# Razvoj Web aplikacije u programskom jeziku Node.js

**Naglić, Kristian**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Kristian Naglić**

# Razvoj Web aplikacije u programskom jeziku Node.js

**ZAVRŠNI RAD**

**Varaždin, 2020.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Kristian Naglić**

**Matični broj: 44837/16–R**

**Studij: Informacijski sustavi**

# Razvoj Web aplikacije u programskom jeziku Node.js

**ZAVRŠNI RAD**

**Mentor/Mentorica:**

Matija Kaniški, mag. inf.

**Varaždin, srpanj 2020.**

*Kristian Naglić*

**Izjava o izvornosti**

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi*

**Sažetak**

Cilj završnog rada je obraditi proces izrade Web aplikacija pomoću Node.js programskog jezika. U prvom dijelu ovog rada opisati će metode i tehnike koje su korištene što uključuje HTML, CSS, JavaScript, MySQL, Github, Node.js i njegove dodatke. Posebno će se spomenuti i opisati dodatni Node.js paketi koji su ekstenzivno korišteni a to su EJS, Express i Imdb-api. Dalje će se opisati generalna struktura Web aplikacija i glavne značajke Node.js programskog jezika te njegove osnove. Biti će prikazano, zajedno sa primjerima, prije svega postavljanje radnog okruženja zatim rad i neke operacije sa različitim tipovima podataka, primjer stvaranja poslužitelja te koncepti izvezivanja modula i korištenja *middleware* blokova. Opisati će se što znači MVC (Model-view-controller) i kako to iskoristiti za postavljanje dobre strukture pri izradi Web aplikacija. Zatim će se opisati povezivanje sa MySQL bazom podataka te rad sa upitima, dinamično generiranje sadržaja te rad sa podacima iz obrazaca, što uključuje rad sa samim podacima u svrhu njihove validacije, slanje povratnih poruka korisniku i rad sa multipart formama. Pred kraj glavnog poglavlja o Node.js-u će se opisati još i pronalaženje grešaka što uključuje i njihovo obrađivanje. Na kraju rada prikazati će se razlike i sličnosti Node.js, PHP i Python programskih jezika te opis jednostavne integracije sa MongoDB, Angular.js i Express.js dodacima kako bi se izradila aplikacija na MEAN platformi. U poglavlju o MEAN platformi ponajviše će se fokusirati na vezu između komponenti i njihovu međusobnu interakciju. Kroz rad će se koristiti primjeri koda iz aplikacije koja je izrađena u svrhu ovog rada te će se njezina struktura uz korištenu bazu podataka, uloge korisnika i funkcionalnosti na kraju rada i opisati.

**Ključne riječi:** Node.js; JavaScript; Express; MySQL; Web aplikacija; paket; poslužitelj;

# Sadržaj

# 1. Uvod

U današnje vrijeme, sve je više tehnologija i JavaScript dodataka za izradu brzih i funkcionalnih Web aplikacija poput React, Angular, Express, Vue i ostalih.

Bez obzira na sve, svaka od tih tehnologija je orijentirana na rješavanje određenih problema i olakšavanje same izrade Web aplikacija, bilo to na način da se koristi ljepša i čišća sintaksa ili da se noćna mora zvana 'dizajniranje korisničkog sučelja' učini što lakšim. No ono što odvaja programske jezike kao PHP i Node.js od ostalih je orijentiranost na poslužiteljsku stranu. To u praksi znači da se mogu kombinirati Node.js, JavaScript i dodaci kao Angular ili Express da bi se olakšala izrada Web aplikacije iz razloga što su takvi dodaci orijentirani na klijentsku stranu. Ti dodaci također rješavaju neke od problema kod formatiranja izgleda sučelja kako se to ne bi moralo rješavati ručno i, u slučaju Express-a, poboljšavaju preglednost samog projekta. U ovom radu će se koristiti samo Node.js, Express i nekoliko manjih paketa kako bi se što bolje prikazala sintaksa Node.js programskog jezika kao i njegove mogućnosti.

Cilj ovog rada je apstraktno prikazati potpunu izradu Web aplikacije sa Node.js programskim jezikom ali i potkrijepiti praktičnim primjerom Node.js aplikacije koja služi za praćenje i vođenje vlastite liste filmova i serija koje se gledaju ili planiraju gledati kao i pretraživanje filmova ili serija, promjena podataka o filmovima i serijama u listi, sortiranje, pregled statistike i detalja pojedinih filmova ili serija. Svi paketi koji su korišteni u izradi ovog projekta biti će opisani u nastavku rada.

Prije toga, definirati će se i opisati tehnologije korištene u izradi ovog rada, njihove sintakse i uporabe, te nakon toga će se definirati Web aplikacija i njezine značajke kako bi se mogao dalje opisati Node.js i objasniti proces izrade Web aplikacije u narednim poglavljima.

# 2. Metode i tehnike rada

U ovom poglavlju će se opisati korištene tehnologije u izradi web aplikacije. Biti će navedene najvažnije tehnologije, dok će ostatak manjih paketa biti ukratko opisan u zasebnom poglavlju.

## 2.1. HTML

Hypertext Markup Language (HTML) predstavlja *markup* jezik koji se koristi kako bi se naznačila struktura neke web stranice i značenje pojedinih dijelova od kojih se ona sastoji. HTML se sastoji od elemenata, a svaki od tih elemenata se sastoji od oznaka koje taj element otvaraju i zatvaraju te sadržaja ukoliko se radi o primjerice tekstualnim elementima [1]. Primjerice, element za paragraf bi izgledao ovako:

```
<p>Ovo je jedan paragraph</p>
```

Kod nekih elemenata nije potrebno koristiti oznake za zatvaranje, primjerice kod elementa “img”. Svakom elementu se mogu dodati atributi. Neki atributi vezani su za određene elemente poput atributa „src“ koji je vezan za „img“ element i predstavlja putanju do slike koja se prikazuje ili „href“ atributa koji je vezan za „a“ element i predstavlja putanju na koji treba voditi link. Neki atributi se mogu koristiti na gotovo svim elementima, a to su primjerice „id“ i „class“ atributi. Primjer atributa bi izgledao ovako:

```
<a href=”/” class=”link”>Jedan link</a>
```

Naravno, pojedine elemente je moguće postavljati unutar drugih što znači i da je element koji obuhvaća drugi element ujedno i njegov roditelj. Dva glavna elementa od kojih se sastoji svaki HTML dokument su elementi „body“ i „head“. „body“ element sadržava sve elemente koji se prikazuju korisniku, dok „head“ element sadržava sve što je nevidljivo korisniku, osim naslova stranice. Struktura glave HTML-a u aplikaciji projekta izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
     <link rel="stylesheet" href="/css/main.css">
     <link rel="stylesheet" href="/css/nav.css">
     <link rel="stylesheet" href="/css/media.css">
```

```
    <title>Naslov stranice</title>
</head>
```

Na početku dokumenta zadan je „DOCTYPE“ element, koji se prije koristio kako bi se naznačila pravila HTML dokumenta, no u novijim verzijama se jednostavno stavlja kako bi stranica radila te je ostatak starijih verzija [1]. „lang“ elementom se naznačava jezik HTML dokumenta. Unutar „head“ elementa navode se importi za CSS datoteke koje se koriste, naslov stranice i meta podaci.

## 2.2. CSS

Cascading Style Sheets (CSS) je jezik koji se koristi za postavljanje pravila prikazivanja pojedinih *markup* dokumenata poput HTML-a i XML-a. CSS se sastoji od pravila koja se definiraju unutar .css datoteka i uključuju u HTML ili XML dokumente u kojima su i potrebna [2]. Pravila je moguće zadavati na pojedine elemente poput elemenata „p“, „div“, „body“ i slično, te bi u tom slučaju bila zadana na globalnoj razini za svaki taj element unutar *markup* dokumenta za koji ta pravila i vrijede. Pravila je moguće postavljati i na pojedine klase navođenjem točke uz ime klase, te za pojedine identifikatore navođenjem znaka „#“ uz ime identifikatora nekog elementa. Klase, identifikatori ili elementi mogu se navoditi jedan iza drugog odvojeni zarezom kako bi se stiliziralo više elemenata odjednom, te navoditi jedan iza drugog bez zareza kako bi se kretalo kroz „stablo“ HTML-a i stiliziralo neki pojedini element. Na svako pravilo je moguće postavljati događaje poput „hover“ koji označavaju kada se to pravilo primjenjuje elementu. Oznakom „*” se označava stiliziranje cijelog dokumenta. Primjer nekoliko postavljenih pravila iz aplikacije:

```
a, a:hover {
  text-decoration: none;
  color: inherit;
  font-size: 1.4rem;
}
.wrapper .sidebar {
  grid-area: c;
}
```

U projektu je korišten fleksibilan dizajn i grid koji su dio CSS stiliziranja. Fleksibilan dizajn odnosi se na skalabilnost elemenata i veličine fonta aplikacije. To se uz CSS može postići na više načina, no u ovom radu je korištena kombinacija „flex“ atributa i grida. Također, radi postizanja fleksibilnog dizajna, korištena je oznaka veličina „rem“ umjesto „px“ što označava da je svaki element relativan veličini fonta korijenskog HTML elementa. Postavljanjem pravila „display: flex“ naznačuje korištenje fleksibilnog prikaza elemenata što

znači da se elementi pozicioniraju relativno prethodnom elementu koji također ima postavljeno ovo pravilo, odnosno sva djeca nekog elementa koji je roditelj će imati to svojstvo. Dodatno je moguće postavljati omotavanje elemenata, poravnanje, razmak između elemenata i slično. Drugo svojstvo je grid. Grid označava rešetku koju je moguće precizno definirati uz svojstvo „grid-template“. Za rešetku se mogu postavljati veličine redaka, stupaca, broj stupaca i redaka, pozicije pojedinih elemenata u rešetci i druga svojstva. Primjer definiranja rešetke:

