```

Izvor: izradio autor

`Res.render()` metoda sastoji se od dva parametra; prvim se navodi putanja te naziv `.ejs` datoteke koju želimo generirati (nije potrebno dodavati `.ejs` na kraju naziva datoteke, server zna da je riječ o `.ejs` formatu), a drugim parametrom serveru dajemo objekt s čijim će podacima u ovom slučaju `index.ejs` moći raspolagati, tj. prikazivati ih. U objekt vrijednost-ključ (pošto je riječ o JSON objektu) spremljena je i varijabla `actors`, a koja je definirana prethodno, te je prikazana na slici 28.

Slika 28: deklariranje i inicijalizacija varijable `actors` koristeći `mongoDB` metodu `find()`

```
const actors = await Actor.find(searchOptions)
```

Izvor: izradio autor

Dakle u ključ vrijednost `actors` spremljena je varijabla `actors` koja je jednaka modelu `Actor.find(searchOptions)`, pri čemu je `find()` metoda `MongoDB`-a kojom se prikazuju željeni podatci iz baze, a njegov parametar `searchOptions` je prisutan zbog prikaza onih glumaca čiji se naziv pretražuje. Kako je ostvarena funkcionalnost pretraživanja je na slici 29.:

Slika 29: if statement za prikaz onih glumaca čiji se naziv pretražuje

```
let searchOptions = {}
if (req.query.name != null && req.query.name !== '') {
  searchOptions.name = new RegExp(req.query.name, 'i')
}
```

Izvor: izradio autor

Dakle funkcionalnost pretraživanja tj., eng. *search*-a implementirana je jednostavnom if provjerom. Ukoliko polje za pretraživanje glumaca kojim se pristupa kroz `req.query.name` nije prazno i ako nije jednako praznom eng. *stringu*, tada se ona postavlja na eng. *Regular expression* (sekvenca simbola koja pripada JavaScript-u) unesenoga imena zanemarujući velika i mala slova, što je ostvareno korištenjem 'i' u kôdu.

Ako korisnik nije ništa pretraživao, varijabla `searchOptions` će jednostavno ostati prazna i stranica `index.ejs` će prikazati sve glumce prisutne u bazi podataka.

`Index.ejs` datoteka kojom se generira stranica opisana u prethodnom opisu prikazana je na slici 30.:

Slika 30: `index.ejs` glumaca

```
1 <h2>Pretraži glumce</h2>
2
3 <form action="/actors" method="GET">
4   <div class="form-row">
5     <div class="form-item">
6       <label>Ime</label>
7       <input type="text" name="name" value="<%= searchOptions.name %>">
8     </div>
9   </div>
10  <div class="form-row form-row-end">
11    <button class="btn btn-primary" type="submit">Traži</button>
12  </div>
13 </form>
14
15 <br>
16 <br>
17
18 <div class="movie-grid">
19   <% actors.forEach(actor => { %>
20     <a href="/actors/<%= actor.id %> " ></a></a>
22     <% }) %>
23 </div>
```

Izvor: izradio autor

8.1.2 Putanja dodavanja novoga glumca (GET)

Slika 31: putanja za prikaz stranice za unos novoga glumca

```
// new actor route
router.get('/new', async (req, res) => {
  try {
    const movies = await Movie.find()
    const params = {
      actor: new Actor(),
      // movies: movies
    }
    res.render('actors/new', params)
  } catch {
    res.redirect('/')
  }
})
```

Izvor: izradio autor

Na slici 31 je prikazana putanja za generiranje stranice za dodavanje novoga glumca u bazu, koji je naravno vrlo sličan putanji za prikaz svih glumaca, osim nekoliko razlika;

Umjesto traženja određenih glumaca u bazi, kreira se novi, naredbom koja pripada MongoDB-u; `new Actor()`, koja se prosljeđuje datoteci 'new' koja se nalazi u mapi actors (`res.render('actors/new'), params`).

New.ejs datoteka za kreiranje novoga glumca je prikazana na slici 32.:

Slika 32: new.ejs za kreiranje glumca

```
<h2>Novi glumac</h2>
<form action="/actors" method="POST">
  <%- include('_form_fields')%>
  <div class="form-row form-row-end btn-row">
    <a class="btn btn-danger" href="/actors">Odustani</a>
    <button class="btn btn-primary" type="submit">Kreiraj</button>
  </div>
</form>
```

Izvor: izradio autor

Iz razloga što su .ejs datoteke za kreiranje i uređivanje glumaca gotovo isti, a i zbog dobre prakse kojom je kôd čišći, korištena je `_form_fields.ejs` datoteka koja je tada samo uključena unutar `new.ejs` datoteke. U njoj se zapravo nalazi forma za unos novoga glumca a prikazana je na slici 33.:

Slika 33: `_form_fields.ejs` datoteka - forma za unos novoga glumca

```
<div class="form-row">
  <div class="form-item">
    <label>Ime i Prezime</label>
    <input type="text" name="name" value="<%= actor.name %>">
  </div>
</div>
<div class="form-row">
  <div class="form-item form-item-no-grow">
    <label>Slika Glumca</label>
    <input type="file" name="cover" class="cover-size filepond">
  </div>
  <div class="form-item">
    <label>Opis</label>
    <textarea rows="4" cols="50" name="description"><%= actor.description %></textarea>
  </div>
</div>
```

Izvor: izradio autor

8.1.3 Putanja dodavanja novoga glumca (POST)

Prethodna metoda dodavanja novoga glumca bila je *eng. get* metoda kojom se samo dohvatila forma za unos novoga glumca sa servera. Stvarno spremanje unesenih podataka u formi je izvršeno *eng. post* metodom, prikazanom na slici 34.:

Slika 34: putanja za spremanje unesenih podataka

```
// post actor route
router.post('/', async (req, res) => {
  const actor = new Actor({
    name: req.body.name,
    description: req.body.description,
    // movie: req.body.movie
  })
  try {
    saveCover(actor, req.body.cover)
    const newActor = await actor.save()
    res.redirect('actors')
    // res.redirect(`actors/${newActor.id}`)
  } catch {
    res.render('actors/new', {
      actor: actor,
      errorMessage: 'Pogreška pri kreiranju glumca'
    })
  }
})
```

Izvor: izradio autor

Naredba kojom se uneseni podatci novoga glumca spremaju u bazu podataka je naredba `actor.save()`.

`saveCover` je funkcija kojom se učitana slika umjesto na server sprema na bazu podataka, iz razloga što se zbog besplatne opcije servera Heroku-a nakon resetiranja servera slika izgubi, a rješenje tomu je forsiranje spremanje slike na bazu podataka.

Slika 35: `saveCover` funkcija za spremanje slika na bazu podataka

```
function saveCover(actor, coverEncoded) {
  if (coverEncoded == null) return
  const cover = JSON.parse(coverEncoded)
  if (cover != null && imageMimeTypes.includes(cover.type)) {
    actor.coverImage = new Buffer.from(cover.data, 'base64')
    actor.coverImageType = cover.type
  }
}
```

Izvor: izradio autor

8.1.4. Putanja za prikaz specifičnoga glumca

Putanja za prikaz specifičnog, određenog glumca je prikazana na slici 36. Sâmi naziv putanje sadržava, tj. sastoji se od ID-ja svakog određenog glumca, kojim se raspoznaje svaki različiti glumac pošto svaki glumac dobiva automatsko generirani identifikator (ID) kojega stvara MongoDB pri kreiranju; i to je način na koji se prikazuje svaki različiti glumac: `router.get('/:id')`.

Slika 36: putanja za prikaz svakog određenog glumca

```
// show actor route
router.get('/:id', async (req, res) => {
  try{
    const actor = await Actor.findById(req.params.id)
    const movie = await Movie.find({actor: req.params.id})
    res.render('actors/show', {
      actor: actor,
      movie: movie
    })
  }catch{
    res.redirect('/')
  }
})
```

Izvor: izradio autor

Na slici 36 je vidljivo koja će se .ejs datoteka generirati kada se pristupi određenom glumcu, a ta show.ejs datoteka je prikazana na slici 37.:

Slika 37: show.ejs datoteka za prikaz glumaca

```
<h2><%= actor.name %></h2>
<br><br>
<div class="actor-details">
  <div>
    
      <a class="btn btn-primary" href="/actors/<%= actor.id %>/edit">Uredi</a>
      <%= include('../partials/deleteForm', {url: `/actors/${actor.id}`}) %>
    </div>
  </div>
</div>
<div class="actor-details-grid">
  <div class="actor-details-label">Pojavljivanja: </div>
  <div class="actor-details-text">
    <% if(movie.length > 0) { %>
    <% movie.forEach(movie => { %>
    <a href="/movies/<%= movie.id %>"><%= movie.title %></a>
    <% }) %>
    <% } else { %>
    <label>-</label>
    <% } %>
  </div>
  <div class="actor-details-label">Opis: </div>
  <div class="actor-details-text"><%= actor.description %></div>
</div>
</div>
```

Izvor: izradio autor

8.1.5. Putanja za uređivanje glumca

Slika 38: putanja za generiranje stranice za uređivanje glumca

```
// edit movie route
router.get('/:id/edit', async (req, res) => {
  try{
    const actor = await Actor.findById(req.params.id)
    res.render('actors/edit', {
      actor: actor
    })
  }catch{
    res.redirect('/actors')
  }
})
```

Izvor: izradio autor

Klikom na 'uredi' se aktivira navedena putanja kojom se generira edit.ejs datoteka prikazana na slici 39.:

Slika 39: .ejs datoteka za uređivanje glumca

```
<h2>Uredi Glumca</h2>

<form action="/actors/<%= actor.id %>?_method=PUT" method="POST">
  <%- include('_form_fields')%>
  <!-- anchor tags are used to link things -->
  <!-- buttons used to interact with forms or other objects -->
  <a class="btn btn-danger" href="/actors">Odustani</a>
  <button class="btn btn-primary" type="submit">Spremi</button>
</form>
```

Izvor: izradio autor

Ranije spomenuta datoteka `_form_fields` se iz ranije opisanih razloga također ovdje uključuje te se koristi *eng. put* metoda za *eng. update*-anje određenoga glumca.

8.1.6. Putanja za brisanje glumca

Zadnja putanja koja se nalazi u datoteci `actors.js`, je putanja za brisanje glumca. Njen izgled je prikazan na slici 40.:

Slika 40: putanja za brisanje glumca

```
// delete actor route
router.delete('/:id', async (req, res) => {
  let actor
  try{
    actor = await Actor.findById(req.params.id)
    await actor.remove()
    res.redirect('/actors')
  }catch{
    res.redirect('/')
  }
})
```

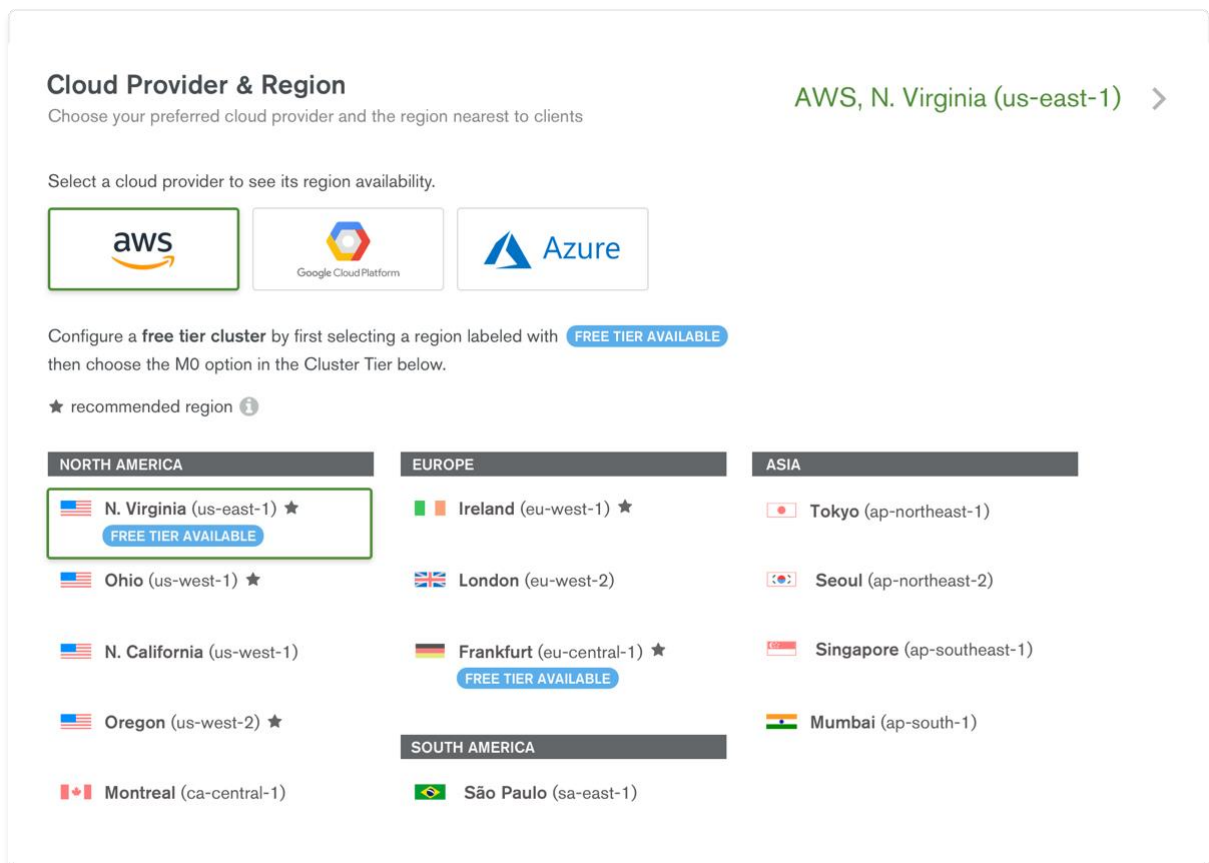
Izvor: izradio autor

9. MONGODB ATLAS

Dosad je u svrhu testiranja korištena lokalna MongoDB baza podataka. Za kreiranje *eng. cloud* baze podataka, koristi se MongoDB Atlas, koji se može povezati sa Heroku platformom.

Kako bi se ona stvorila ta baza, kreira se *eng. cluster*, te se odabire za koju regiju se želi kreirati. To je prikazano na slici 41.:

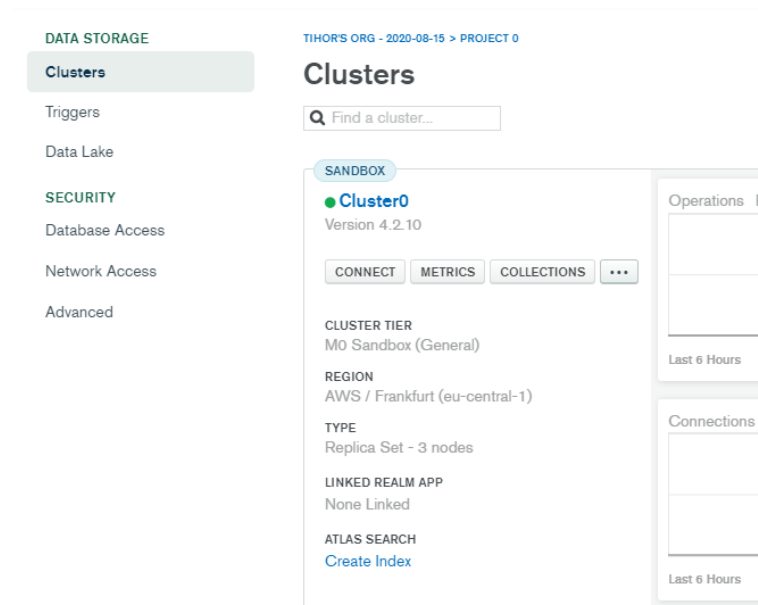
Slika 41: kreiranje clustera unutar MongoDB Atlasa



Izvor: <https://www.mongodb.com/assets/images/cloud/atlas/2018/cloud-provider.png>

Kada se *eng. cluster* kreira, ekran izgleda kao na slici 42. Tada je potrebno povezati tu bazu podataka sa Heroku-om, ali prije toga treba kopirati ključ kojim se ta veza ostvaruje.

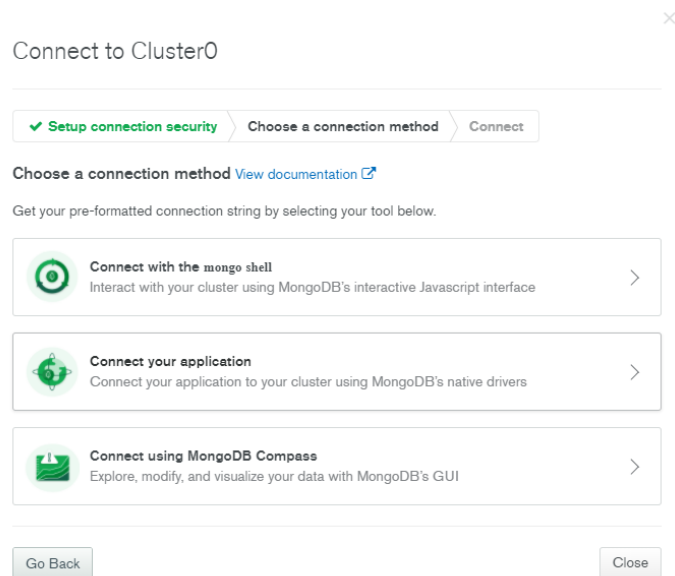
Slika 42: kreirani cluster



Izvor: izradio autor

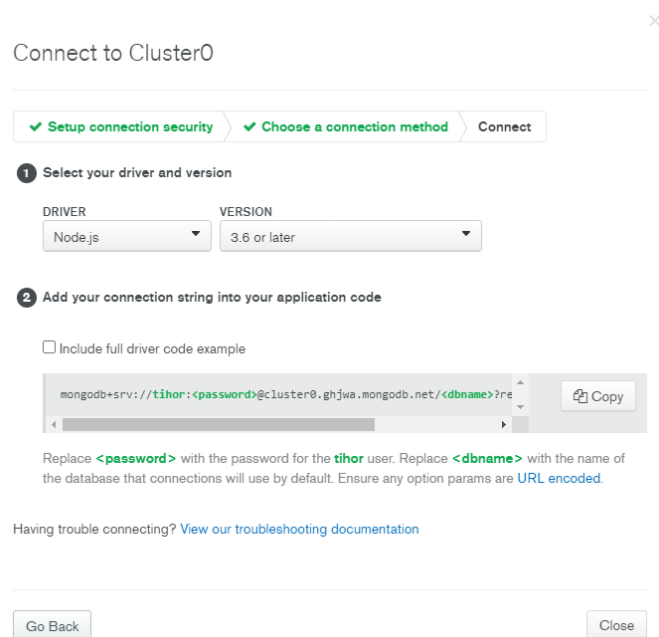
Kako bi se dobio taj ključ, klikom na 'Connect' se otvara prozor te se odabira opcija koju metodu konekcije želimo, a u ovom slučaju je odabrana 'Connect your application' te se tada dobiva ključ za povezivanje baze sa aplikacijom, u ovom konkretnom slučaju Heroku-om. To je prikazano na slikama 43 i 44.:

Slika 43: dostupne opcije spajanja clustera



Izvor: izradio autor

Slika 44: spajanje clustera na aplikaciju



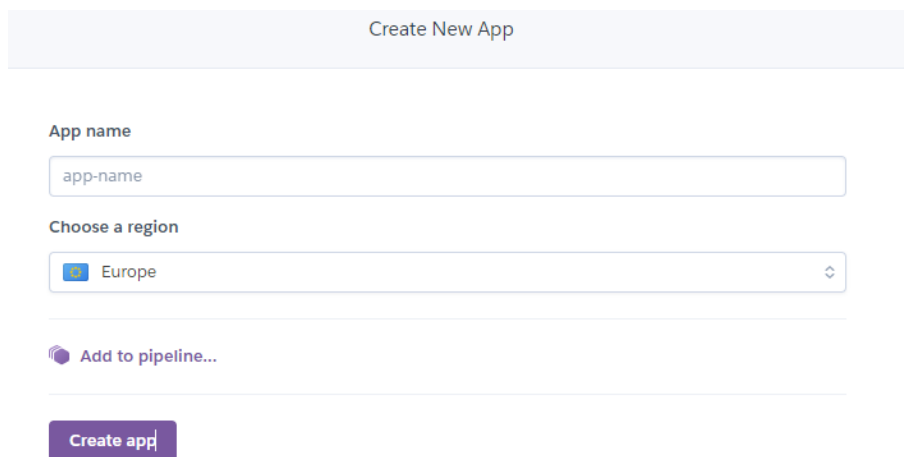
Izvor: izradio autor

Tada dobivamo za kopirati ključ za spajanje na aplikaciju u koju se unosi lozinka i naziv baze podataka.

10. HEROKU

Za podizanje aplikacije na Heroku, nakon kreiranja korisničkog računa, odabire se opcija 'Create new app'.

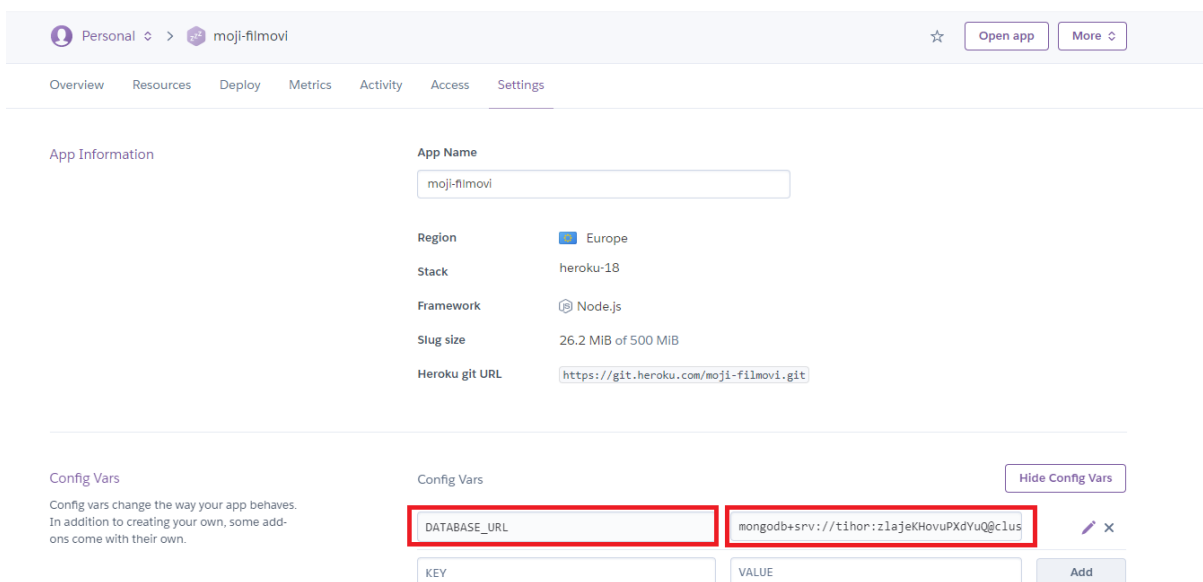
Slika 45: kreiranje aplikacije na Heroku-u



The screenshot shows the 'Create New App' form on Heroku. At the top, there is a header 'Create New App'. Below it, there is a section 'App name' with a text input field containing 'app-name'. Underneath is a section 'Choose a region' with a dropdown menu showing 'Europe'. Below the dropdown is a link 'Add to pipeline...'. At the bottom of the form is a purple button labeled 'Create app'.

Izvor: izradio autor