```
display: grid;
grid-template-columns: 1fr;
grid-template-rows: 6rem auto;
grid-template-areas:
  "a b"
  "c d";
```

U ovom primjeru definirana je rešetka sa dva stupca i dva retka od kojih svaki ima svoju slovčanu oznaku koja se može dodijeliti nekom elementu djetetu putem svojstva „grid-area“. Svaki stupac je veličine jedne frakcije odnosno dijela stranice. Prvi redak je veličine 6rem dok je drugi redak postavljen na „auto“.

## 2.3. Javascript

Prvi i osnovni programski jezik koji je potpora ostalim jezicima opisanim u ovom radu je JavaScript. JavaScript je skriptni pravovremeni (eng. *just-in-time)* kompilacijski programski jezik koji se većinom koristi u sklopu web preglednika [3]. Iako se koristi većinom u web preglednicima, nije na njih ograničen te je jedno od okruženja izvan web preglednika koje koristi JavaScript upravo i Node.js.

JavaScript je najpoznatija implementacija ECMAScript specifikacije koja je trenutno na verziji ES10 (ECMAScript 2019) [3]. Trenutno, svaki web preglednik podržava barem ES6 specifikaciju, što znači da se JavaScript nalazi u gotovo svakom elektroničkom uređaju, bilo da se radi o pametnom telefonu, televizoru, računalu ili čak i štednjaku.

JavaScript se koristi za manipuliranje elemenata web sadržaja zvanim Document Object Model (DOM) [4]. DOM je API za preglednike koji predstavlja HTML sadržaj u obliku shematskog stabla . Svaki dio tog stabla se naziva čvor (eng. *Node*), te je svakom tom dijelu moguće zasebno pristupiti. Korištenjem JavaScripta, moguće je svakom elementu HTML-a dodati tvz. osluškivače događaja (eng. *Event listener*) koji će pokrenuti funkciju u slučaju nekog događaja. Elementi DOM-a se mogu dohvaćati putem postavljenog ID-a, imena ili drugog atributa te se njihova svojstva poput stila prikaza ili vrijednosti mogu mijenjati.

Sintaksa JavaScripta je poprilično slična Javi uz stvaranje objekata i funkcija sa zakrivljenim zagradama. Inkorporira većinu ostalih značajki drugih jezika poput if-else i do-while blokova. No za razliku od Jave, varijable se mogu definirati sintaksama *let, var* i *const*. *Var* je standardna sintaksa za definiranje varijabli u JavaScriptu. Ona ima lokalan obuhvat i dostupna je bilo gdje unutar funkcije u kojoj je definirana. *Const* i *let* su novije sintakse koje su uvedene uz ES6 (ECMAScript 6) [5] te one omogućavaju deklariranje varijabli unutar blokova, što znači da im se može pristupiti samo unutar blokova u kojima su deklarirane.

Uz obične funkcije, postoje i tvz. *Arrow* funkcije, odnosno funkcije koje su anonimne, što znači da se moraju definirati kao varijable [3]. Te funkcije se razlikuju od običnih po još jednom svojstvu, njihov obuhvat se odnosi na sam obuhvat funkcije u kojoj su definirane, dok normalne funkcije imaju definiran svoj vlastiti obuhvat. Time su vrlo korisne kod objektno orijentiranog programiranje te su većinom korištene u izradi praktičnog dijela ovog rada.

U izradi aplikacije korišteno je nekoliko JavaScript osluškivača događaja kako bi se postigla fluidnost nekih elemenata aplikacije (npr. Fluidno otvaranje menija). Također je korišten i na strani poslužitelja u obliku Node.js okruženja koji će se sljedeći opisati, kao i njegovog dodatnog okvira, Express.

## 2.4. Node.js

Node.js je JavaScript okvir, dakle može se reći da je Node.js zapravo drugačija inačica JavaScripta koja omogućava da se JavaScript kod pokreće na poslužitelju, odnosno bilo kojem računalu izvan web preglednika [6]. Node.js je asinkron i većinom baziran na događajima (eng. *Events*), no postoje i metode koje imaju svoje sinkrone verzije, primjerice *File System* biblioteka, ali koje ne bi trebali koristiti ukoliko radimo na objektno orijentiranom asinkronom projektu.

Node.js je baziran na Google-ovom Chrome V8 JavaScript *engine-u*. V8 nije verzija već samo ime *engine-a* koji je originalno namijenjen za korištenje u browseru, no kasnije je prešao na stranu otvorenog koda i na taj način postao podloga za Node.js [6].

Glavna karakteristika Node.js-a je to što je ne-blokirajući, što znači da svi događaji definirani u Node.js-u dodaju u tvz. skup događaja (eng. *event pool*) te se registriraju i pokreću samo onda kada je to potrebno bez blokiranja daljnjeg izvršavanja kod. Može se reći da su ti događaji glavni elementi Node.js-a kao programskog jezika te se rad s Node.js-om temelji na njima. Ti događaji mogu imati povratne funkcije (eng. *callbacks*) koje se pozivaju nakon izvršavanja događaja [7]. Takav način rada je vrlo pogodan za izradu pravovremenih brzih i lakih aplikacija pošto izvršavanje manje bitnog koda ne bi blokiralo izvršavanje koda za npr.

prikaz same aplikacije. Nije garantirano da će svi događaji koji se pozovu u stvarnosti biti asinkroni, npr. može se definirati događaj koji u sebi sadrži poziv na sinkronu funkciju koja bi blokirala daljnje izvođenje koda.

Ranije je spomenuto da je Node.js drugačija inačica JavaScripta, upravo zato jer Node.js dodaje nove mogućnosti Javascript-u koje nisu moguće unutar Web preglednika, primjerice, pristupanje i modificiranje lokalnih datoteka. Također, bez obzira da li je aplikacija izrađena u Node.js-u, na klijentskoj strani se i dalje pokreće samo Javascript, dok je Node.js limitiran na pokretanje na poslužiteljskoj strani.

Node.js nema posebnu sintaksu s obzirom da je temeljen na JavaScript-u no definira svoje metode i globalne objekte koji se koriste za rukovanje poslužiteljem.

Još jedna jaka strana Node.js-a je mogućnost proširivanja pomoću paketa. Jednostavno rečeno, paketi su dodaci Node.js okviru u obliku JavaScript biblioteka koje se mogu uključiti u izvedbu bilo koje skripte. Ti paketi se mogu dodavati u Node.js okruženje putem Node Package Manager-a (npm). Node Package Manager je registar paketa koji su dostupni za Node.js. Paketi se mogu instalirati putem konzole bilo unutar aplikacija kao Windows Studio Code, ili izvan putem sučelja naredbenog retka [8]. Svi instalirani paketi biti će izlistani u package.json datoteci projekta zajedno sa svojom verzijom, imenom, opisom i drugim karakteristikama.

U nastavku će se opisati Node.js paketi korišteni prilikom izrade aplikacije.

### 2.4.1. Express

Express.js, ili samo Express je lagan okvir koji razrješava problem pisanja HTTP zahtjeva, što uključuje rukovanja sa zaglavljima (eng. *header*) i podacima iz tijela (eng. *body*) HTML dokumenata uz brojna druga poboljšanja [9].

Neke od prednosti Express-a [9]:

- Vrlo lako se postavlja i koristi
- Pojednostavljuje rukovanje zahtjevima uvođenjem tvz. *middleware-a*, „funkcija“ koje se pokreću tijekom zahtjeva
- Lako se integrira sa *templating engine-ima* od kojih će se jedan koristiti i u ovom projektu
- Daje mogućnost lakog serviranja statičnih dokumenata, poput slika
- Lako se integrira sa poznatim bazama podataka kao što su MongoDb ili MySQL

Naravno to je sve moguće i bez Express-a, no kako bi kod ostao pregledan i da se olakša izrada same aplikacije, preporučljivo je koristiti Express. Express je iz istog razloga izrazito korišten u izradi praktičnog dijela ovog rada.

### 2.4.2. EJS

Embedded JavaScript (EJS) je jedan od dostupnih *templating engine-a* koji se mogu koristiti kako bi se lakše i dinamično generirao HTML sadržaj. EJS uvodi sintaksu „<% %>" kojom omogućuje da se unutar HTML datoteka piše JavaScript i koriste varijable koje se proslijede putem Node.js-a. Korištenjem „<%=" pripisuje se vrijednost neke varijable na to polje, primjerice na vrijednost input polja ili polja klase nekog HTML elementa [10].

EJS podržava i zadavanje vlastitih sintaksi kod prosljeđivanja HTML sadržaja [10].

Postoje i drugi *templating engine-i*, primjerice, Pug i Handlebars, no EJS je odabran za korištenje u projektu iz razloga što se koristi običan HTML + JavaScript, dok se kod primjerice Pug-a, koristi prilagođen HTML jezik i sintaksa. EJS je vrlo lako za razumjeti te je pogodan za kretanje kroz petlju i ispisivanje dinamičnog broja sadržaja.

### 2.4.3. Imdb-api

Imdb-api je paket koji uvodi metode za korištenje OMDb API-a radi dohvaćanja podataka sa IMDb-a [11]. Paket je ekstenzivno korišten kroz projekt te se većina podataka oslanja na ovaj API. Paket se koristi na sljedeći način:

```
const imdb = require('imdb-api');
const imdbHandler = new imdb.Client({apiKey:
`${process.env.IMDB_APIKEY}`});


module.exports = imdbHandler;
```

Dakle, paket ima mogućnost direktnog uključivanja i korištenja metoda na objektu „imdb", no u svrhu lakšeg korištenja, stvara se novi klijent zajedno sa opcijama koje će se koristiti te se taj klijent izvozi za daljnje korištenje kroz projekt. U ovom slučaju, zadan je vlastiti API ključ koji je također moguće zatražiti za korištenje putem OMDb API stranice. Metode koje se sada mogu koristiti na izvezenom objektu su get(), dohvaća specifičnu seriju ili film, i search(), pretražuje po zadanim parametrima. Funkcija get() koristi se na sljedeći način [11]:

```
imdb.get({ id: imdbId })
        .then((show) => {
            //show.ratings - show.description – show.title…
    });
```

U funkciji iznad, zadaje se imdbId te se pronalazi odgovarajući film koji sadrži svojstva kao ocjene, naziv, redatelje, glumce, kratak opis i slično. Funkcija također prima i argument „name“ umjesto „id“ kako bi se dohvatio film po nazivu. Druga funkcija je search() koja prima „name“ argument odnosno naziv filma ili serije koja se pretražuje [11]:

```
imdb.search({ 'name': 'Matrix', reqtype: 'movie' }, page)
            .then((shows) => { //shows je pronađena lista });
```

Funkcija prima i argument „reqtype“ koji može biti „movie“ ili „series“ pri čemu se onda prikazuju samo filmove ili serije. Također prima i argument „page“ koji označava stranicu koja se dohvaća. Rezultat je lista pronađenih filmova ili serija koja sadržava i svojstvo „totalresults“ koje sadržava broj ukupno pronađenih rezultata. Svi argumenti funkcije search() su opcionalni osim argumenta „name“.

### 2.4.4. Ostali Node.js paketi

Do sada su opisani paketi od veće važnosti koji se ekstenzivno koriste kroz projekt. U ovom će se poglavlju ukratko opisati ostali korišteni paketi koji su od manje važnosti ili su korišteni u specifičnim slučajevima. Njihovo korištenje biti će prikazano u narednim poglavljima.

Paketi:

- nodemon – Paket koji se uglavnom instalira na razvojnoj strani (--save-dev) i omogućuje da se svakim spremanjem .js datoteka, projekt automatski ponovo pokrene [12].
- body-parser – Paket koji nas razrješava ručnog izvlačenja podataka iz *body* elementa. Dodaje *body* atribut poslanom zahtjevu popunjen sa podacima forme koji se može koristiti kroz *middleware* [13].
- bcryptjs – Paket koji omogućava sigurno šifriranje lozinki [14].
- express-session i express-mysql-session – Omogućava stvaranje sesija uz zadane opcije. Sam paket sprema sesiju u memoriji, no kako bi se sesija spremala u bazu podataka, koristi se jedan od *store* paketa koje express-session podržava. Ovdje je korišten express-mysql-session radi korištenja MySQL baze podataka [15], [16].
- nodemailer – Omogućava slanje e-mailova preko raznih servisa poput Gmail-a ili SendGrid-a [17].
- moment – Paket za lako formatiranje i rukovanje sa DATE objektima [18].

- multer – S obzirom da body-parser ne podržava forme tipa *multipart*, korišten je multer koji omogućava obrađivanje podataka takvog tipa forme [19].
- connect-flash – Paket za prijenos poruka putem sesije. Jednostavno rečeno, omogućava slanje podataka između različitih zahtjeva. Izuzetno pogodno za korištenje kod slanja povratnih informacija korisniku [20].

## 2.5. MySQL

MySQL je najpopularniji sustav za upravljanje bazom podataka, otvorenog koda [21]. On se temelji na relacijama za razliku od primjerice MongoDb, koji se ne temelji na relacijama već poprima izgled objekata u JSON obliku, odnosno kaže se da je MongoDb NoSQL baza podataka.

Glavna razlika između SQL i NoSQL baza podataka je ta da NoSQL nema relacije. Umjesto tablica, NoSQL sadrži dokumente koji poprimaju izgled objekata sa atributima. Ukoliko se trebaju neki podaci povezati sa drugim dokumentom, ti podaci se pišu dva puta, odnosno u oba dokumenta [22]. S obzirom da spajanja dokumenata nema, NoSQL je izrazito brz i primarno je dizajniran za izradu velikih skalabilnih aplikacija koje spremaju terabajte podataka poput Facebook-a ili Google-a.

Ipak, za izradu aplikacije u sklopu ovog rada, koristiti će se relacijski temeljene SQL baze podataka iz razloga što se rukuje sa malim brojem podataka i što su relacijske baze pogodnije za vizualizaciju podataka.

Za slanje upita i komuniciranje sa bazom podataka korišten je Node.js paket mysql2 koji implementira funkcije za spajanje na bazu podataka jednokratno ili kreiranjem skupa veza (eng. *connection pool)* za više upita. Taj objekt za konekciju se može izvesti te se na njega mogu pozvati metode execute() i query() pomoću kojih šaljemo upite na bazu [23].

Baza podataka je napravljena u MySQL Workbench-u i servirana lokalno.

## 2.6. Github

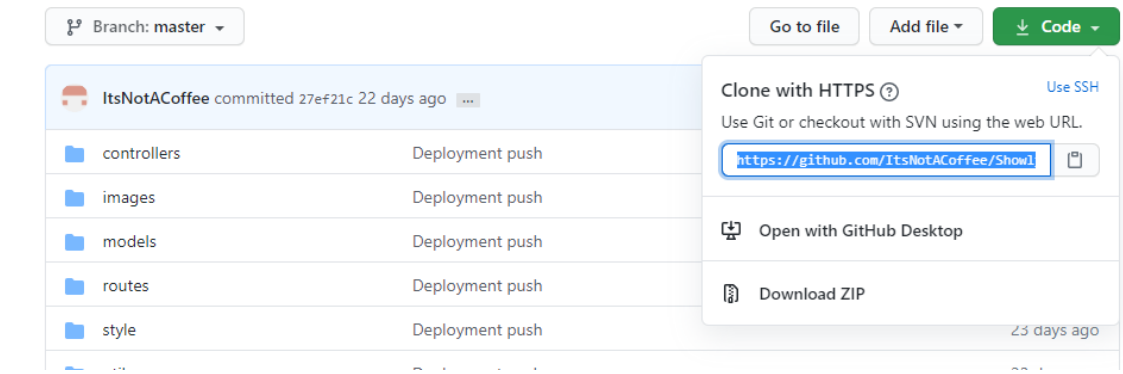
Github je stranica za praćenje koda, kontribucija pojedinih članova tima, dijeljenje projekata i jednostavno promjena koje se događaju nad nekim projektom. Github daje i mogućnost vraćanja starije verzije ili usporedbe koda starije i novije verzije projekta. Sve naredbe se pišu u sučelje naredbenog retka unutar glavnog direktorija projekta. Da bi se koristile naredbe Github-a, potrebno je instalirati Git preko službene poveznice: https://git-

scm.com/downloads . Nakon toga je moguće koristiti bilo koje sučelje naredbenog retka za pisanje naredbi.

Prije svega, za korištenje Github-a potrebno ga je i inicijalizirati za pojedini projekt naredbom „github init“. Naredba stvara „.gitignore“ datoteku unutar direktorija projekta unutar koje se dodaju datoteke ili direktoriji koji se ne žele stavljati na Github (senzitivni podaci poput lozinki, API ključevi, itd.). Repozitorije je moguće stvarati online na Github službenoj stranici gdje će prilikom stvaranja biti moguće odabrati da li se želi koristiti postojeći ili novi repozitorij. Ako se koristi postojeći repozitorij, on se može koristiti lokalno pokretanjem naredbe kojom se dodaje novi izvor [24]:

```
git remote add origin url-do-github-repozitorija
git remote -v
```

Naredba „remote -v“ verificira da li je veza uspostavljena. Poveznica do vlastitog repozitorija se može naći na početnoj stranici repozitorija, kao na slici ispod.




Slika 1: Github poveznica repozitorija (Izvor: vlastita izrada)

Nakon toga, dodavanje datoteka i promjena u kodu se jednostavno radi sa sljedećim naredbama:

```
git add .
git commit -m „moj prvi commit“
git push origin master
```

Naredba „add“ prima kao argument datoteke ili direktorije koji se dodaju u trenutne promjene, u ovom slučaju „.“ označava cijeli direktorij projekta. Naredbom „commit“ dodaje se nova promjena koja obuhvaća dodane datoteke i direktorije uz jednostavnu poruku. Na kraju, naredba „push“ postavlja sve promjene uz poruku na Github na zadanu granu (eng. *branch*), u ovom slučaju „master“. Naredbom „git checkout imegrane“ je moguća zamjena grane koja se koristi. Suprotno „push“ naredbi je „pull“ naredba koja dohvaća sve zadnje promjene kojih

trenutno u projektu nema [24]. Ukoliko se želi klonirati aplikacija ovog projekta, jednostavno se koristi naredba „clone“ u zadanom direktoriju:

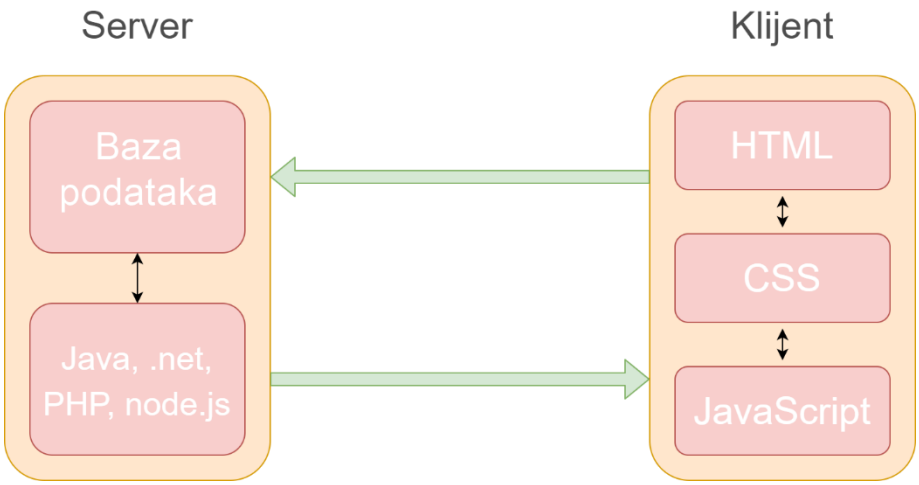
```
git clone https://github.com/ItsNotACoffee/Showly.git
```

# 3. Struktura Web aplikacije

Svaka od aplikacija koje se koriste na mobilnim uređajima, računalima ili drugim uređajima imaju svoju definiranu strukturu, pa tako i Web aplikacija. Web je jednostavno mreža računala koja komuniciraju jedni s drugima te izmjenjuju informacije. Web aplikacija je upravo to sučelje koje služi za komunikaciju jednog računala sa drugim, i koja naravno ima svoju svrhu koju obavlja.

Kada korisnik upiše određenu Web adresu u svoj preglednik, njegovo računalo šalje zahtjev preko HTTP ili HTTPS-a određenom poslužitelju koji obrađuje zahtjev i šalje nazad HTML, CSS i JS datoteke koje služe za oblikovanje samog odgovora poslanog sa poslužitelja na ekranu korisnika.

Iz toga se može zaključiti da se struktura Web aplikacije sastoji od dvije glavne komponente, klijentske i poslužiteljske strane.




Slika 2: Struktura Web aplikacije (Izvor: vlastita izrada)

Poslužiteljska strana, odnosno poslužitelj, je računalo koje poslužuje Web aplikaciju i čini ju dostupnim korisnicima preko određene adrese. Poslužitelj se može dalje podijeliti na bazu podataka i na aplikacijsku logiku, a aplikacijska logika se opet sastoji od programskih jezika koji rukuju podacima na poslužitelju. Ti programski jezici mogu biti primjerice PHP, JAVA, .NET, Ruby On Rails, Python i drugi. U slučaju izrade Node.js aplikacije, tu se zapravo

koristi JavaScript jezik i za klijentsku i za poslužiteljsku stranu što Node.js čini vrlo jednostavnim za korištenje.

Ako pogledamo klijentsku stranu, ona se sastoji od HTML, CSS i JS datoteka. HTML datoteke sadrže podatke koji su vidljivi korisniku na njegovom ekranu. No kako korisnik ne bi gledao samo tekst bez ikakvog formatiranja, tu se koristi CSS. CSS određuje pravila kako će koji element izgledati korisniku. CSS nije limitiran na korištenje statičnih pravila, već se mogu zadavati animacije i događaji na određene elemente. Posljednji u nizu je JavaScript. JavaScript je „mozak“ klijentske strane i pomoću njega se može upravljati HTML sadržajem. On omogućuje dohvaćanje elemenata po ID-u, nazivu, klasi, uređivanje klasi elemenata, stila, promjenu vrijednosti pa čak i dodavanje i brisanje cijelog HTML sadržaja.

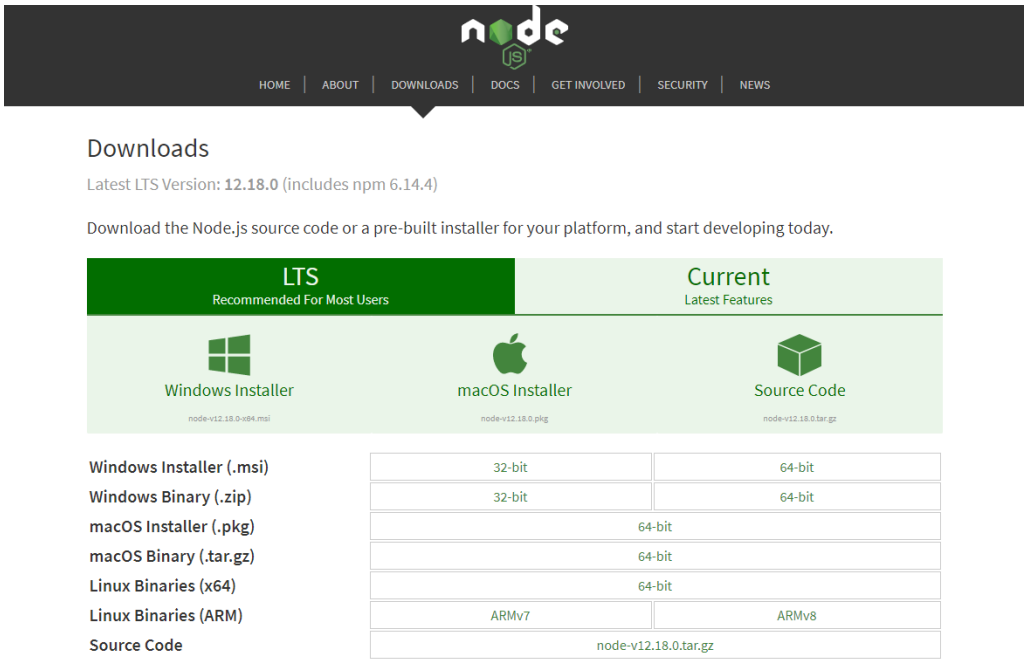
Web aplikacije općenito šalju zahtjev i ponovo učitavaju odgovor, odnosno datoteku koju im poslužitelj šalje. Da bi se to zaobišlo, koristi se AJAX (Asynchronous JavaScript and XML) koji omogućava primanje zahtjeva i dobivanje odgovora, odnosno novog sadržaja bez ponovnog učitavanja stranice na strani korisnika. AJAX je također dio klijentske strane i njegovim korištenjem Web aplikacije zapravo i dobivaju titulu pravih stolnih aplikacija koje naravno rade bez ponovnog učitavanja.

Na strani poslužitelja se mogu, kao u slučaju Node.js-a, koristiti mnogi dodaci da bi se rad sa podacima i zahtjevima olakšao, no što s kolačićima i sesijama, da li su oni dio poslužitelja ili klijenta? Ako se govori o kolačićima, oni su dio klijenta. Kolačići (eng. *Cookies*) predstavljaju informacije koje su spremljene lokalno na klijentskom računalu. Suprotno tome, sesije (eng. *Sessions)* se obično spremaju na strani poslužitelja i sadrže korisnikove podatke zajedno sa identifikatorom kolačića. Dakle, obje strane imaju nekakav oblik pohrane koji također spada pod elemente klijentske i poslužiteljske strane.

# 4. Izrada Node.js Web aplikacije

## 4.1. Osnove Node.js-a

Prije korištenja samog programskog jezika, potrebno ga je instalirati. To se može napraviti putem službene poveznice: https://nodejs.org/en/download/ . Skine se paket ovisno o operacijskom sustavu koje pokreće računalo na kojega se instalira Node.js i slijede se upute prilikom instalacije. Na poveznici se može uočiti da instalacija sadržava i npm koji će se koristiti u svrhu instalacije dodatnih paketa za Node.js.




Slika 3: Node.js stranica za skidanje instalacije (Izvor: Node.js, nodejs.org)

Nakon što je instalacija završena, Node.js aplikacije se pišu kao i obične JavaScript aplikacije. Ekstenzije datoteka su .js i mogu se pisati u bilo kojem programu za pisanje koda, u ovom slučaju, korišten je Visual Studio Code. Kao i kod JavaScript-a, varijable se definiraju sa *let, const* i *var.* Datoteke se mogu odvajati i pisati zasebno te uključiti u izvedbu koda u drugoj datoteci ako se koristi funkcija *require*, primjerice:

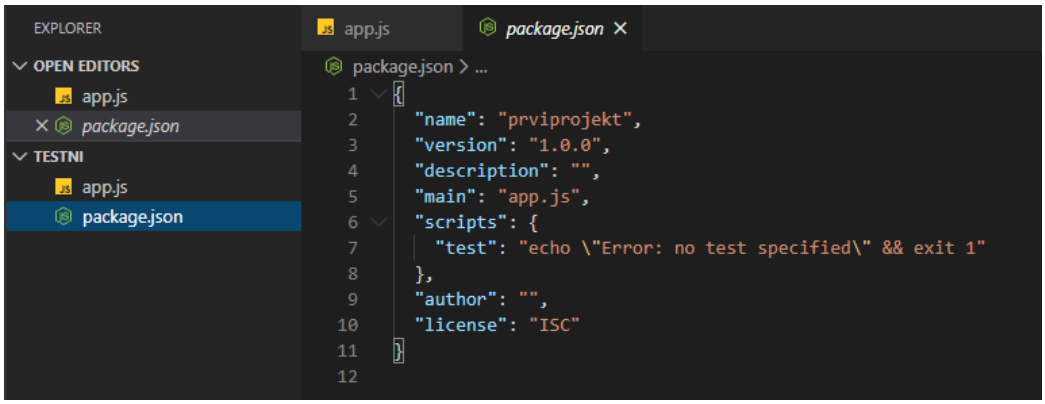
```
const fs = require('fs');
```

Funkcije se mogu definirati na dva načina, preko *function* i kao tzv. *arrow* funkcije. Razlika je objašnjena u prijašnjim poglavljima no dalje u projektu će se većinom koristiti noviji oblik funkcija.

### 4.1.1. Postavljanje razvojnog okruženja

Za demonstraciju ovog poglavlja korišten je program Visual Studio Code.

Na početku svakog Node.js projekta, potrebno je postaviti package.json datoteku. Ona se može postaviti ručno ili se može generirati putem naredbe „npm init“. Ukoliko se naredba pokrene unutar Terminal prozora, npm ispisuje moguće opcije koje se mogu postaviti za package.json. Na kraju, taj dokument bi izgledao ovako:




Slika 4: package.json (Izvor: vlastita izrada)

To je jednostavan dokument json formata koji sadrži osnovne podatke o projektu koji se naravno i dalje mogu mijenjati po želji.

Sljedeći korak je pisanje koda, no prije toga instalirati će se nekoliko dodatnih paketa kako bi odmah u početku pisanje koda bilo jednostavnije. Paketi se instaliraju, kao što je ranije spomenuto, putem npm odnosno Node Package Manager koji dolazi uz instalaciju Node.js-a.

Paketi se mogu instalirati putem naredbe „npm install –save imepaketa“. Ime paketa je točno ime koje je ispisano na online repozitoriju npm paketa, dostupnom na: https://www.npmjs.com/ .

Uz to, opcija –save je za spremanje paketa o kojima zavisi sam projekt, no ako se koristi opcija –save-dev, onda se naznačuje da paket koji se instalira, ovisi samo o razvojnom okruženju te nije potreban za pokretanje projekta [25].

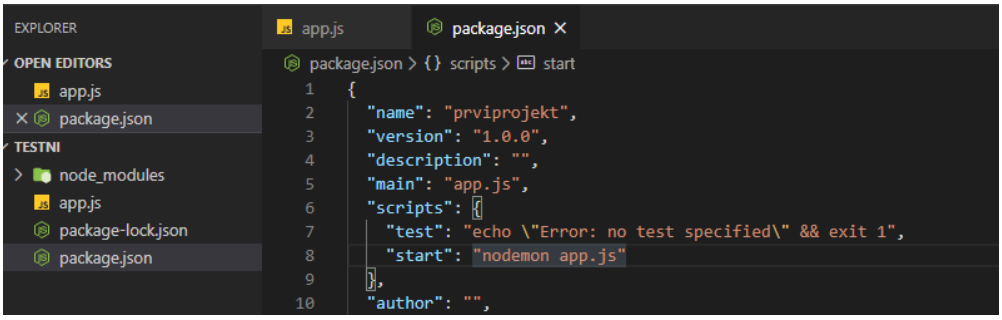
Moguće je instalirati i više paketa odjednom, jednostavno navođenjem imena paketa jedni za drugim. Ovdje će se instalirati četiri dodatna paketa, točnije nodemon, express, body-parser i EJS za početak:

```
PS C:\Users\Kristian\Desktop\testni> npm install --save nodemon express body-parser ejs

> ejs@3.1.3 postinstall C:\Users\Kristian\Desktop\testni\node_modules\ejs
> node --harmony ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)


> nodemon@2.0.4 postinstall C:\Users\Kristian\Desktop\testni\node_modules\nodemon
> node bin/postinstall || exit 0
```

Slika 5: Instaliranje npm paketa (Izvor: vlastita izrada)

Ono što se odmah može napraviti je postaviti nodemon automatsko pokretanje. Nodemon je jednostavan paket koji olakšava izradu Node.js aplikacije time da automatski

ponovo pokreće poslužitelj kada detektira da se dogodila promjena nad .js datotekama prilikom spremanja [12]. Unutar package.json postavlja se novo svojstvo u objektu „scripts“, postavlja se ime skripte kao „start“ i dodjeljuje mu se vrijednost „nodemon app.js“ gdje je app.js naziv primarne .js datoteke koja pokreće poslužitelj [26]:




Slika 6: Dodavanje nodemon automatskog pokretanja (Izvor: vlastita izrada)

Sve skripte koje se navedu pod „scripts“ svojstvo u package.json, se mogu pokrenuti putem komandne linije sa naredbom „npm run-script imeskripte“. Također, pod svojstvom „main“ se vidi da je tu naveden naziv glavne .js datoteke u korijenskom (eng. root) direktoriju projekta. To je naziv datoteke koja se pokreće kada se upiše naredba „npm start“. Tu datoteku je potrebno ručno stvoriti i u nju se piše glavni Node.js kod koji poziva sve ostale funkcije kao i samo pokretanje osluškivanja na zahtjeve.

### 4.1.2. Rukovanje s tipovima podataka

S obzirom da se Node.js bazira na JavaScript-u, tipovi podataka se također odnose na JavaScript. Osnovni tipovi podataka sastoje se od objekata, booleana, funkcija i simbola. Objekti se definiraju putem oznake „var“, ali i sa oznakama „let“ i „const“. Node.js, kao i JavaScript, definira tipove podataka dinamično ovisno o podatku koji se dodjeljuje varijabli [3]. Primjerice ako se definira varijabla „ime“ i dodijeli joj se tekstualna vrijednost, ona će poprimiti tip podatka string. Tip podatka boolean se također definira putem „var“ oznake i jednostavno sadrži vrijednost „false“ ili „true“.

Objekti mogu imati svoja svojstva i funkcije. Njih se definira sa vitičastim zagradama unutar kojih se navode svojstva i funkcije koje će taj objekt sadržavati, primjerice:

```
var osoba = {
     ime: 'Kristian',
     pozdrav: () => { console.log('Bok, ja sam ' + this.ime); }
};
```

Preko objekta osoba se sada može pozivati funkcija pozdrav() te pristupiti svojstvima, u ovom slučaju svojstvu “ime”. Nizovi podataka definiraju se uglatim zagradama i odvajaju se

zarezom. Operacije nad nizovima uključuju push(), dodavanje u niz, i pop(), vađenje elementa iz niza. Objekti se mogu dodati u nizove, te objekt može imati svojstvo koje označava niz objekata ili svojstava. Niz je moguće iterirati pozivanjem funkcije foreach():

```
students.forEach((student) => {
     console.log(student.ime);
}
```

U ovom primjeru se iterira niz studenata te se ispisuje svojstvo „ime“ za svakog pojedinog studenta. Nizove ili svojstva objekta se mogu kopirati u novonastali objekt ili niz pomoću oznake „…“, primjerice ako se želi kopirati objekt osoba u novi niz:

```
var nizOsoba = […osoba];
```

Ranije su spomenute oznake „const“, kojom se definira konstanta čija se vrijednost neće mijenjati, i „let“, kojom se definira varijabla čiji se obuhvat odnosi samo na blok unutar kojeg je deklarirana [5]. Objekti i funkcije mogu se odvajati u modele te se ti modeli mogu stvarati putem oznake „new“. Primjer toga je objekt „Date“ koji predstavlja datum:

```
var datum = new Date();
```

Dakle, poziva se konstruktor objekta Date koji stvara novu inačicu tog objekta. Objekti se mogu uspoređivati operatorima jednakosti (==, <, >, <=, >=) koji vraćaju Boolean povratnu vrijednost. Postoji i nekoliko funkcija za rad na varijablama, primjerice za varijablu tipa String neke od funkcija su split(), vraća niz elemenata odvojenih po zadanom znaku, i includes(), vraća Boolean vrijednost ovisno o tome da li zadani znak postoji u Stringu.

### 4.1.3. Stvaranje poslužitelja

Stvaranje poslužitelja u Node.js-u je vrlo jednostavno. U direktoriju u kojemu se želi stvoriti novi projekt, stvara se app.js datoteka u koju se može uređivačem teksta, ili bilo kojim programom kao Visual Studio Code, unijeti sljedeće [25]:

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(port, hostname, () => {
```

```
  console.log(`Server running at port 3000`);
});
```

Dakle, prvo se mora uključiti http biblioteku koja je sam dio Node.js-a i ona je zaslužna za samo stvaranje poslužitelja. Nakon toga se mogu deklarirati varijable *hostname* i *port* koje će držati adresu i port na kojemu se pokreće poslužitelj. Naravno, te varijable ne trebaju biti definirane i vrijednosti se mogu jednostavno dodijeliti direktno funkciji listen() no radi preglednosti je poželjno ih definirati. Također, umjesto IP adrese za poslužitelja, može se definirati i „localhost“, pri čemu bi se automatski poslužitelj pokrenuo na lokalnoj IP adresi.

Prije nego se pokrene poslužitelj, potrebno ga je i definirati. Dodjeljuje se konstanti poslužitelj rezultat poziva funkcije http.createServer() koja prima jedan argument, povratnu funkciju koja se poziva svaki put kada poslužitelj primi zahtjev od korisnika. Ta povratna funkcija ima dva argumenta, req odnosno zahtjev (eng. *request)* i res odnosno odgovor (eng. response). Tim objektima se manipuliraju podaci iz poslanog zahtjeva kao i informacije koje se šalju nazad korisniku. Na odgovor se postavlja status kod na 200 kako bi se naznačilo uspješno primanje zahtjeva, postavlja se jednostavno zaglavlje u kojemu se definira tip sadržaja kao tekst, te na kraju sa funkcijom end() se šalje rezultat kao HTML zapis korisniku. Postoji i druga varijanta, a to je korištenje funkcije res.write() prije pozivanja funkcije end() [26]:

```
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.write('<html>');
  res.write('<head><title>Moj prvi server!</title></head>');
  res.write('<body><h1>Hello, World!</h1></body>');
  res.write('</html>');
  res.end();
});
```

Ovdje se ručno piše HTML kod koji se šalje korisniku. Objekt req sadrži polja „url“ i „method“ pomoću kojih se mogu filtrirati zahtjevi [26]:

```
const server = http.createServer((req, res) => {
  if (req.url === '/') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello, World!\n');
  }
   else if (req.url === '/sendmessage' && req.method === 'POST') {
     //nešto drugo
```

```
    }
});
```

Ovdje je već moguće uočiti problem kod stvaranja većih aplikacija, a to jest da je ručno serviranje HTML koda apsolutno neefikasno. Taj problem se rješava postavljanjem ruta i korištenjem fs paketa odnosno *filesystem*, točnije metode fs.readFile() [25]:

```
const fs = require('fs');

fs.readFile('./index.html', function (error, content) {
    if (!error) {
        response.writeHead(200, { 'Content-Type': 'text/html' });
        response.end(content, 'utf-8');
    }
});
```

Na ovaj način se odvajaju HTML datoteke od samog koda koji ih učitava, no postoji i bolji način, a to je cjelokupno korištenje paketa Express čije korištenje još dalje olakšava stvaranje poslužitelja. Sa Express paketom, jednostavan poslužitelj izgleda ovako [27]:


Slika 7: Express poslužitelj (Izvor: vlastita izrada)

Express poslužitelj se može stvoriti sa samo tri linije koda. Naravno, taj poslužitelj trenutno ne vraća sadržaj prilikom primanja zahtjeva ali osluškuje port 3000. Varijabla „app“ predstavlja aplikaciju. Pozivom funkcije express() stvara se Express aplikacija koja se dodjeljuje „app“ varijabli koja se dalje koristi za sve radnje vezane za poslužitelj, u ovom slučaju za osluškivanje porta.

### 4.1.4. Izvezivanje modula

Jedna od bitnijih značajki Node.js programskog jezika je njegov sistem paketa (eng. Module system). Kada se govori o paketima, ne misli se samo na pakete koji se mogu instalirati putem npm-a nego i na kod, koji uključuje funkcije, metode, varijable i sl., koji se može izvesti i učitati bilo gdje na globalnoj razini projekta. Node.js to postiže putem globalnog objekta

„module“ koji ima svojstvo „exports“ [25]. Jednostavno rečeno, definira se funkcija ili svojstvo koje se želi izvesti i koristiti na drugom mjestu u samom projektu:

```
module.exports = nekaFunkcija;
```

Treba skrenuti pažnju na to da se funkcija izvezuje bez znakova (), dakle kao svojstvo. To je jednostavno kako taj sustav djeluje. Naravno, moguće je izvesti više funkcija ili podataka, a to se radi na dva načina:

```
module.exports.nekaFunkcija = nekaFunkcija;
module.exports.drugaFunkcija = drugaFunkcija;
```

ili

```
module.exports = {
     nekaFunkcija: nekaFunkcija,
     drugaFunkcija: drugaFunkcija
};
```

Dakle može se izvesti definiranjem naziva svojstva direktno na „exports“ svojstvu ili definiranjem objekta koji sadrži funkcije kao svojstva. Treba naglasiti da se može koristiti svojstvo „exports“ i bez vodeće varijable module, no nije poželjno jer bi se time lokalno promijenila „exports“ varijabla i ona više ne bi bila povezana na globalni objekt module [25].

### 4.1.5. Middleware i preusmeravanje

*Middleware* je koncept na kojemu je građen paket Express. Koncept se može zamisliti kao vlak koji prolazi stanicama. Vlak predstavlja program, dok stanice predstavljaju *middleware*. Stanice su poredane u nizu i svaki put kada vlak naiđe na stanicu, iz njega se iskrcavaju i ukrcavaju putnici, odnosno u slučaju Express i Node.js-a, primaju se zahtjevi koji se obrađuju i šalju odgovori. Nekoliko *middleware-a* u Express-u bi izgledalo ovako [27]:

```
const express = require('express');
const app = express();

app.use((req, res, next) => {
    console.log("Jedan middleware");
    next();
});

app.use((req, res, next) => {
    console.log("Drugi middleware");
    res.send('<h1>Pozdrav svijete!</h1>');
});
```

```
app.listen(3000);
```

Učitavanjem stranice na adresi localhost:3000, na ekranu bi se dobio rezultat “Pozdrav svijete!”. Tu se nalaze dva *middleware-a* jedan iza drugoga. Uporabom funkcije app.use() pokreće se novi *middleware*. Funkcija kao parametar prima anonimnu funkciju sa parametrima req (zahtjev), res (odgovor), next. „Req“ i „res“ su objekti koji sadrže podatke o poslanom zahtjevu i odgovoru koji se šalje korisniku [27].

Pošto Express ne može sam zaključiti da je jedan *middleware* gotov sa obradom, pozivanjem next() funkcije poziva se sljedeći *middleware* u nizu. Druga opcija je korištenje jedne od metodi za slanje podataka, a jedna od njih je korištena na primjeru iznad, res.send(). Send(), kako i samo ime kaže, šalje odgovor na zahtjev i izlazi iz *middleware-a.* Druge metode koje se mogu koristiti su res.redirect() (Prima kao argument relativnu putanju na koju se preusmjerava zahtjev) i res.render() (Prima kao argument relativnu putanju do HTML dokumenta koji će se prikazati uz objekt koji prima svojstva sa vrijednostima koje se mogu proslijediti istom dokumentu) [27]. Dobra je praksa da se u zadnje pozvanom *middleware-u* na kraj uključi funkcija end() koja signalizira da je lista *middleware-a* došla kraju. Pozivanje end() funkcije nije nužno u slučaju da se koristi jedna od ranije spomenutih funkcija poput res.render(), res.redirect() ili res.send() jer one uvijek same signaliziraju Express-u da je izvođenje *middleware-a* došlo do kraja ukoliko su zadnje u nizu.

Funkcija app.use() prima još jedan argument na samom početku, a to je relativna url putanja čije učitavanje će pokrenuti taj *middleware* na kojemu je zadana [27]. Taj argument se zadaje na sam početak prije anonimne funkcije u String obliku. Primjerice:

```
app.use('/', (req, res, next) => {
    console.log("Middleware samo za url pod localhost/");
     res.end();
});

app.use('/nesto', (req, res, next) => {
    console.log("Middleware samo za url pod localhost/nesto/");
    res.end();
});
```

Na taj način postiže se koncept preusmjeravanja gdje se mogu slati drugi odgovori na različite zahtjeve. Moguće je koristiti res.redirect() unutar jednog *middleware-a* i također ga preusmjeriti u drugi kojemu prvotno taj zahtjev nije bio usmjeren. Takav pristup se koristi u konceptu koji se naziva Post-Redirect-Get (PRG). PRG je pristup kod izrade Web aplikacija koji se koristi kod izrade formi i služi tome da se POST zahtjev ne može poslati više od jednog puta. Pristup je jednostavan kako i samo ime kaže, nakon obrađivanja POST zahtjeva, jednostavno umjesto učitavanja HTML datoteke direktno, šalje se GET zahtjev na rutu koja

obrađuje taj zahtjev za tu datoteku [28]. U slučaju Express-a, to je pozivanje funkcije res.redirect() umjesto res.render(). To omogućava korisnicima da dodaju kraticu na poveznicu koja sadrži formu bez da se ona ponovo pošalje, ili da se osvježavanjem stranice ponovo ne pošalje forma već samo učita.

Moguće je obrađivati samo specifične zahtjeve primjerice tipa POST ili GET, korištenjem funkcija app.get() ili app.post() umjesto app.use(). One primaju jednake argumente i rade na jednak način kao i app.use(), no pokreću se samo na POST ili GET zahtjev [27].