Slika 46: spajanje baze podataka na kreiranu aplikaciju na Heroku-u



The screenshot shows the Heroku app settings page for an app named 'moji-filmovi'. The page has a navigation bar with 'Personal' and 'moji-filmovi' tabs, and buttons for 'Open app' and 'More'. Below the navigation bar are tabs for 'Overview', 'Resources', 'Deploy', 'Metrics', 'Activity', 'Access', and 'Settings'. The 'Settings' tab is selected. Under 'App Information', there is a table with the following details: App Name (moji-filmovi), Region (Europe), Stack (heroku-18), Framework (Node.js), Slug size (26.2 MiB of 500 MiB), and Heroku git URL (https://git.heroku.com/moji-filmovi.git). Below this is the 'Config Vars' section, which has a 'Hide Config Vars' button. The 'Config Vars' section shows a table with two columns: 'KEY' and 'VALUE'. The first row has 'DATABASE_URL' in the key column and 'mongodb+srv://tihor:zIajeKHovuPXdYuQ@clus' in the value column. Both the key and value cells are highlighted with a red border. There is also an 'Add' button to the right of the table.

Izvor: izradio autor

Nakon kreiranja aplikacije, pod postavkama unutar 'Config Vars' unosi se prethodno kopirani ključ za spajanje na MongoDB Atlas bazu, i nakon toga aplikacija koja će biti podignuta na Heroku pomoću Git-a, će biti spojena na prethodno kreiranu *eng. cloud* bazu podataka.

11. GITHUB I GIT

Kako bi se sav kôd mogao podignuti na Heroku, koristi se Git, koji je prvo postavljen na *eng. online* repozitorij projekata Github. Dakle sav kôd ove web aplikacije je prvo postavljen na Github, a zatim se sa njega taj kôd postavlja na Heroku.

Prvo je potrebno kreirati repozitorij, te zatim sav kôd *eng. pushati* (standardna terminologija kod rada sa Githubom, koja označava postavljanje kôda na taj repozitorij) na njega. Nakon toga jednostavno je potrebno u konzolu kopirati naredbe pomoću kojih se inicijalizira projekt, te se kreiranom repozitoriju pridružuje lokalni kôd. Taj je proces prikazan na slikama 47 i 48.:

Slika 47: kreiranje Github repozitorija

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * / Repository name *

tihornezic / mojifilmovi ✓

Great repository names are **mojifilmovi is available.** Need inspiration? How about jubilant-bassoon?

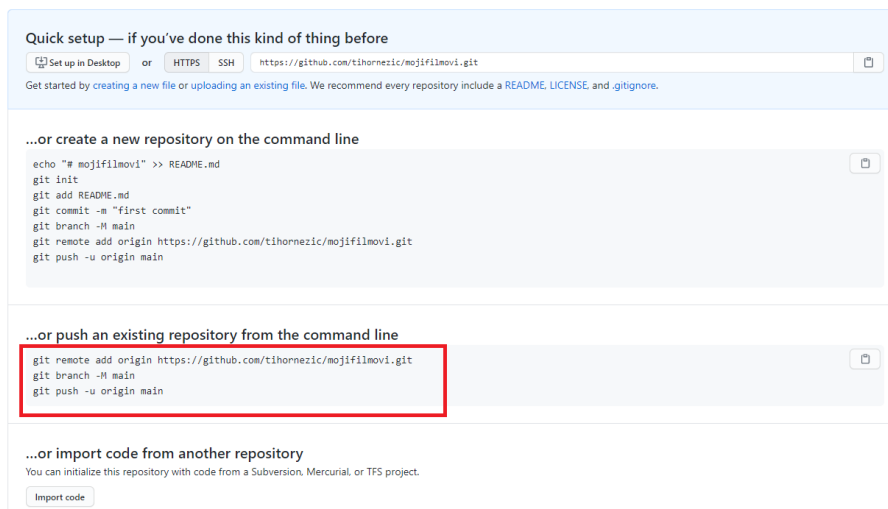
Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Izvor: izradio autor

Slika 48: naredbe za "pushanje" kôda na Github repozitorij



Izvor: izradio autor

Označene linije kôda kopiraju se u terminal unutar VS Code-a i pokreću se, čime je izvršen postupak „pushanja“ kôda na Github.

Prethodno tome, naravno, izvršene su standardne naredbe koje se koriste prije „pushanja“ za pripremu kôda za *eng. push*, a to su naredbe:

- `git init` (za inicijalizaciju projekta sa `git`-om)
- `git add .` (dodavanje svih datoteka u fazu spremu za izvršavanje)
- `git commit -m „Komentar“` (predavanje kôda sa nekim komentarom – kôd je spremljen lokalno)
- `git push -u origin master` („pushanje“ na Github na master granu)

Nakon uspješnoga „pushanja“ kôda na Github, moguće je sada podignuti taj kôd i na Heroku, a naredbe su:

- `heroku login` (za spajanje na Heroku korisnički račun)
- `heroku git:remote -a moji-filmovi` (davanje do znanja Heroku-u da ćemo „pushati“ kôd sa Githuba)
- `git push heroku master` („pushanje“ svog kôda na Heroku)

12. APLIKACIJA MOJI NAJDRAŽI FILMOVI

12.1. Opis aplikacije

Aplikacija je izrađena u svrhu spremanja osobnih najdražih filmskih naslova te pripadajućih glumaca i redatelja, omogućavajući uređivanje njihovih podataka kao što su opis, ocjena, slike, godina izlaska i sl.

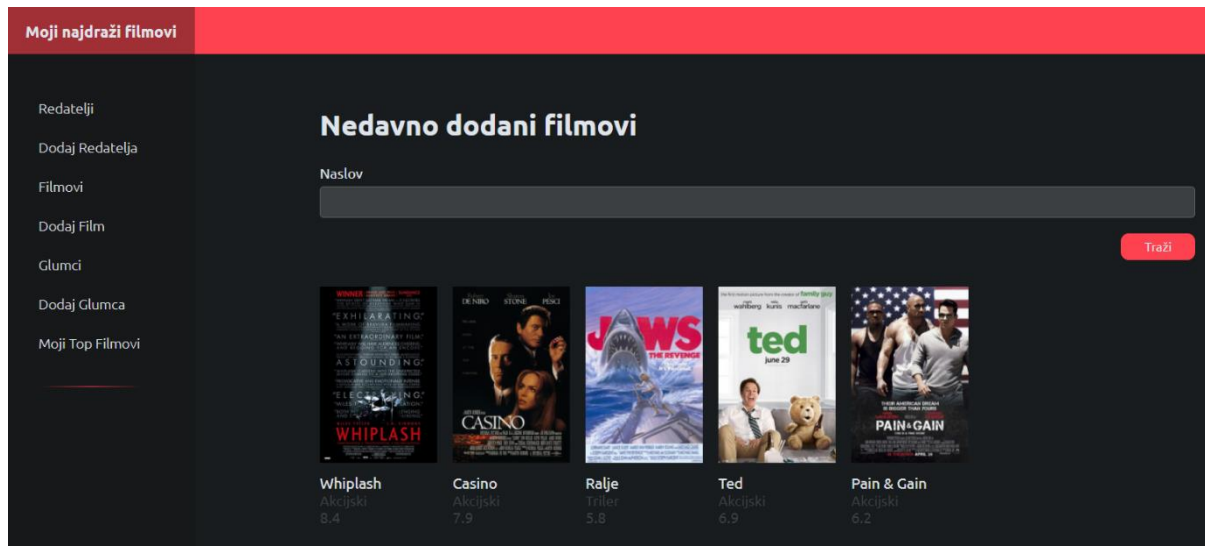
Prilikom ulaska na aplikaciju, na naslovnoj stranici nalaze se nedavno dodani filmovi, počevši od najnovijeg prema najstarijem, prikazani naslovnom slikom svakog učitanoog filma, zajedno sa podacima o naslovu, žanru i ocjeni. Prvi vrhu nalazi se tražilica koja omogućuje brzo traženje željenoga filma. Klikom na naslovnu sliku svakoga filma dolazi se do stranice sa prikazom detalja toga filma: naziv redatelja, godina izlaska, glavne uloge, ocjena, komentar, trajanje i žanr. Na toj stranici nalaze se tipke za uređivanje i brisanje tog filma, te tipka koja vodi do redatelja. Glumci, tj. uloge u tom filmu se mogu kliknuti te taj događaj vodi do stranice toga odabranoga glumca, sa svojim pripadajućim podacima: filmovi gdje se pojavljuje te opis i slika.

Slijeve strane nalazi se navigacijsko okno, koje sadrži tipke za prikaz redatelja, dodavanje redatelja, prikaz filmova, dodavanje filmova, prikaz glumaca, dodavanje glumaca i prikaz filmova prema ocjenama, umjesto kao na početnoj stranici po datumu dodavanja. Svaka od spomenutih stavki može se nakon dodavanja urediti ili obrisati po želji.

Aplikacija Moji najdraži filmovi je izrađena je u svrhu jednostavnog i brzog spremanja i pregledavanja i uređivanja osobnih najdražih filmova, glumaca i redatelja.

12.2. Indeks stranica aplikacije

Slika 49: prikaz indeks stranice aplikacije



Izvor: izradio autor

Na indeks (naslovnoj) stranici aplikacije nalaze se nedavno dodani filmovi, poredani od najnovijeg prema najstarijem. Svaki film je prikazan kao prethodno učitana naslovna slika filma, a ispod nje prikazani su podatci za svaki pripadajući film: naslov, žanr i ocjena. Iznad njih nalazi se tražilica za brzo traženje željenoga filma.

Za svrhu prezentiranja aplikacije, već je prethodno dodano nekoliko filmskih naslova, kao i redatelja i glumaca kako bi se stekao dojam izgleda nakon nekoliko dodanih filmova, a u nastavku je prikazan i primjer dodavanja svakog od njih.

Kao i na svakoj drugoj stranici, s lijeve strane nalazi se navigacijsko okno za navigaciju kroz aplikaciju.

Klikom na sliku jednog filma, otvara se indeks stranica toga odabranoga filma, sa njegovim pripadajućim unesenim podacima, što je prikazano na slici 50.

12.3. Stranica prikaza filma

Slika 50: stranica prikaza pojedinog filma

Moji najdraži filmovi

Redatelji

Dodaj Redatelja

Filmovi

Dodaj Film

Glumci

Dodaj Glumca

Moji Top Filmovi

Whiplash

WINNER: BEST MUSIC (MUSIC)
EXHILARATING
"AN EXTRAORDINARY FILM"
ASTOUNDING
"ELECTRIFYING"

Redatelj: Damien Chazelle

Godina izlaska: 2014

Trajanje: 97'

Žanr: Akcijski

Uloge: Miles Teller, J. K. Simmons

Ocjena: 8.4

Komentar: Andrew Neyman ambiciozni je mladi jazz bubnjar odluan u namjeri da se probije u sam vrh elitnog glazbenog konzervatorija na istonoj amerikoj obali. Terence Fletcher, profesor poznat po predavakom talentu koliko i po svojim zastraujuim metodama, otkrije Andrewa i ukljui ga u svoj sastav.

Uredi Obrisi Pogledaj Redatelja

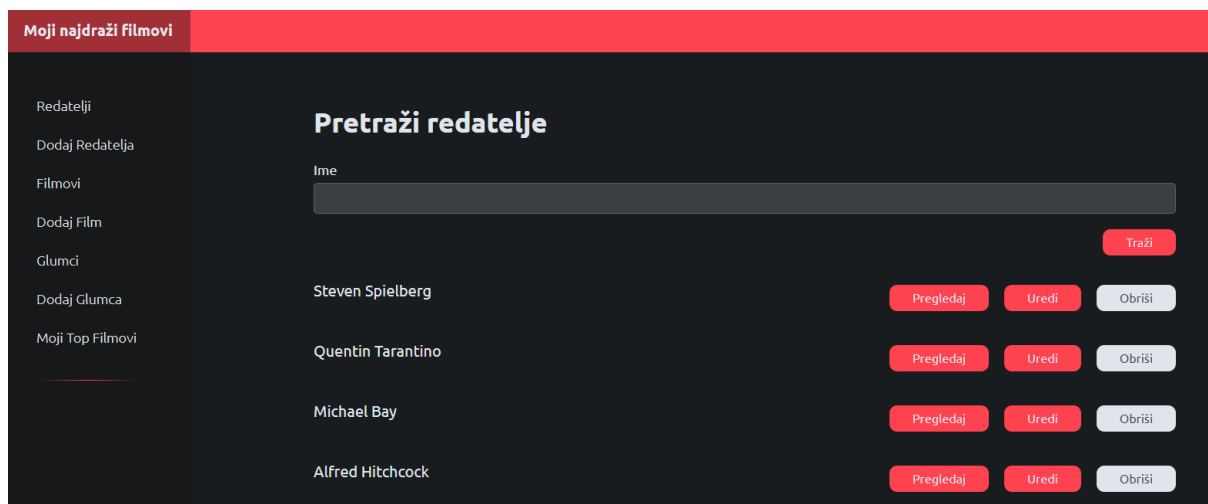
Izvor: izradio autor

Odabirom nekog filma otvara se njegova stranica sa pripadajućim podacima iz baze podataka: slika, redatelj, godina izlaska, trajanje, žanr, uloge, ocjena i komentar. Svaki film ima tipke za uređivanje, brisanje i 'Pogledaj Redatelja' kojim se otvara stranica sa prikazom redatelja toga filma. Iz slike 50 vidljivo je kako pod retkom uloge, tj. unesene glumce za taj film je moguće kliknuti, a ta radnja dovodi do stranice sa prikazom detalja toga odabranoga glumca.

12.4. Redatelji

12.4.1. Indeks stranica redatelja

Slika 51: indeks stranica svih redatelja

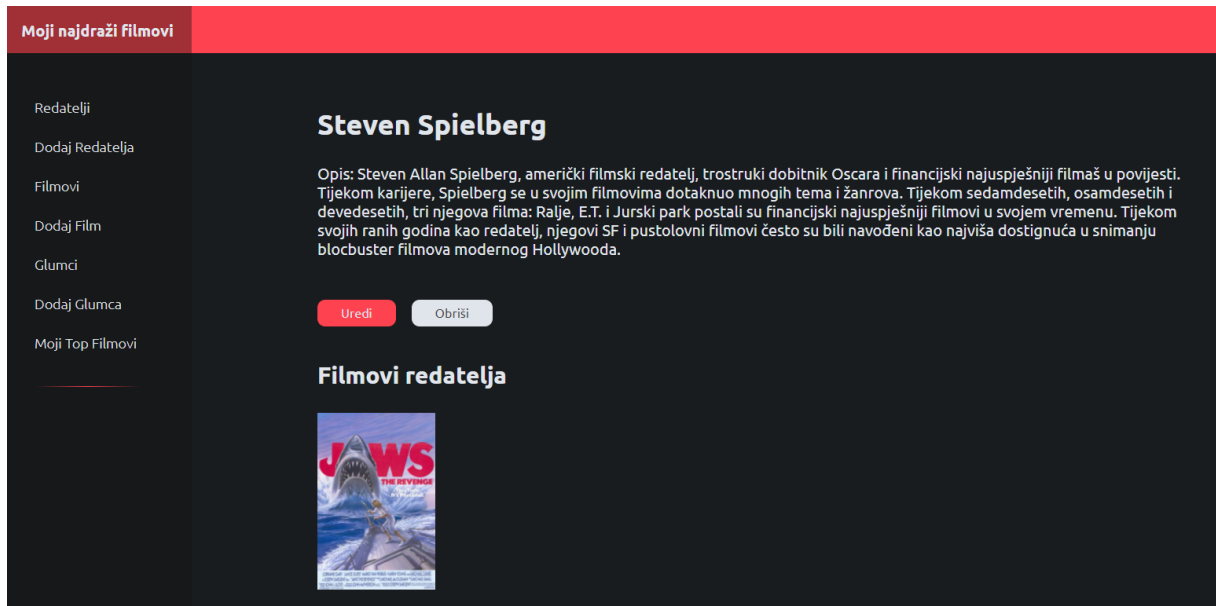


Izvor: izradio autor

Odabirom tipke 'Redatelji' unutar navigacijskog okvira, otvara se indeks stranica redatelja, sa popisom svih unesenih redatelja. Pri vrhu se kao kod filmova nalazi tražilica za brži pristup željenom redatelju. Svaki različiti redatelj ima mogućnost pregleda, uređivanja i brisanja.

Klikom na 'Pregledaj' otvara se stranica sa detaljima odabranoga redatelja, te se i unutar nje kao i na indeksu može urediti sadržaj, tj. uneseni podatci. Primjer odabira na tipku 'Pregledaj' jednog redatelja je prikazana na slici 52.:

Slika 52: primjer prikaza pregleda redatelja

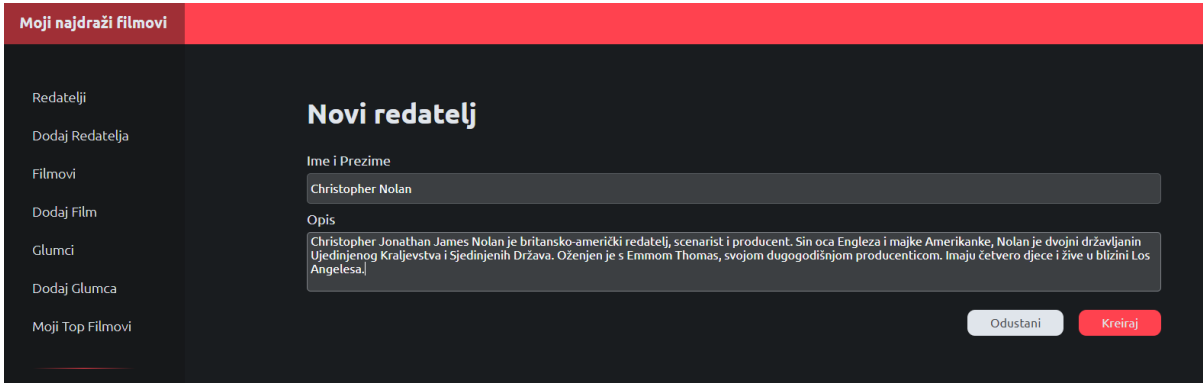


Izvor: izradio autor

Iz iznad priložene slike vidljivo je kako se pregledom određenoga redatelja uz opis, slikom prikazuju svi filmovi toga redatelja unesene unutar aplikacije. Za konkretan primjer redatelja Spielberga, u aplikaciju je unesen jedan njegov film, Ralje. Naravno, moguć je odabir slike filma koji dovodi do njegove stranice, koji izgleda na isti već prethodno opisan način (slika 50).

12.4.2. Dodavanje redatelja

Slika 53: dodavanje novog redatelja

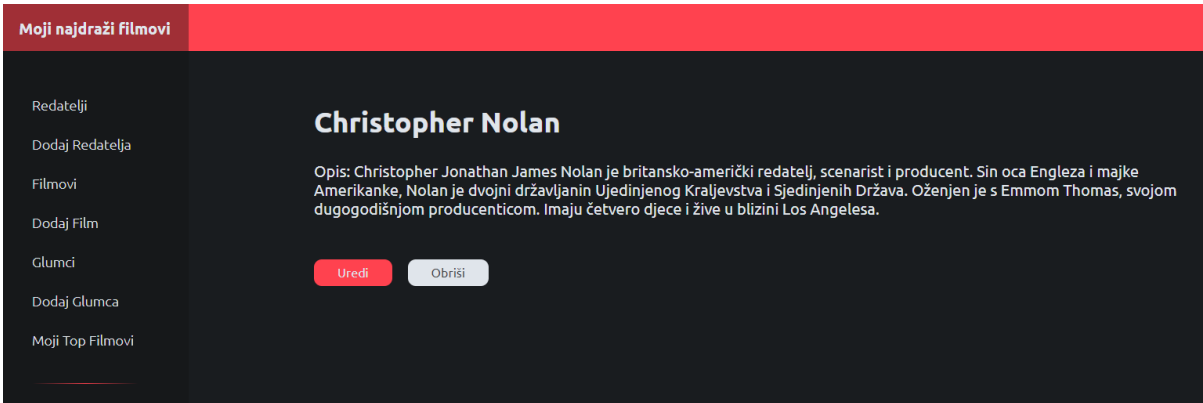


The screenshot shows a web interface for adding a new director. On the left is a sidebar menu with options: Redatelji, Dodaj Redatelja, Filmovi, Dodaj Film, Glumci, Dodaj Glumca, and Moji Top Filmovi. The main content area is titled 'Novi redatelj'. It contains a form with two input fields: 'Ime i Prezime' with the value 'Christopher Nolan' and 'Opis' with a detailed biographical text about Christopher Nolan. At the bottom right of the form are two buttons: 'Odustani' (grey) and 'Kreiraj' (red).

Izvor: izradio autor

Klikom na tipku 'Dodaj redatelja', otvara se stranica prikazana na slici 53 te se može unijeti novi redatelj. Nakon pritiska na tipku 'Kreiraj' otvara se stranica sa detaljima kreiranog redatelja, kao na slici 54.:

Slika 54: kreirani redatelj stranica detalja