Sljedeći koncept Express-a je usmjerivač (eng. Router). Usmjerivač je sličan izvezivanju modula u Node.js-u i služi za signaliziranje Express-u na kojoj lokaciji da traži određenu rutu [27]. To u principu znači da se sve rute koje se zadaju sa app.use(), mogu odvojiti u različite .js datoteke i jednostavno signalizirati Express-u da one postoje. Sljedeći primjer koji je izrađen u Visual Code-u prikazuje takvu primjenu:


Slika 8: Primjer korištenja rutera u Expressu (Izvor: vlastita izrada)

Datoteka router.js se nalazi u direktoriju “router” i sadrži rute koje su prijašnje bile definirane u app.js datoteci.

Usmjerivač se poziva definiranjem konstante „router“ i dodjeljivanjem rezultata poziva funkcije express.Router() koja inicijalizira novi usmjerivač odnosno novu rutu. Na objekt usmjerivača je moguće pozvati funkciju use() kojom se definiraju rute [27]. Na kraju datoteke, usmjerivač se izvezuje pomoću Node.js module sistema i učitava kao ruta sa app.use(), koja se naravno prije toga mora i uključiti u datoteku sa require(). Dalje je moguće još proširiti ovaj koncept i umjesto anonimne funkcije u router.js datoteci, uključiti poznatu funkciju iz druge datoteke koja može predstavljati kontroler.

Vrlo je važan redoslijed definiranja ruta. Svaki put kada Node.js primi zahtjev, Express prolazi kroz sve definirane rute redoslijedom kojim su definirane i traži rutu koja je zaslužna za obrađivanje tog zahtjeva. Pomoću toga, može se postaviti rutu koja će uvijek preusmjeravati

na stranicu 404 ukoliko određena ruta nije pronađena. Takva ruta se u konceptu postavlja na kraj niza ruta koje su definirane [26]:

```
router.use((req, res, next) => {
    res.status(404).send('<h1>Stranica nije pronađena</h1>');
});
```

Status funkcijom postavlja se status odgovora na 404 (eng. *Not found*) i šalje se jednostavan HTML odgovor.

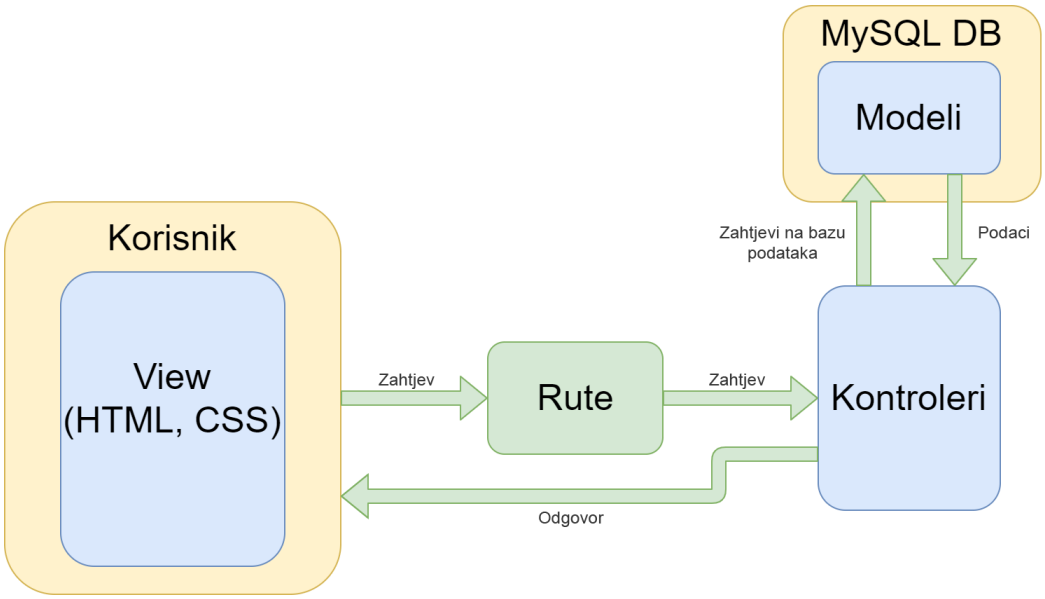
U slučaju da se želi pozvati više funkcija u jednom usmjerivaču, to je moguće napraviti nabrajanjem funkcija unutar usmjerivača odvojenim zarezima [26]:

```
router.use('/', jednaFunkcijaIliMiddleware, drugaFunkcijaIliMiddleware);
```

Time se može implementirati, na primjer, provjera da li je korisnik prijavljen u sustav prije nego što se krene sa obrađivanjem zahtjeva.

## 4.2. Model-View-Controller

Model-View-Controller (MVC) je način izrade Web aplikacija koji se primarno orijentira na podjelu aplikacije na dijelove ovisno o tome koji dio ima koju ulogu. To bi značilo da aplikacija u krajnosti ima tri dijela, model, pogled i kontroler [29].

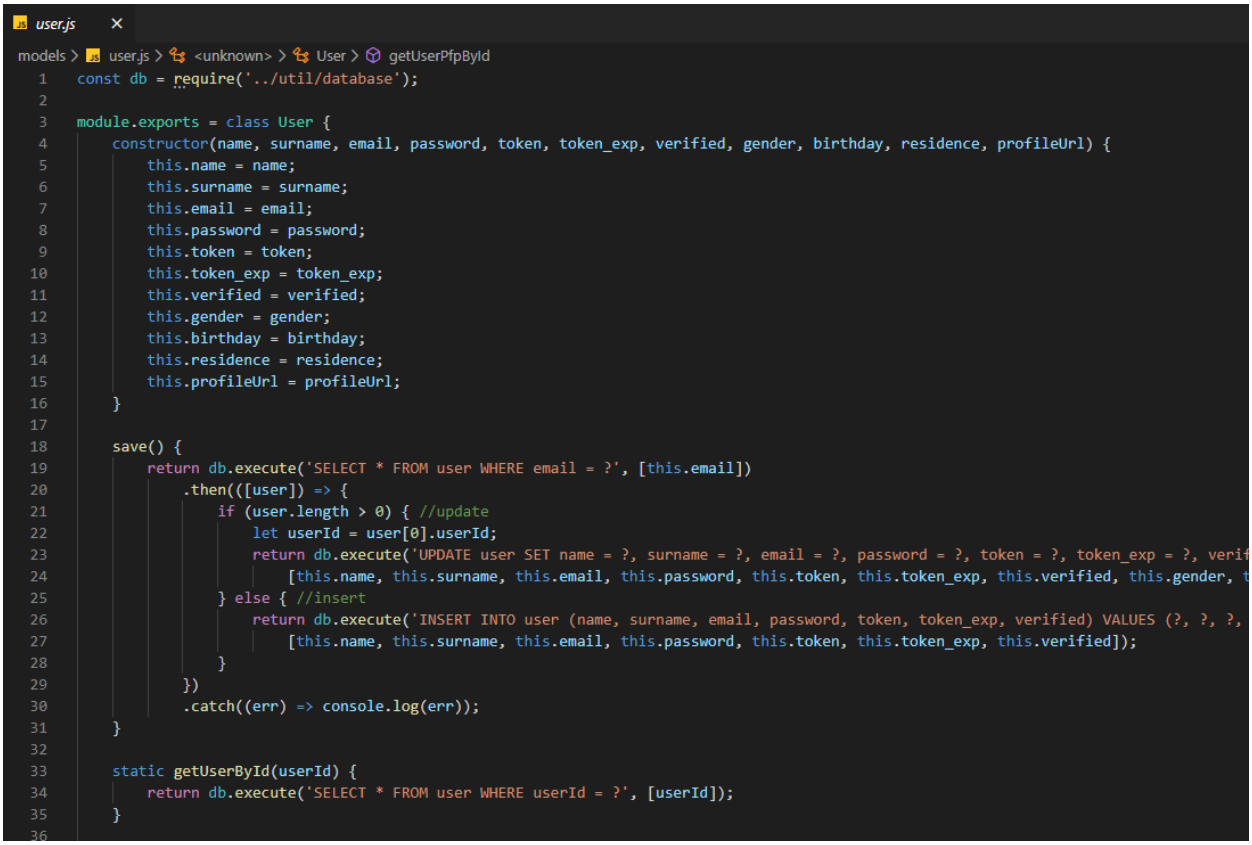



Slika 9: Model-View-Controller shema (Izvor: vlastita izrada)

Ako se pogleda primjer Web aplikacije za prodaju artikala, može se reći da ona ima svoj izgled, ima artikle koje prodaje i nekakve funkcije koje time upravljaju. Funkcije i sve što upravlja podacima na toj stranici bi bile dio kontrolera koji je nevidljiv korisniku. Ti kontroleri su

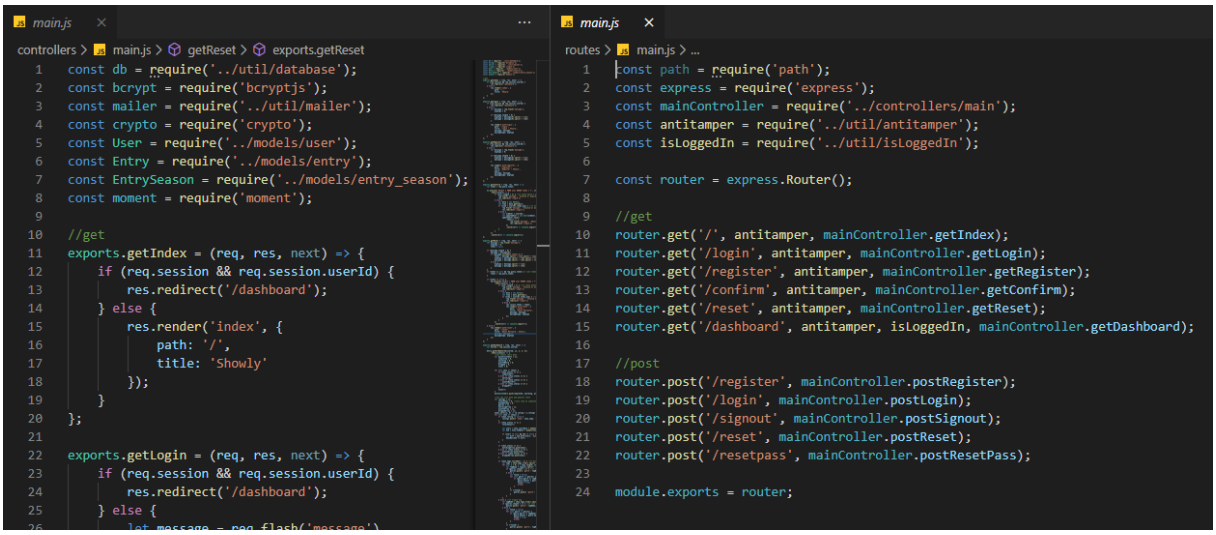
jezgra aplikacije. Artikli bi u ovom slučaju bili modeli koji sadrže same informacije o artiklima, dakle cijenu, naziv, količinu i sl. Ti modeli također mogu sadržavati i statične ili metode povezane na objekt Artikla koji dohvaćaju podatke o njima. Finalno, izgled stranice i sam prikaz podataka o artiklu spada u pogled. Pogled se obično sastoji od HTML i CSS datoteka uz moguć dodatak paketa za upravljanje tim pogledima kao EJS ili pug.

Cilj korištenja MVC metode je da se kod organizira u direktorije ovisno o vrsti kojoj pripadaju. U slučaju izrade aplikacije u sklopu ovog projekta, kontroleri su razdvojeni na dva dijela, rute koje upravljaju koji zahtjev ide kojem kontroleru i same kontrolere koji upravljaju podacima. Također, uveden je direktorij „util“ u kojemu se nalazi nekoliko metoda koje se koriste kroz više mjesta u aplikaciji te uvode u datoteke po potrebi.




Slika 10: Primjer modela u Node.js-u (Izvor: vlastita izrada)

Na slici iznad je dio koda modela za korisnika aplikacije. On sadrži konstruktor koji definira izgled samog modela kao i pripisivanje vrijednosti kod stvaranja novog korisnika. Također sadrži metodu save(), koja sprema novog ili ažurira postojećeg korisnika i statičnu metodu getUserById() koja nije ograničena na jedan objekt korisnika već se koristi za dohvaćanje bilo kojeg korisnika iz baze po ID-u. Ostale metode koje su definirane unutar ovog modela se također odnose samo na dohvaćanje podataka iz baze.



```
// controllers > main.js > getReset > exports.getReset
const db = require('../util/database');
const bcrypt = require('bcryptjs');
const mailer = require('../util/mailer');
const crypto = require('crypto');
const User = require('../models/user');
const Entry = require('../models/entry');
const EntrySeason = require('../models/entry_season');
const moment = require('moment');

//get
exports.getIndex = (req, res, next) => {
    if (req.session && req.session.userId) {
        res.redirect('/dashboard');
    } else {
        res.render('index', {
            path: '/',
            title: 'Showly'
        });
    }
};

exports.getLogin = (req, res, next) => {
    if (req.session && req.session.userId) {
        res.redirect('/dashboard');
    } else {
```

```
// routes > main.js > ...
const path = require('path');
const express = require('express');
const mainController = require('../controllers/main');
const antitamper = require('../util/antitamper');
const isLoggedIn = require('../util/isLoggedIn');

const router = express.Router();

//get
router.get('/', antitamper, mainController.getIndex);
router.get('/login', antitamper, mainController.getLogin);
router.get('/register', antitamper, mainController.getRegister);
router.get('/confirm', antitamper, mainController.getConfirm);
router.get('/reset', antitamper, mainController.getReset);
router.get('/dashboard', antitamper, isLoggedIn, mainController.getDashboard);

//post
router.post('/register', mainController.postRegister);
router.post('/login', mainController.postLogin);
router.post('/signout', mainController.postSignout);
router.post('/reset', mainController.postReset);
router.post('/resetpass', mainController.postResetPass);

module.exports = router;
```

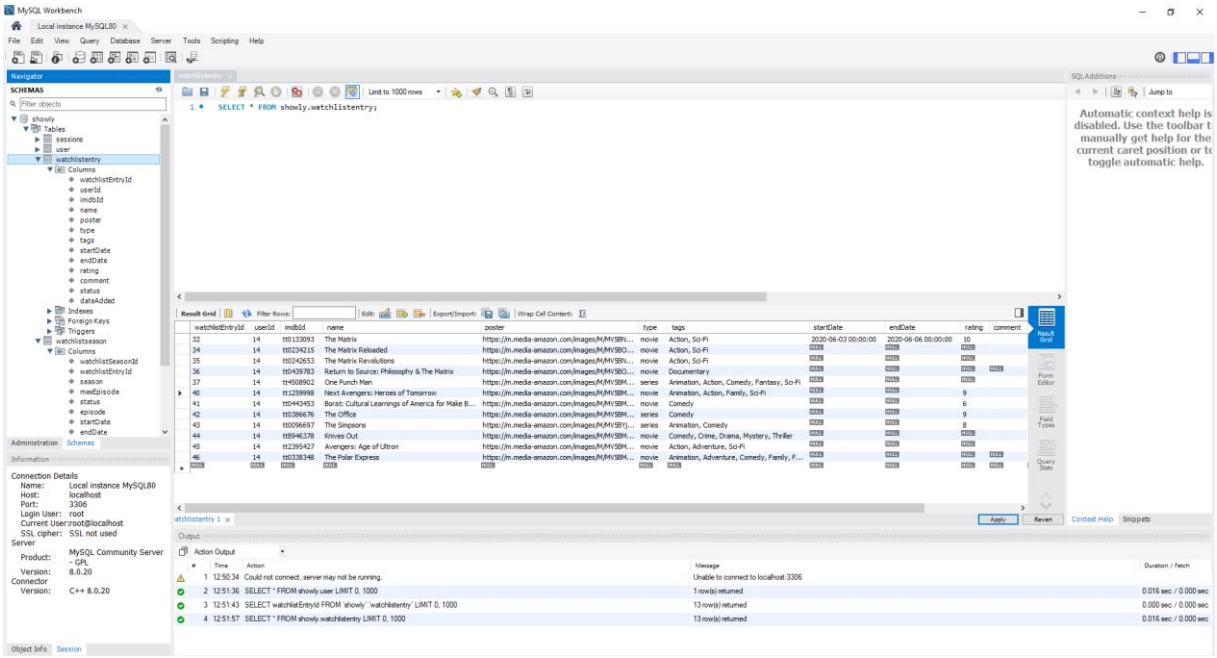
Slika 11: Primjer kontrolera i odgovarajuće rute u Node.js-u (Izvor: vlastita izrada)

Ovdje je vidljiv kontroler i njegov odgovarajući usmjerivač za glavni dio aplikacije koji sadržava rute za registraciju, prijavu korisnika, resetiranje lozinke i sl. Kod svake rute, prije poziva funkcije glavnog kontrolera, pozivaju se pomoćne funkcije „antitamper“ i „isLoggedIn“ koje se brinu o tome da li su podaci u formi prilikom slanja zahtjeva identični očekivanima na strani poslužitelja te da li je korisnik prijavljen ukoliko ruta ovisi o statusu korisnika. Vidljivo je da sama ruta uključuje kontroler putem require() funkcije kao i odgovarajuće dvije pomoćne funkcije koje koristi. U slučaju kontrolera, koji se nalazi lijevo na slici, svaka funkcija se zasebno izvezuje kako bi se mogla i zasebno pozivati u usmjerivaču.

## 4.3. Povezivanje sa MySQL bazom podataka

Prije povezivanja projekta sa MySQL bazom podataka, potrebno je instalirati MySQL Server i Workbench ukoliko se želi pokretati poslužitelj lokalno. To se može napraviti putem službene poveznice: https://dev.mysql.com/downloads/windows/installer/8.0.html . MySQL nudi alat MySQL Installer kojim se može istovremeno odabrati i instalirati MySQL Server uz Workbench. Prilikom instalacije poslužitelja, potrebno je odabrati opciju da se poslužitelj pokreće kao Windows servis, postaviti administratorsku lozinku te „legacy“ enkripciju lozinki s obzirom da većina npm modula nije kompatibilna sa novom vrstom enkripcije [26].

Nakon instalacije se može pokrenuti Workbench te stvoriti i povezati sve tablice koje će se koristiti u projektu.



Slika 12: MySQL Workbench (Izvor: vlastita izrada)

Na slici iznad je prikaz strukture dvoje tablica koje se koriste za aplikaciju. Kao i unutar Workbench-a, moguće je i unutar Node.js-a u obliku objekata dinamično definirati tablice i njihove veze pomoću paketa kao što su Sequelize za MySQL, te Mongoose za MongoDb. No kako bi se prikazalo korištenje upita i kako bi se odvojila logika baze podataka od same Node.js aplikacije, korišten je samo jedan modul, a to je mysql2 koji služi za spajanje na bazu podataka i izvršavanje upita.

Postoji i paket mysql na kojemu je mysql2 baziran. Ono što razlikuje mysql2 od mysql su dodatne značajke. Uz sve što nudi mysql, mysql2 također nudi bolje performanse, kompresiju, SSL, i sl. Nudi i tzv. udruživanje (eng. *pooling*) što znači da je moguće postaviti jednu vezu koja će ostati otvorena i obavljati sve upite umjesto da se svaki puta otvara nova. Mysql2 podržava obećanja (eng. Promises) što ga čini boljim izborom od mysql-a, te pripremljene upite odnosno izjave (eng. Prepared statements) [23]. Pripremljene izjave su vrlo korisne u zaštiti od *SQL Injection* napada jer unaprijed pripremaju i obavještavaju bazu podataka o kakvom se upitu radi te gdje se u upitu točno očekuje varijabla koja će biti poslana. Varijabla koja se šalje naknadno je jednostavno podatak koji MySQL ne gleda kao dio upita već samo kao podatak.

Prije prikaza rada sa mysql2 dodatkom, kratko će se opisati operacije sa bazom podataka. Upiti na lokalnu bazu podataka se mogu postavljati putem MySQL Workbencha. Primjerice, kako bi se dohvatili sve filmove i serije nekog korisnika koji ima identifikator se piše:

```
SELECT * FROM watchlistentry a JOIN user b ON a.UserId = b.UserId WHERE
b.UserId = 1;
```

Ukoliko se želi dodati novi korisnik u tablicu, koristimo operaciju „INSERT“ kojoj zadajemo vrijednosti koje želimo dodati, te vrijednosti koje dodajemo:

```
INSERT INTO user (name, email) VALUES (‘Kristian’,
‘nekiemail@gmail.com’);
```

Moguće je dodati više instanci odjednom jednostavno navođenjem uz odvajanje zarezom. Identifikator nije potrebno navesti i sam će se generirati ukoliko je postavljen na „Autoincrement“ prilikom stvaranja tablice. Ažuriranje tablice je moguće putem „UPDATE“ i „ALTER TABLE“ operacija. Operacija “UPDATE” služi za ažuriranje ili promjenu podataka nekog retka tablice, dok “ALTER TABLE” operacija služi za promjenu structure tablice:

```
UPDATE user SET name = ‘Marko’ WHERE userId = 1;
ALTER TABLE user CHANGE COLUMN ‘name’ VARCHAR(255) NOT NULL ;
```

Spajanje više tablica u jedan rezultat je moguće putem “JOIN” operacija, gdje je isti prikazan i u primjeru iznad kod selektiranja. Postoji više vrsta spajanja, od kojih su neki unija sa “UNION”, vrste presjeka sa “JOIN”, “LEFT JOIN”, “RIGHT JOIN”, „INNER JOIN“ i „CROSS JOIN“. Primjer unije dohvaćanja svih redaka tablice “user” i tablice “watchlistentry”:

```
SELECT * FROM user
UNION
SELECT * FROM watchlistentry;
```

Posljednje, iako nisu korišteni u ovom projektu, opisat će se okidači. Okidači se postavljaju putem „CREATE TRIGGER“ operacije koja prima ime okidača, događaj kada bi se okidač pokrenuo, na koju tablicu te SQL operacije koje bi se trebale izvršiti:

```
DELIMITER $$
CREATE TRIGGER ime_okidaca
    BEFORE INSERT
    ON ime_tablice FOR EACH ROW
BEGIN
    -- operacije koje treba obaviti
END$$
DELIMITER ;
```

Mysql2 se može instalirati kao i svaki drugi paket u Node.js-u, putem npm-a. Nakon toga, poželjno je napraviti novu .js datoteku u koju će se postaviti kontroler za otvaranje veze na bazu podataka. Primjer jednog takvog kontrolera [26]:

```
const mysql = require('mysql2');

const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
```

```
    database: 'showly',
    password: 'admin'
});

module.exports = pool.promise();
```

Kontroler se izvezuje kao obećanje kako bi se mogli vezati then() i catch() funkcije. Db kontroler se sada može uključiti u bilo koju datoteku projekta i koristiti pozivanjem jedne od mogućih funkcija koje omogućava mysql2. Jedna od njih je query() koja služi za slanje upita [23]:

```
const db = require('../util/database.js');

db.query('SELECT * FROM user')
.then(([rows]) => {
    console.log(rows);
});

db.query('SELECT * FROM user WHERE user.userId = ?', [userId])
.then(([rows]) => {
    console.log(rows);
});
```

Query() prima dva argumenta od kojih je jedan obavezan a to je sam upit u String formatu. Drugi argument se dodaje ukoliko se koriste varijable, koje se u upitu označavaju sa znakom „?“, a same vrijednosti varijabli se navode odvojene sa zarezom unutar uglatih zagrada [23]. Treći argument može biti povratna funkcija, no ovdje je izvezen db kontroler kao obećanje što znači da se može koristiti then(). Rezultat poziva se dobiva kao niz nizova, od kojih prvi niz sadrži podatke, a drugi informacije o upitu (npr. status uspjeha). U slučaju da se radi o upitu kao INSERT ili UPDATE ili slično tome, rezultat će biti samo jedan niz koji sadrži status o uspjehu odnosno neuspjehu te koliko je redova obuhvaćeno upitom.

Postoji i funkcija execute() koja je gotovo identična query() funkciji, ali razlikuje se u jednoj bitnoj stavki koja ju čini boljom za uporabu u većini slučajeva, pripremanje upita [23]. Dakle, ukoliko se želi zaštiti aplikacija od SQL Injection napada, poželjno je koristiti execute() funkciju koja će unaprijed pripremiti upit i naknadno poslati varijable.

Korištenjem ovog paketa znatno je jednostavno povezivati više upita u svrhu obrade podataka:

```
save() {
    return db.execute('SELECT * FROM user WHERE email = ?', [this.email])
```

```
        .then(([user]) => {
            if (user.length > 0) { //update
                let userId = user[0].userId;
                return //db.execute UPDATE user SET …
            } else { //insert
                return //db.execute INSERT INTO user VALUES …
            }
        })
        .catch((err) => console.log(err));
}
```

Primjer koda uzet je iz modela korisnika iz aplikacije, gdje sama metoda služi za spremanje novih podataka o korisniku ukoliko on postoji ili spremanje novog korisnika ukoliko ne postoji. Iz svih funkcija se vraća sa „return“ kako bi se povezala sva obećanja i vratili podaci u točno onom trenutku kada se zadnji upit provede.

## 4.4. Dinamično generiranje sadržaja

Kako bi se dinamično generirao HTML sadržaj, kroz projekt je korišten dodatan paket EJS. EJS je tzv. *templating engine*, jedan od mnogih kao Pug ili Handlebars, koji omogućava korištenje JavaScript-a unutar HTML dokumenata [10]. To znači da se mogu koristiti svi objekti unutar HTML-a koji se pošalju preko Express-ove render() funkcije. Na primjer, ako se želi, može se iterirati kroz niz objekata i dinamično pripisivati vrijednosti na klase unutar HTML-a kao i generirati odgovarajući broj „div“ elemenata.