The screenshot shows the details page for the director Christopher Nolan. The sidebar menu is the same as in Slika 53. The main content area is titled 'Christopher Nolan'. Below the title is a biographical 'Opis' paragraph. At the bottom of the page are two buttons: 'Uredi' (red) and 'Obriši' (grey).

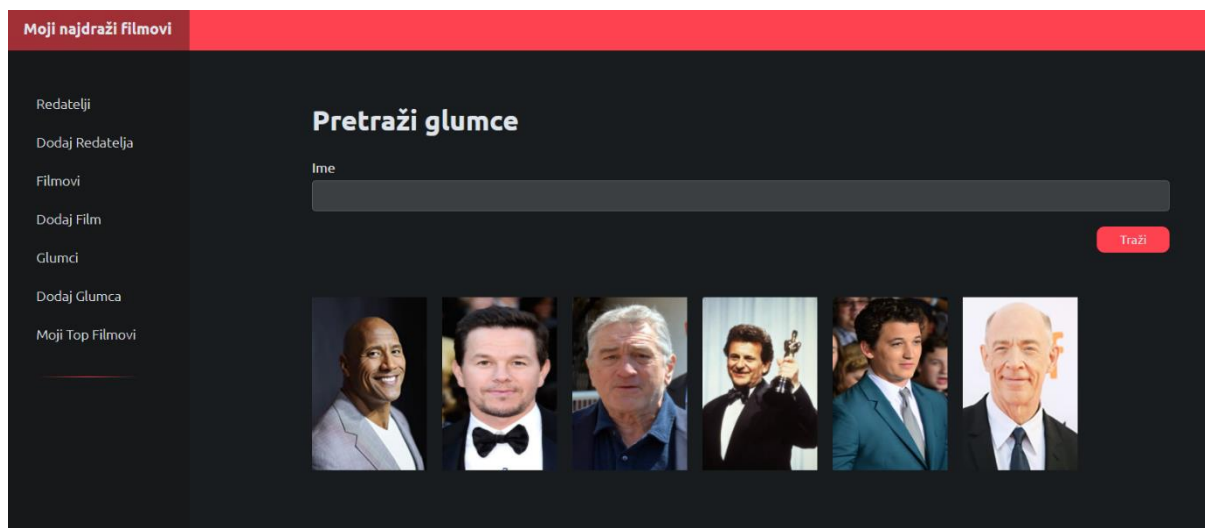
Izvor: izradio autor

Stranica nakon kreiranja redatelja također ima tipke koje omogućuju naknadno uređivanje i brisanje redatelja.

12.5. Glumci

12.5.1. Indeks stranica glumaca

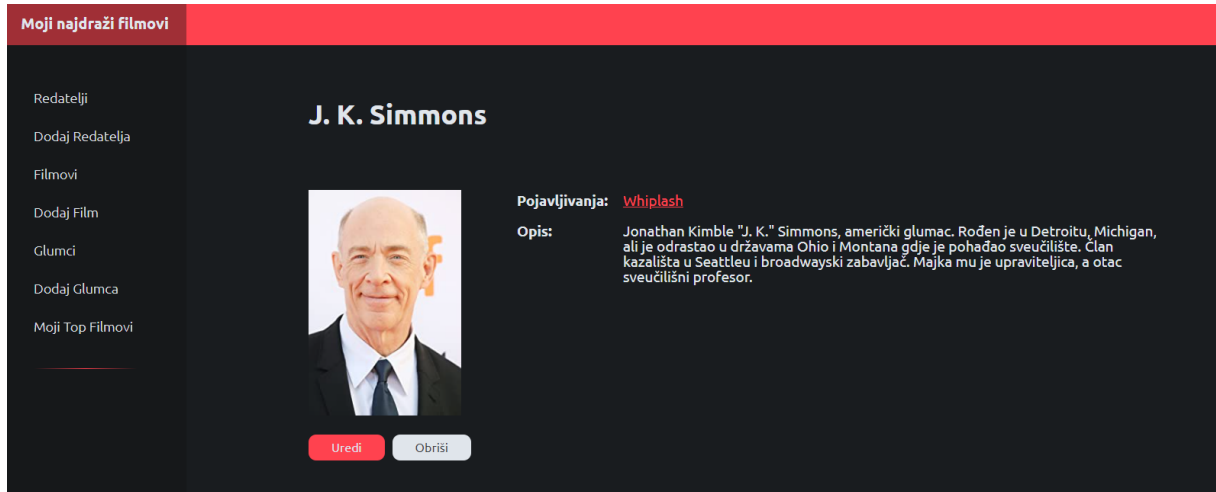
Slika 55: indeks stranica glumaca



Izvor: izradio autor

Odabirom tipke 'Glumci' unutar navigacijskog okvira, otvara se indeks stranica sa prikazom svih unesenih glumaca. Iznad njih nalazi se tražilica. Kao i kod filmova, klikom na sliku određenog glumca, otvara se stranica sa njegovim detaljima, što je prikazano na slici 56.:

Slika 56: pregled detalja glumca

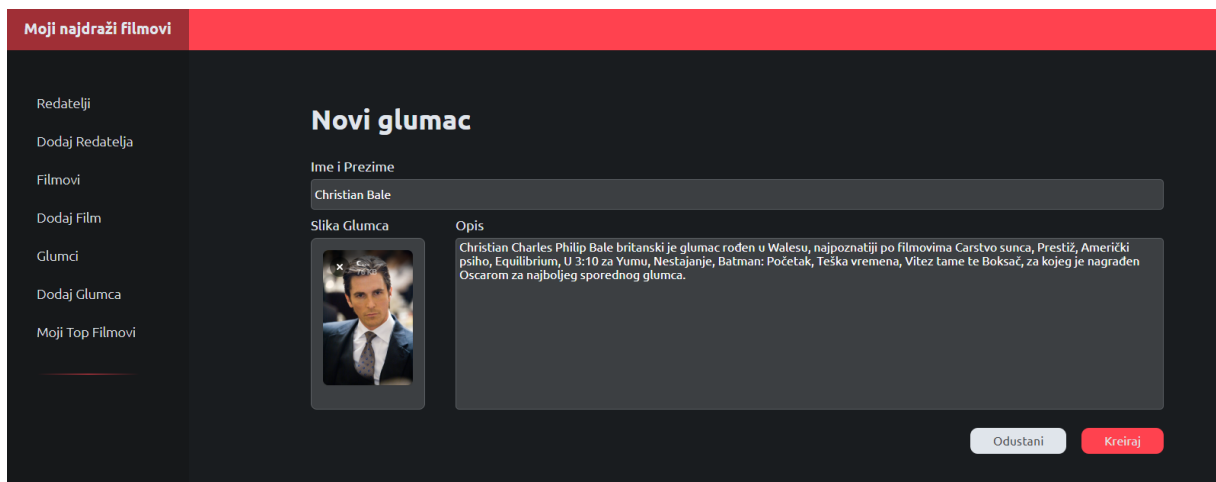


Izvor: izradio autor

Odabirom nekog glumca, uz sliku i opis, vidljivo je i može se kliknuti na film na kojemu se glumac pojavljuje, kao što je slučaj kod prikaza detalja filmova, gdje se odabirom glumca pod 'Uloge' dolazi na ovu stranicu prikaza detalja o glumcu (slika 50).

12.5.2. Dodavanje glumca

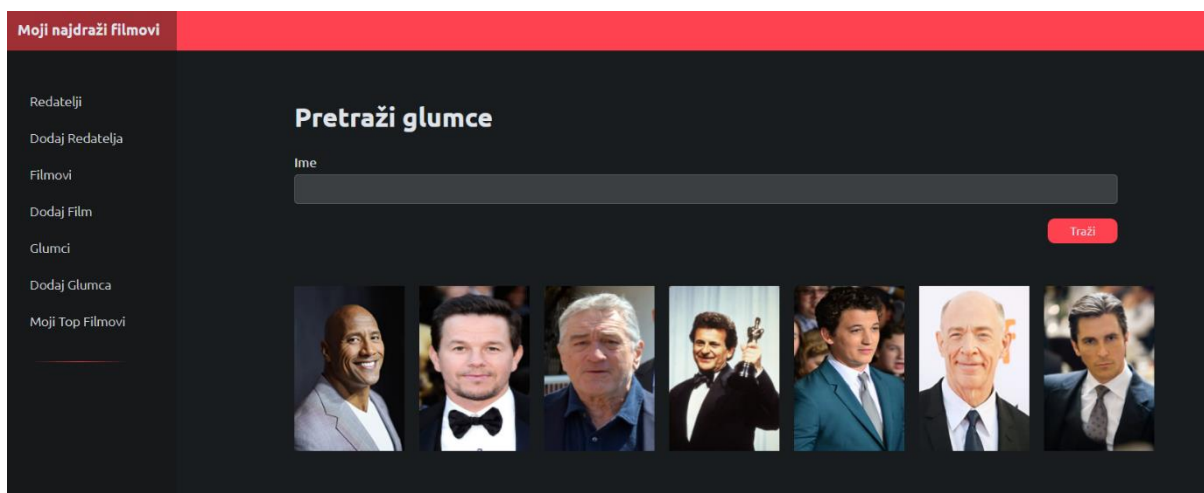
Slika 57: prikaz dodavanja novog glumca



Izvor: izradio autor

Nakon unosa podataka i učitavanja slike, klikom na 'Kreiraj', kreira se glumac u bazi i korisnika se preusmjerava na indeks stranicu svih glumaca pri čemu se vidi novo unešeni glumac. Slika 58.:

Slika 58: prikaz novounesenog glumca na indeks stranici glumaca

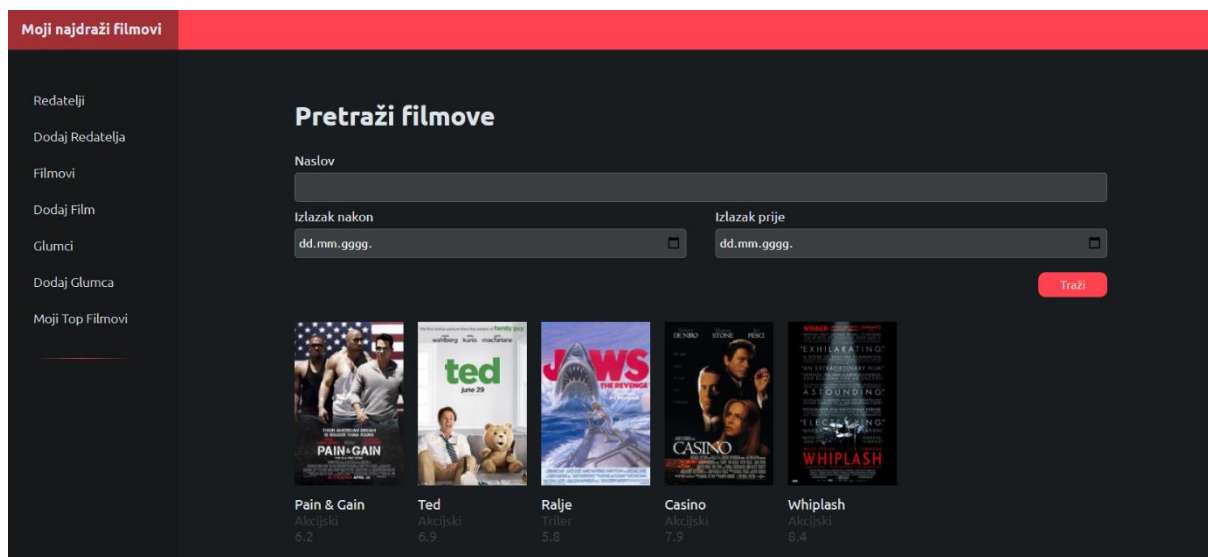


Izvor: izradio autor

12.6. Filmovi

12.6.1. Indeks stranica filmova

Slika 59: indeks stranica svih filmova

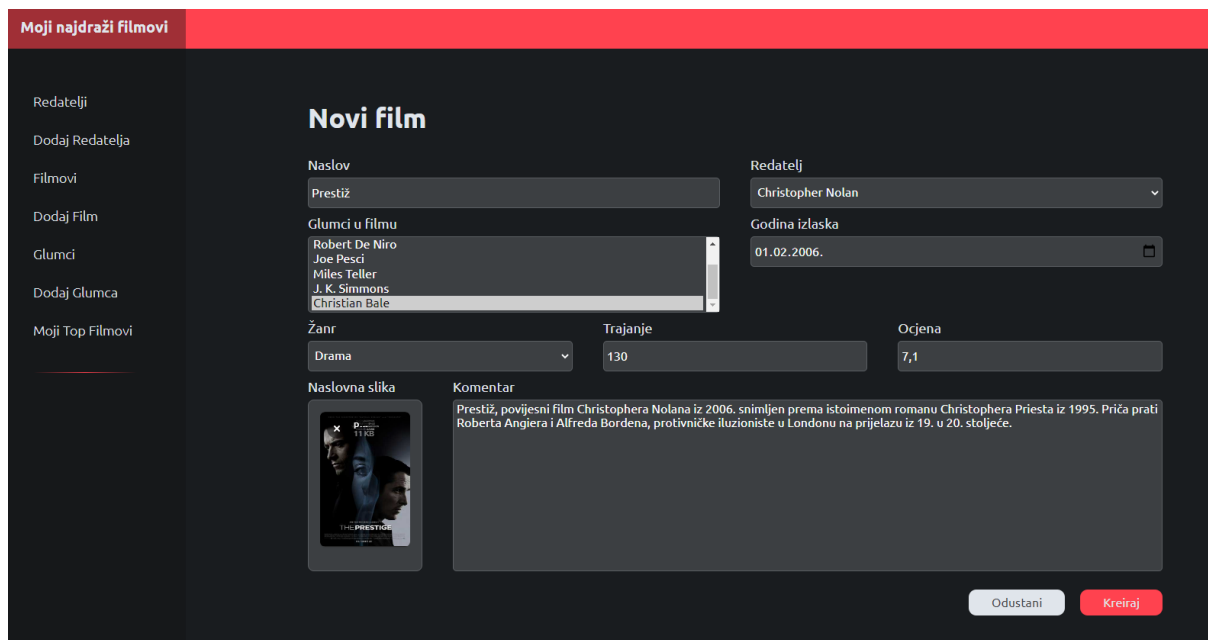


Izvor: izradio autor

Odabirom tipke 'Filmovi' unutar navigacijskog okvira, otvara se indeks stranica sa prikazom svih unesenih filmova. Iznad njih nalaze se tražilice po naslovu te traženje s obzirom na godinu izlaska. Klikom na sliku nekog filma, otvara se prikaz detalja filma kao što je prikazano na slici 50.

12.6.2. Dodavanje filma

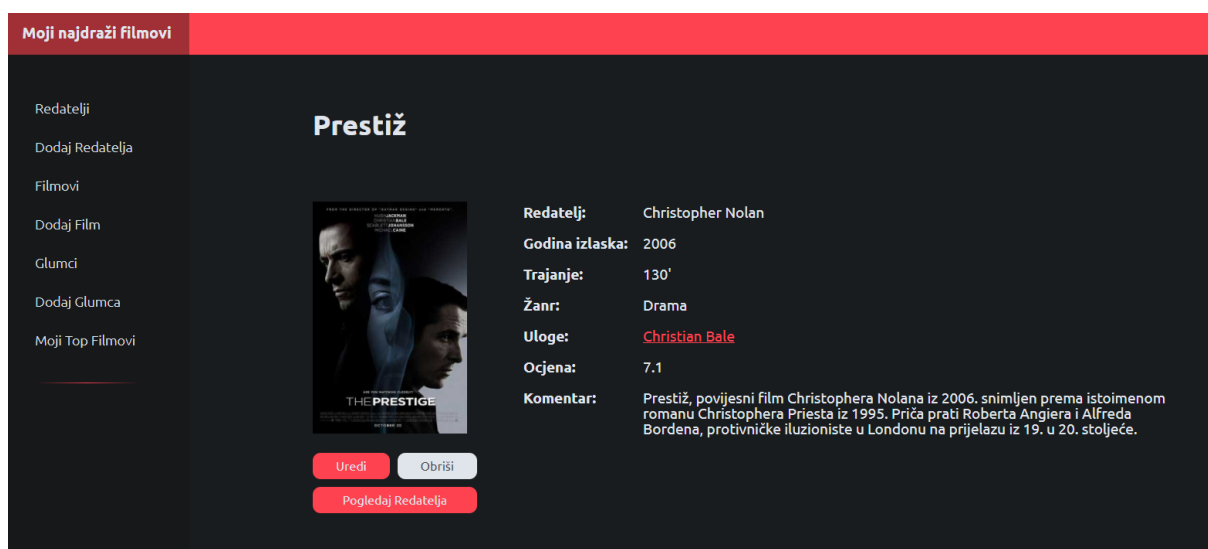
Slika 60: primjer prikaza kreiranja novog filma



Izvor: izradio autor

Nakon unosa podataka i ućivanja slike filma, klikom na tipku 'Kreiraj', kreira se novi film te se korisnika preusmjerava na stranicu sa detaljima novounesenog filma. Slika 61.:

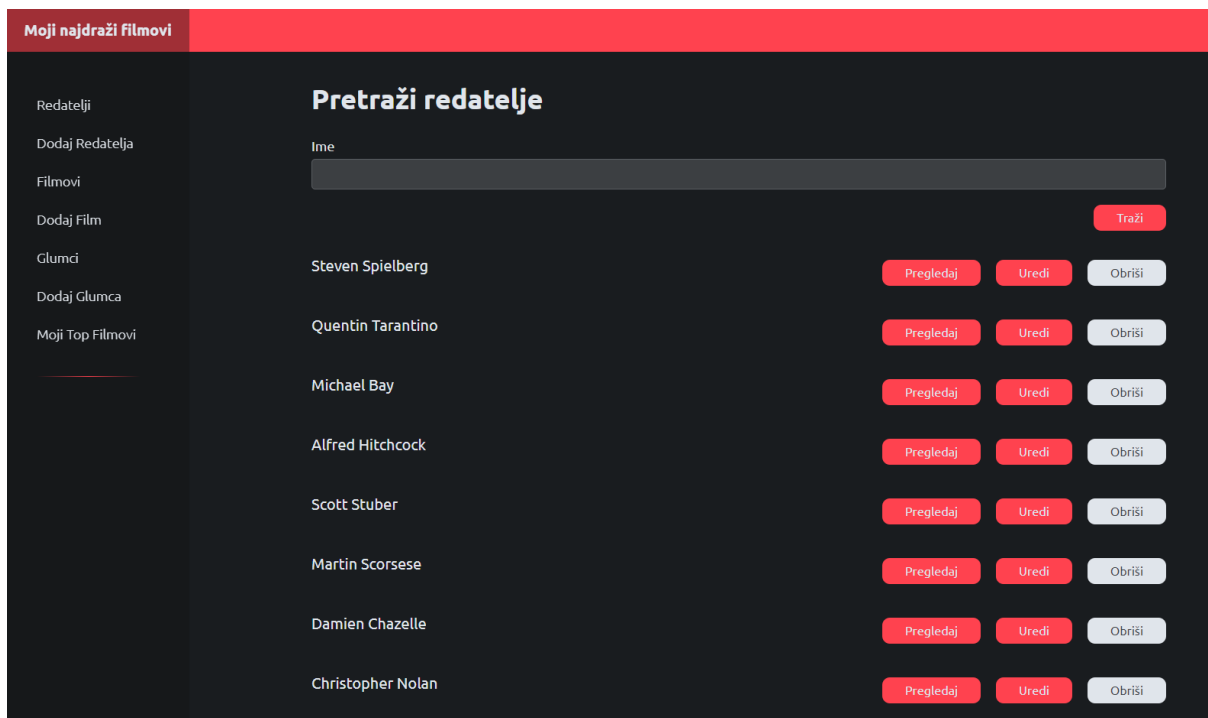
Slika 61: prikaz stranice sa detaljima novounesenog filma



Izvor: izradio autor

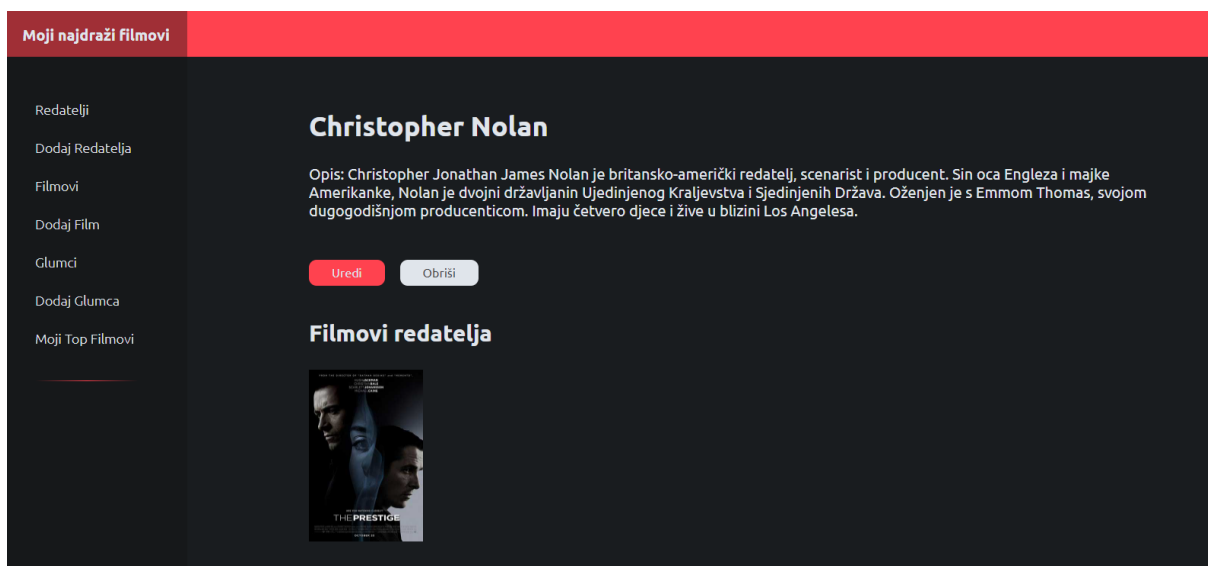
Nakon novog unesenog redatelja, glumca i filma, stranice sa unesenim entitetima izgledaju ovako: (Slike 62, 63 i 64)

Slika 62: prikaz novounesenog redatelja pri dnu indeks stranice redatelja



Izvor: izradio autor

Slika 63: detalji novounesenog redatelja (vidljivo je pojavljivanje novounesenog filma redatelja)

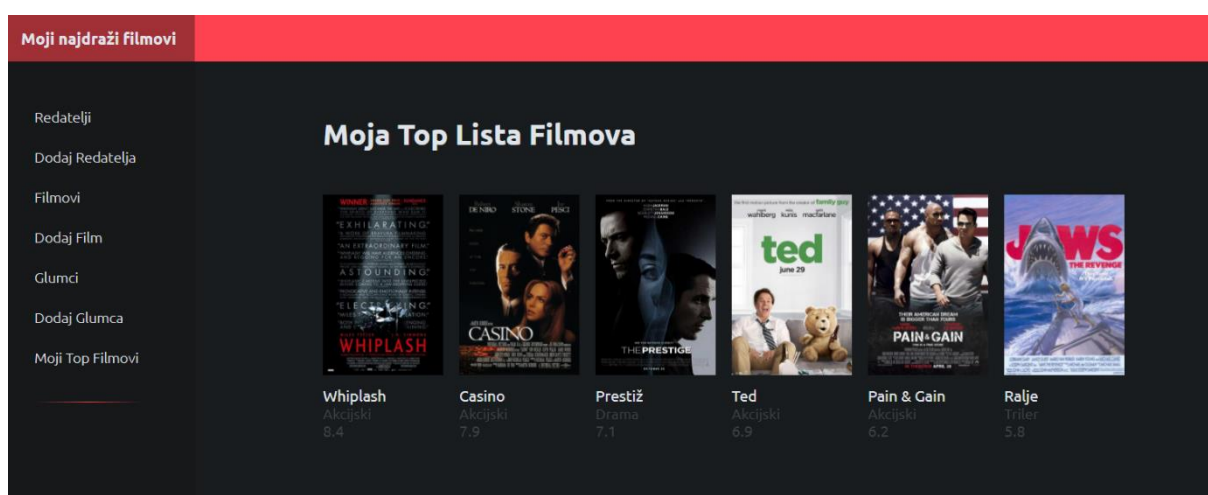


Izvor: izradio autor

12.7. Moji Top Filmovi

Nakon kreiranja novog redatelja Christophera Nolana, glumca Christiana Balea i filma Prestiž, odabirom tipke 'Moji Top Filmovi' prikazani su svi filmovi (uz prethodno unešen) poredani po ocjeni. Slika 64.:

Slika 64: Indeks stranica 'Moji Top Filmovi'



Izvor: izradio autor

Klikom na sliku svakog od prikazanih filmova, korisnika se preusmjerava na stranicu detalja prikazanim kao na slici 50.

13. ZAKLJUČAK

Za izradnju ovog projekta utrošeno je nekoliko tjedana predanog rada. Nakon svladavanja osnovnog principa funkcioniranja servera i putanja, izrada je tekla lakše i brže. Najviše vremenski zahtjevna faza bila je serverska strana; postavljanje svih potrebnih putanja; indeks stranice, pregled, brisanje, uređivanje za svaki model, a najmanje klijentski dio – CSS.

Glavni problem prilikom izrade aplikacije bili su mogućnost unošenja više glumaca za jedan film, a riješen je na način da se u modelu filma za objekt 'actors' omogući polje glumaca, koje zapravo i omogućava spremanje polja, dakle više glumaca za jedan film; te spremanje slika umjesto na Heroku server (iz razloga što se nakon resetiranja servera slike više ne pokazuju na klijentskoj strani) na bazu podataka.

Kao daljnje poboljšanje projekta predlaže se izrada responzivnosti za druge uređaje, poboljšati korisničko iskustvo, dodati registraciju i prijavljivanje za više korisnika.

Aplikacija je testirana na pregledniku Google Chrome unutar Windows 10 operativnog sustava.

Autor se nada rastu u popularnosti i korištenju tehnologija Node.js, Express.js i MongoDB i JavaScripta općenito kao i njihovom korištenju na budućem poslu.

14. LITERATURA

1. CSS (2020), <https://hr.wikipedia.org/wiki/CSS>
2. D. Mihovilović (2014), Hello node.js, <http://blog.king-ict.hr/hello-nodejs> (21.7.2014.)
3. E. Veledar (2019), MVC Architecture, <https://medium.com/@emirveledar/mvc-architecture-819c5918f160>
4. Framework (2020), <http://www.tematikawebstudio.com/sta-je-framework.php>
5. GitHub (2020), <https://en.wikipedia.org/wiki/GitHub>
6. M. Rouse (2020), vanilla, <https://whatis.techtarget.com/definition/vanilla>
7. MongoDB (2020), <https://en.wikipedia.org/wiki/MongoDB>
8. Početak rada s Express.js (2020), hr.admininfo.info/empezando-trabajar-con-express
9. Software Library (2016), <https://www.techopedia.com/definition/3828/software-library>
10. Software stack (2020), <https://www.pcmag.com/encyclopedia/term/software-stack>
11. Uvod u HTML (2020),
<https://tesla.carnet.hr/mod/book/view.php?id=5430&chapterid=885>
12. What Does it Mean by Full-Stack Development? (2020),
<https://medium.com/@sagarajkt/what-does-it-mean-by-full-stack-development-d07b3f5aa5c3> (2.1.2020.)
13. What is EJS? (2020), <https://ejs.co/>
14. What is npm? (2020), https://www.w3schools.com/whatis/whatis_npm.asp

15. POPIS SLIKA

Slika 1: inicijalizacija projekta	5
Slika 2: Instalacija Express.js-a te EJS-a.....	5
Slika 3: package.json datoteka cijelog projekta.....	6
Slika 4: definiranje varijable za pokretanje servera.....	7
Slika 5: pokretanje node.js servera	8
Slika 6: MVC arhitektura	9
Slika 7: MVC model primijenjen na projektu	11
Slika 8: Uključivanje routes-a u server.js datoteku	12
Slika 9: korištenje use naredbe za korištenje uvezenih routes-a	12
Slika 10: naredba kojom se postavlja port servera	13
Slika 11: deklariranje korištenja express.js frameworka i ostalih prethodno instaliranih dependencies-a.....	13
Slika 12: set i use naredbe za korištenje i postavljanje prethodno instaliranih dependencies-a	13
Slika 13: Mongoose JSON model glumaca.....	14
Slika 14: Mongoose JSON model redatelja.....	15
Slika 15: Mongoose JSON model filmova.....	15
Slika 16: primjer "exportanja" modela redatelja	16
Slika 17: spajanje na mongoose lokalnu bazu podataka.....	16
Slika 18: prikaz sadržaja .env datoteke	17
Slika 19: uspješno spajanje na lokalnu MongoDB bazu podataka.....	17
Slika 20: layout.ejs datoteka	18
Slika 21: uključenje header.ejs datoteke u layout.ejs datoteci	19
Slika 22: uključenje body-ja unutar layout.ejs datoteke.....	19
Slika 23: struktura layout.ejs i header.ejs datoteka i mapa.....	19
Slika 24: sadržaj header.ejs datoteke	20
Slika 25: korištenje Express.js-a te spremanje Router() varijable uz uvoženje potrebnih modela	21
Slika 26: Putanja index stranice glumaca	22
Slika 27: render metoda za prikaz index EJS stranice svih glumaca.....	23

Slika 28: deklariranje i inicijalizacija varijable actors koristeći mongoDB metodu find()	23
Slika 29: if statement za prikaz onih glumaca čiji se naziv pretražuje.....	24
Slika 30: index.ejs glumaca	24
Slika 31: putanja za prikaz stranice za unos novoga glumca	25
Slika 32: new.ejs za kreiranje glumca	25
Slika 33: _form_fields.ejs datoteka - forma za unos novoga glumca.....	26
Slika 34: putanja za spremanje unesenih podataka	27
Slika 35: saveCover funkcija za spremanje slika na bazu podataka.....	27
Slika 36: putanja za prikaz svakog određenog glumca	28
Slika 37: show.ejs datoteka za prikaz glumaca	29
Slika 38: putanja za generiranje stranice za uređivanje glumca	30
Slika 39: .ejs datoteka za uređivanje glumca.....	30
Slika 40: putanja za brisanje glumca	31
Slika 41: kreiranje clustera unutar MongoDB Atlasa.....	32
Slika 42: kreirani cluster.....	33
Slika 43: dostupne opcije spajanja clustera	34
Slika 44: spajanje clustera na aplikaciju.....	34
Slika 45: kreiranje aplikacije na Heroku-u	35
Slika 46: spajanje baze podataka na kreiranu aplikaciju na Heroku-u	35
Slika 47: kreiranje Github repozitorija	37
Slika 48: naredbe za "pushanje" kôda na Github repozitorij.....	38
Slika 49: prikaz indeks stranice aplikacije	40
Slika 50: stranica prikaza pojedinog filma	41
Slika 51: indeks stranica svih redatelja.....	42
Slika 52: primjer prikaza pregleda redatelja.....	43
Slika 53: dodavanje novog redatelja.....	44
Slika 54: kreirani redatelj stranica detalja	44
Slika 55: indeks stranica glumaca	45
Slika 56: pregled detalja glumca	46
Slika 57: prikaz dodavanja novog glumca.....	46
Slika 58: prikaz novounesenog glumca na indeks stranici glumaca	47
Slika 59: indeks stranica svih filmova.....	48

Slika 60: primjer prikaza kreiranja novog filma.....	49
Slika 61: prikaz stranice sa detaljima novounesenog filma.....	49
Slika 62: prikaz novounesenog redatelja pri dnu indeks stranice redatelja.....	50
Slika 63: detalji novounesenog redatelja (vidljivo je pojavljivanje novounesenog filma redatelja)	50
Slika 64: Indeks stranica 'Moji Top Filmovi'	51