EJS se može koristiti u nizu unutar vrijednosti atributa pojedinih HTML elemenata. Također je moguće uključivati druge HTML datoteke pomoću include() funkcije [10]. U slučaju aplikacije, izvezen je HTML kod za navigaciju i CSS import te je dinamično uključivan u ostale datoteke kako se ne bi morao mijenjati na više mjesta, već na jednom.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <%- include('../includes/favicon_import.ejs') %>
    <title><%= title %></title>
    <%- include('../includes/css_import.ejs') %>
</head>
```

Slika 13: Primjer uvezivanja EJS datoteka (Izvor: vlastita izrada)


Slika 14: Primjer korištenja EJS-a (Izvor: vlastita izrada)

Na slici iznad je prikazan isječak HTML koda za stranicu za pretraživanje unutar aplikacije. Unutar znakova „<% %>" se piše JavaScript kod koji u ovom slučaju gleda da li je objekt "shows" popunjen te prema tome ulazi u petlju koja generira onoliki broj div elemenata koliko ima tih objekata. U donjoj else funkciji je tipično uspoređivanje sa JavaScript-om koje generira različiti HTML ovisno o rezultatu. Sa znakovima "<%=" se dodjeljuje vrijednost varijable nekom polju, čija je uporaba vidljiva kod „img" elementa i njegovog „src" atributa kao i kod dodjeljivanja vrijednosti h3 elementu iznad.

## 4.5. Rad sa podacima iz obrazaca

Node.js podatke u poslanim zahtjevima sprema u komadiće (eng. chunks) koji se preko strujanja (eng. stream) dostavljaju poslužitelju u slučaju POST zahtjeva. Ti komadići se šalju jedan po jedan sve dok se cijeli podatak ne pošalje i dok ne završi strujanje [26]. Tim podacima se može pristupiti preko privremenog spremnika (eng. *buffer*). Taj spremnik je objekt koji reprezentira podatke u obliku bajtova. Koristi se na sljedeći način [26]:

```
const server = http.createServer((req, res) => {
  if (req.method == POST) {
        var body = [];
        req.on('data', (chunk) => {
             body.push(chunk);
        });
        req.on('end', (chunk) => {
             var parsedBody = Buffer.concat(body).toString();
        });
    }
```

```
});
```

Prvo se postavljaju dva osluškivača događaja od kojih se jedan pokreće svaki put kada uoči da se šalje jedan djelić podatka. U tom osluškivaču se pokreće funkcija koja u jednostavan niz stavlja komadiće podataka. Drugi osluškuje „*end*“ događaj odnosno kraj jednog strujanja nekog podatka gdje preko spremnika i metode toString(), pretvara podatke u čitljiv niz teksta. Taj tekst sadrži parove ključ i vrijednost odvojeni znakom =. Dalje se može koristiti npr. metoda split() kako bi se ti podaci odvojili i poprimili mogućnost rada na njima.

Da bi se lakše došlo do podataka, koristi se modul pod imenom body-parser. Body-parser je modul koji rastavlja podatke i sprema ih u req.body objekt zajedno sa atributom kojim je moguće pristupiti svakom podatku. Ti podaci su dostupni unutar svakog *middleware-a* od Express-a, što znači da ovaj modul zahtjeva instaliran Express. Da bi se modul koristio, potrebno ga je instalirati putem npm-a i uključiti u app.js datoteku [13]:

```
var bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: false }));
```

Pozivanjem urlencoded metode na bodyParser objekt se signalizira paketu da koristi rastavljač (eng. *parser*) za forme tipa “x-www-form-urlencoded”. Paket podržava i json rastavljanje koje se može uključiti pomoću metode bodyParser.json() unutar app.use(). Svaka od tih metoda prima nekoliko argumenata, odnosno opcija koje se mogu koristiti. U ovom slučaju, korišten je „extended: false“ što znači da ne koristi biblioteku koja omogućava da se razni objekti i nizovi mogu šifrirati u URL-encoded formatu, već samo u „*query-string*“ obliku odnosno obliku ključa i String vrijednosti [13]. Također, moguće je postavljati i ograničenja na veličinu sveukupnih podataka ili broj parametara koji se šalju, kao i uključivati paket samo u metodama koji ga trebaju koristiti, navođenjem kao *middleware* unutar app.use() kao što se to radi i sa drugim funkcijama. U projektu, definiran je globalno da se ne mora definirati zasebno na više mjesta.

Do podataka sa ovim paketom se može doći na jednostavan način, primjerice za formu oblika:

```
<form action="/dodaj" method="POST">
     <input type="text" name="ime">
     <input type="number" name="broj">
     <button type="submit">Dodaj</button>
</form>
```

Unutar funkcije za obradu „/dodaj“ POST zahtjeva, do podataka se može jednostavno doći pristupanjem svojstva req.body.imeparametra, pri čemu je ime parametra samo ime „input“ elementa čija vrijednost se želi dohvatiti [13]:

```
exports.postDodaj = (req, res, next) => {
     var ime = req.body.ime,
         broj = req.body.broj;
};
```

Ako se koristi GET metoda prilikom slanja forme, parametri se neće nalaziti unutar „body“ elementa u zahtjevu niti je potrebno koristiti dodatne pakete, već će se podaci poslati unutar URL-a odvojeni sa znakom „&“ u obliku ključ-vrijednost. Primjerice, u gornje navedenom slučaju, ako bi se slao GET zahtjev umjesto POST zahtjeva, URL format bi bio izgleda „dodaj?ime=nekavrijednost&broj=drugavrijednost“.

Podaci se mogu dohvatiti iz samo URL-a, no u Node.js-u postoji i „query“ objekt na „req“ objektu kojim je moguće pristupiti upravo tim podacima. Primjerice [26]:

```
exports.getDodaj = (req, res, next) => {
     var ime = req.query.ime,
         broj = req.query.broj;
};
```

Nije garantirano da će svi ti podaci biti upravo oni koji se očekuju, što znači da ih je potrebno provjeriti na strani poslužitelja.

### 4.5.1. Validacija vrijednosti na strani poslužitelja

Provjera vrijednosti na strani klijenta je opcionalna i nije sigurna s obzirom da korisnik može isključiti JavaScript prilikom posjeta stranice. Validacija na strani poslužitelja je obavezna ukoliko se želi osigurati da korisnik uvijek pošalje podatke koji se očekuju. Vrijednosti se mogu provjeriti na razne načine. Najjednostavniji način je dohvaćanje vrijednosti iz forme i provjeravanje prije samog preusmjeravanje korisnika ili slanja odgovora. Prije svega je potrebno provjeriti da li su poslane vrijednosti popunjene usporedbom vrijednosti sa null. Ukoliko se želi, može se provjeriti i da li je tip podatka onaj koji se očekuje. Ako postoje tekstualne vrijednosti, poželjno je provjeriti da li su unutar dopuštenog raspona odnosno duljine koja je zadana kao maksimalna u bazi podataka.

Sve ostalo ovisi o podacima koji se obrađuju, primjerice, ako je potreban ID korisnika koji sprema podatke, može se provjeriti postojanje tog korisnika u sesiji ili bazi podataka prije slanja upita. Moguće je koristiti neke od paketa za validaciju vrijednosti kao express-validation, no nije potrebno.

```
exports.postLogin = (req, res, next) => {
    let email = req.body.email,
        password = req.body.password;

    if (email == null || email == '' || password == null || password == '') {
        req.flash('message', 'Email and/or password cannot be empty!=1');
        res.redirect('/login');
    } else {
        User.getUserByEmail(email)
            .then(([user]) => {
                if (user.length > 0) {
                    bcrypt.compare(password, user[0].password)
                        .then((valid) => {
                            if (valid) {
                                if (user[0].verified == 1) { //login
                                    req.session.userId = user[0].userId;
                                    res.redirect('/');
                                } else { //not verified
                                    req.flash('message', 'You have not verified your email yet!=1');
                                    res.redirect('/login');
                                }
                            }
                            else { //not a valid password
                                req.flash('message', 'Wrong password!=1');
                                res.redirect('/login');
                            }
                        })
```

Slika 15: Primjer validacije podataka (Izvor: vlastita izrada)

Na slici se nalazi isječak funkcije iz aplikacije koja je zadužena za primanje i obradu POST zahtjeva za prijavu korisnika. Ta funkcija obrađuje podatke na način da provjeri prvotno da li su svi podaci uneseni i da li korisnik koji se pokušava prijaviti uopće postoji. Nakon toga pomoću bcrypt paketa provjerava ispravnost unesene lozinke. Na kraju se gleda da li je korisnik validan, odnosno da li mu je račun potvrđen i nakon toga se postavlja sesija i korisnika se preusmjerava na početnu stranicu. Svaka od funkcija za dohvat podataka i obradu je asinkrona što znači da se ne kreće na idući korak ukoliko prošli nije završen. Ukoliko ne prođe bilo koji korak u ovoj validaciji, korisniku se postavlja povratna poruka i preusmjerava ponovo na stranicu za prijavu.

### 4.5.2. Slanje povratnih poruka

S obzirom da se svakim novim zahtjevom ponovo pokreće Node.js ciklus, to znači da se podaci ne čuvaju između zahtjeva. Ipak, podatke je moguće sačuvati između zahtjeva na nekoliko načina. Jedan vrlo poznat način je naravno spremanje podataka unutar sesija, no ako bi se spremali senzitivni podaci poput ID-a korisnika, bili bi dostupni korisnicima za pregled, što nije poželjno.

Drugi način je spremanje preko lokalnih Node.js varijabli (eng. *locals*). Te lokalne varijable postavljaju se unutar glavne .js datoteke (u slučaju aplikacije projekta, app.js) i sadržavaju postavljene podatke .Njih je poželjno postaviti pri samom kraju dokumenta, ali prije definiranja korištenih ruta kako bi unutar ruta bile dostupne.

Postavljaju se na objektu res.locals [26]:

```
app.use((req, res, next) => {
    res.locals.lastQuery = '';
    res.locals.lastType = '';
    next();
});
```

Svakako je potrebno pozvati next() na kraju kako bi se prešlo u sljedeći *middleware*. Te varijable unutar app.js datoteke se samo definiraju i dodjeljuju im se prazne vrijednosti kako bi se dalje koristile u obradi zahtjeva:

```
exports.postFunkcija = (req, res, next) => {
     var query = req.body.lastQuery;
     res.locals.lastQuery = query;

     …
     res.render('/search', {
            result: result
     });
};
```

Lokalne varijable koje se dodjeljuju mogu se koristiti samo za naredni res.render() odgovor, nakon čega se ponovo resetiraju i zahtijevaju ponovno postavljanje. Unutar res.render() ih nije potrebno postavljati ručno kao i ostale podatke koji se žele poslati sa odgovorom. Zbog ograničenja, te lokalne varijable se ne mogu prenositi kod pozivanja res.redirect() funkcije koju je poželjno koristiti ako se ne želi da korisnici imaju mogućnost ponovnog slanja forme [26]. Kako bi se taj problem riješio, koristi se paket connect-flash.

Connect-flash je paket koji omogućava prijenos povratnih poruka u obliku String-a putem sesije. Sesija se stvara kod postavljanja poruke, te prvim pozivom povratne poruke briše. Kao i svaki drugi paket, connect-flash se prvotno postavlja u glavnoj .js datoteci [20]:

```
const flash = require('connect-flash');

app.use(flash());
```

Funkciju flash() je potrebno pozvati nakon definiranja sesije, u protivnom neće raditi. Povratne poruke se postavljaju na „req“ objektu koji će sada sadržavati funkciju flash() [20]:

```
req.flash('message', 'Wrong username or password!');
```

Nakon toga, može se preusmjeriti na drugi GET zahtjev gdje se može dobaviti bilo koja od postavljenih poruka korištenjem ključa pod kojom je i postavljena [20]:

```
var message = req.flash('message');
```

Poruka se, kao i svaka druga varijabla, može zatim dodati u res.render() funkciju i koristiti uz EJS.

### 4.5.3. Prijenos datoteka i multipart forme

Paket body-parser nema mogućnost rukovanja sa multipart formama, zbog toga se koristi paket multer koji omogućava obradu multimedijskih podataka, odnosno slika.

```
//file handling
const fileFilter = (req, file, cb) => { //prevent other image/non-image formats
    if (file.mimetype === 'image/png' ||
        file.mimetype === 'image/jpg' ||
        file.mimetype === 'image/jpeg') {
        cb(null, true);
    } else {
        cb(null, false);
    }
}
const fileStorage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'images');
    },
    filename: (req, file, cb) => {
        crypto.randomBytes(16, (err, buffer) => {
            if (err) {
                console.log(err);
            }
            var extension = '.' + file.mimetype.split('/')[1];
            var rand = buffer.toString('hex');
            cb(null, req.session.userId + '_' + rand + extension);
        });
    }
})
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'));
```

Slika 16: Primjer korištenja multer paketa (Izvor: vlastita izrada)

Slika prikazuje isječak koda iz aplikacije unutar app.js datoteke, zadužen za rukovanje prijenosom datoteka. Kod je izrađen prema predavanju Maximiliana Schwarzmüllera [26]. Kod je poželjno smjestiti prije inicijalizacije ruta.

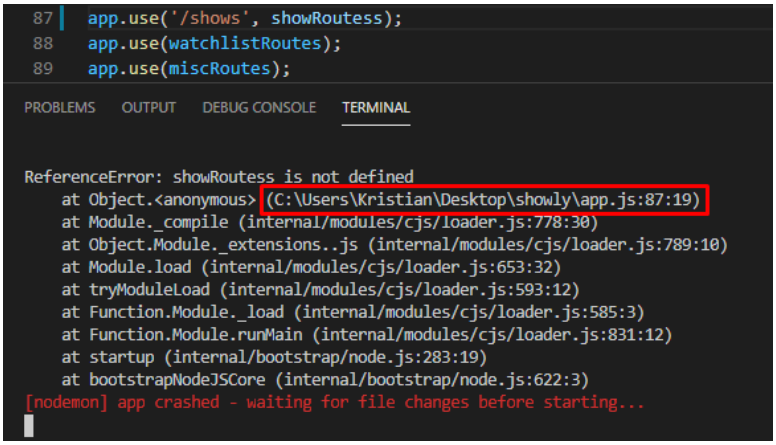
Pozivanjem funkcije multer() inicijalizira se paket, te dodavanjem argumenata kao „storage“ i „fileFilter“ mogu se zadati neke od opcija po želji. Unutar „fileStorage“ je zadana opcija „destination“ koja određuje gdje će se slike spremati na poslužitelju, te „filename“ koja određuje kako će se slika zvati, u ovom slučaju se generira slučajan niz od 16 znakova koji se spaja sa ID-em od korisnika i ekstenzijom datoteke. U funkciji fileFilter() je definirano koje vrste datoteka su dopuštene. U svakoj funkciji je potrebno pozvati funkciju „cb“ odnosno povratnu funkciju zadanu od strane multer paketa, čiji je drugi argument opcija koja se prosljeđuje. Naknadno, na multer() funkciju na samom kraju poziva se i funkcija single() kojom se naznačuje očekivanje samo jedne datoteke tipa slika [19].

Naravno, kako bi multer „pokupio“ sliku i spremio ju na poslužitelj, treba postaviti „enctype“ opciju forme na „multipart/form-data“. Unutar *middleware-a* može se pristupiti

objektu req.file koji sadrži podatke o spremljenoj slici kao i putanju koja se može spremiti u bazu podataka i koristiti za dohvaćanje slika korisnika.

## 4.6. Pronalaženje grešaka

Ni jedna aplikacija nije savršena i uvijek će postojati neke greške koje će biti potrebno ispraviti ili zaobići. Postoji nekoliko vrsta grešaka i svaka od njih se razlikuje po načinu njihove obrade i pronalaženja, pa tako greške uključuju tehničke greške, tzv. očekivane greške i logičke greške. Tehničke greške su greške nastale uglavnom prilikom spajanja na bazu podataka ili nekog servisa o kojemu aplikacija ovisi, prilikom čega nije moguće koristiti aplikaciju. Za takve greške je poželjno postaviti povratnu poruku korisniku aplikacije o tome da se dogodila greška te, na primjer, ponuditi mogućnost slanja maila sa povratnim informacijama o grešci. Sljedeća vrsta su „očekivane“ greške. One nastaju upravo na mjestima na kojima je očekivana pojava greške primjerice prilikom čitanja podataka sa diska, operacija sa bazom podataka, šifrirana lozinki i slično. Rješavaju se na sličan način kao i tehničke greške, dakle informiranjem korisnika i mogućnošću ponovnog pokušaja operacije koja je izbacila grešku. Zadnja vrsta su logičke greške. Takve greške nije potrebno obrađivati, već ih je potrebno ispraviti jer su rezultat loše pisanog koda. Svaku od ovih vrsta grešaka je moguće pronaći i ispraviti pomoću povratnih informacija koje dobivamo u konzoli prilikom pokretanja ili rada sa aplikacijom [26]. U slučaju Visual Studio Code-a, u konzoli je vidljiva greška:




Slika 17: Primjer greške u Studio Code-u (Izvor: vlastita izrada)

Tip greške vidljiv na slici iznad je logička greška. U konzoli je vidljiva vrsta greške, u ovom slučaju greška reference, te kratka poruka o grešci zajedno sa lokacijom na kojoj se dogodila koja sadrži i točnu liniju te početni znak u liniji. Iz poruke se može zaključiti da objekt „showRoutess“ ne postoji, dakle, potrebno je provjeriti da li je taj objekt zaista definiran prije korištenja, no u ovom slučaju ga je samo potrebno preimenovati u „showRoutes“.

Za većinu grešaka, javiti će se slična povratna poruka u konzoli no u nekim slučajevima, primjerice kod dodatka mysql2, ispisati će se greška koju neće izbaciti JavaScript nego mysql2. To su prilagođene greške koje se obično ispisuju od strane dodataka vezane uz njihove metode. S obzirom da se svaka takva poruka o grešci razlikuje između dodataka, lakoća pronalaženja i popravka će ovisiti o kompletnosti sadržaja poruke.

### 4.6.1. Obrađivanje pogrešaka

Svaka aplikacija koja se izrađuje mora imati način obrađivanja pogrešaka. To je potrebno kako bi se pogreška zapisala unutar neke log datoteke te postavila mogućnost korisniku za nastavak uporabe aplikacije. S obzirom da se Node.js kod razlikuje na dva načina pisanja (asinkroni i sinkroni), postoje i dva načina obrada pogrešaka koji su gotovo identični, osim što su za druge scenarije. Prvi način je pomoću try-catch blokova koji se koriste unutar sinkronog koda. Radi na principu da se unutar try bloka, piše nekakav kod koji može ali ne mora izbaciti pogrešku, a unutar catch bloka koji se veže na try blok, se piše kod koji rješava pogrešku, primjerice console.log i preusmjeravanje [26].

Drugi način je putem then-catch blokova koji se koriste unutar asinkronog koda. Po prirodi, then() se može vezati neograničen broj puta i svaki blok će se izvoditi nakon završetka prethodnog. Na isti način, funkcija catch() se veže na zadnji then blok koji će „uhvatiti“ i obraditi bilo koju pogrešku koja se dogodila unutar bilo kojeg vezanog then bloka. Catch() se može vezati više puta unutar povezanih then blokova, odnosno na svaki blok [26].

```
bcrypt.hash(password, 12, (err, hash) => {
    if (!err) {
        crypto.randomBytes(32, (err, buffer) => {
            if (err) {
                console.log(err);
                return res.redirect('/');
            }

            let token = buffer.toString('hex');
            let date = new Date();
            let token_exp = date.getTime() + 86400000; //token valid for 1 day since generation

            let user = new User(null, null, email, hash, token, token_exp, 0, null, null, null, null);
            user.save()
                .then(() => {
                    req.flash('message', 'User successfully created, check your email!=2');
                    res.redirect('/login');
                    return mailer.sendVerificationEmail(email, token, 'register');
                })
                .catch((err) => {
                    const error = new Error(err);
                    error.httpStatusCode = 500;
                    return next(error);
                });
        })
    } else {
        throw new Error(err);
    }
```

Slika 18: Korištenje blokova za obradu pogrešaka (Izvor: vlastita izrada)

Na slici je prikazan isječak koda iz aplikacije za generiranje šifrirane lozinke i spremanje korisnika u bazu. Ukoliko funkcija bcrypt.hash() vrati pogrešku, prelazi se u else blok koji baca

novu pogrešku i zaustavlja aplikaciju. Da bi se nastavilo korištenje aplikacije, moguće je sve unutar bcrypt.hash() funkcije omotati u try-catch blok, no u slučaju ove aplikacije se izbacuje pogreška. Na user.save() funkciji uključen je then-catch blok, koji ako uhvati pogrešku postavlja status kod na 500 (eng. *Server error*) i poziva sljedeći *middleware*.

Express ima poseban *middleware* zadužen za obrade pogrešaka u kojega prelazi samo onda kada otkrije poziv funkcije next() zajedno sa objektom Error. Ta funkcija se dodaje u glavnu .js datoteku (u ovom slučaju, app.js), na sam kraj iza poziva ostalih ruta [26]. Unutar nje se može dodati bilo kakav kod za obradu pogrešaka, primjerice console.log koda i preusmjeravanje na rutu koja korisniku prikazuje informativnu stranicu o događaju pogreške, koji se i koristi u aplikaciji projekta:

```
app.use((error, req, res, next) => {
    console.log(error);
    res.redirect('/500');
});
```

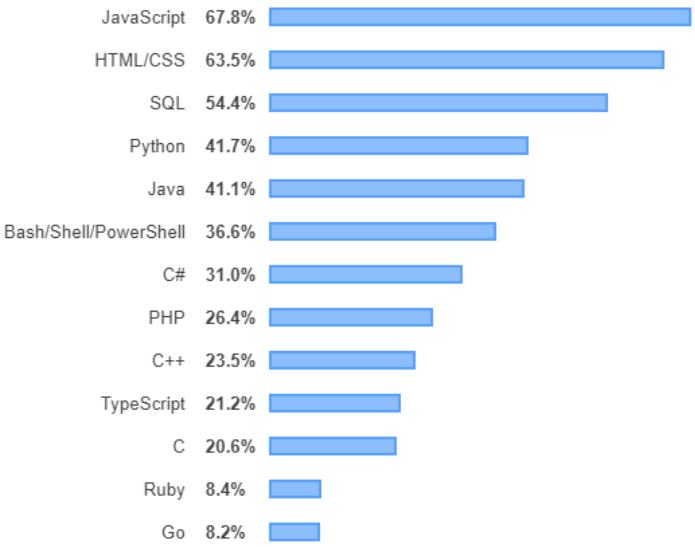
# 5. Usporedba Node.js, Python i PHP

Node.js je jedan od programskih jezika na strani poslužitelja koji se mogu koristiti za izradu web aplikacija. Tu se također uključuju PHP i Python. Svaki od tih jezika je pogodan za izradu aplikacija ovisno o tome što nam je potrebno.

Ako se pogledaju Node.js i Python, prva uočena razlika je da Node.js zapravo nije programski jezik već okruženje za JavaScript. Za razliku od Node.js-a, Python je programski jezik nastao 1991. godine, što ga čini zrelijim „jezikom“ od Node.js-a. Još jedna razlika je *engine* na kojemu rade. Kod Node.js-a je to Chrome V8 napravljen od strane Google-a koji je izuzetno poznat po brzini. Python se također pokreće preko *engine-a* napravljenog od strane Google-a, App Engine. No jedna velika razlika između ova dva jezika je da Node.js radi na principu događaja (eng. events), što znači da je asinkron. To mu omogućava rukovanje sa nekoliko zahtjeva istovremeno bez prekidanja daljnjeg rada aplikacije. S druge strane, Python se bazira na sinkronom principu rada i zbog toga nije pogodan ukoliko se žele izrađivati brze pravovremene (eng. *real-time*) aplikacije. Moguće je koristiti module u Pythonu koji mu omogućavaju asinkroni rad nalik Node.js-u, no nisu jednostavne za postaviti. Ako se govori o većim projektima i proširenjima postojećih aplikacija, tu je Node.js također iznad Pythona. Node.js aplikacije je moguće jednostavno proširiti dodavanjem novih servisa, modula i resursa. Node.js ima dobru skalabilnost iz razloga što je izuzetno modularan i podržava rad s više dretvi, dok Python radi samo sa jednom dretvom istovremeno što ga čini nepogodnim za daljnja proširenja. Broj paketa i okvira koje podržavaju oba jezika su vrlo slični. Svaki od njih ima svoj menadžer paketa pomoću kojega ih je moguće dodati u projekt. Python koristi Pip

installs Pythin (PiP), dok Node.js koristi Node package manager (npm). Sintaksa i vrijeme učenja oba jezika ovisi o osobi, no generalno se Python smatra lakšim za naučiti. Ono što Node.js čini vrlo poželjnim u današnjem svijetu izrade Web aplikacija je da je baziran na JavaScript-u, što znači, da se koristi isti jezik za klijentski i poslužiteljski dio aplikacije te mogućnost izrade i mobilnih i stolnih aplikacija neovisno o operacijskom sustavu. Python nije namijenjen izradi mobilnih i fleksibilnih aplikacija, no izuzetno se koristi kod AI rješenja i aplikacija namijenjenih za analizu velikog broja podataka (algoritmi za prepoznavanje lica, strojno učenje, neuronske mreže) [30].

PHP, kao Python i Node.js, je jedan od najpoznatijih programskih jezika na strani poslužitelja. Originalno, PHP nije imao specifikaciju sve do 2014. godine. Za razliku od Node.js-a koji je interpretacijski jezik, PHP je skriptni jezik. Kao i Python, PHP je sinkron i zbog toga sporiji od Node.js-a, no asinkron kod ne dolazi bez mana. Za razliku od sinkronog, kod asinkronog koda se može dogoditi da izvedba nikad ne izađe iz povratnih funkcija. Jedno od rješenja je pokrenuti kod kao sinkron, što je u Node.js-u također moguće, ili jednostavno voditi računa o funkcijama koje se povezuju. Uz PHP se većinom koristi LAMP platforma (Linux-Apache-MySQL-PHP), odnosno pogodan je za korištenje relacijske baze podataka kao postgreSQL ili MySQL. Uz Node.js je popularna MEAN platforma (MongoDb, Express, Angular, Node.js) koja uključuje rad sa MongoDb bazom podataka koja nije bazirana na relacijama i orijentirana je na brzinu. Naravno, to ne znači da se MySQL ne može koristiti uz Node ili MongoDb uz PHP, već samo prikazuje današnji trend. Node.js i dalje drži titulu brzine i proširenja, no PHP je pogodan za izradu aplikacija koje zahtijevaju puno CPU snage. Proširenje paketima je moguće u oba jezika. Za Node.js je ranije spomenut npm, a PHP koristi dva alata, PEAR i Composer. PEAR služi za distribuiranje PHP komponenti, dok Composer služi za određivanje komponenti o kojima zavisi projekt [31].




Slika 19: Najpopularniji programski jezici 2019. godine (Izvor: Stack Overflow, https://insights.stackoverflow.com/survey/2019#technology)

Iz grafa na slici je vidljivo koliko je zapravo JavaScript zajedno sa svojim proširenjima popularan. To naravno ne znači da jezici poput Pythona ili PHP-a nisu, već je potrebno odabrati programski jezik ovisno o vrsti aplikacije, odnosno projekta na kojem želimo raditi.

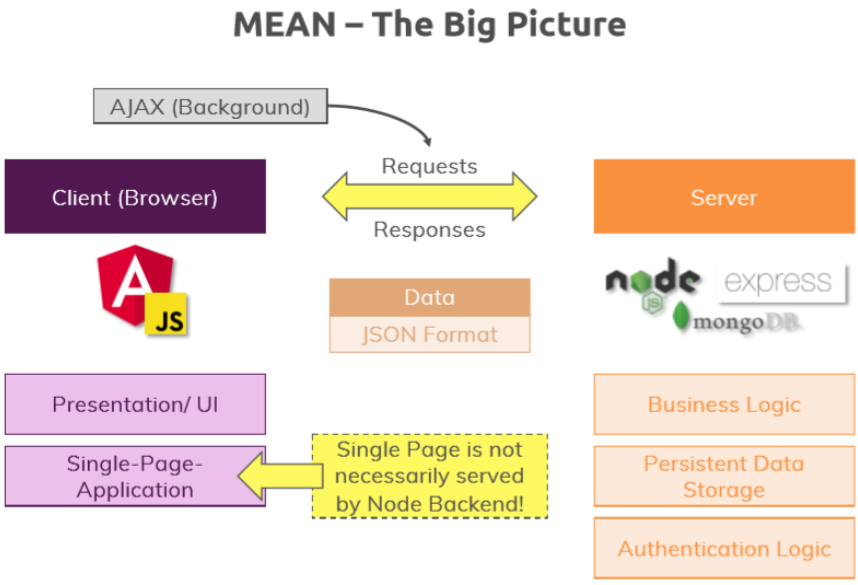
# 6. MEAN platforma

MEAN platforma (eng. MEAN stack) je skup tehnologija koje se koriste za izradu jedno-straničnih responzivnih aplikacija. Taj skup se sastoji od četiri tehnologije [32]:

- MongoDb – Baza podataka bez relacija i shema koja se orijentira na brzinu i format podataka u json-u.
- Express – Predstavlja Node.js modul za upravljanje zahtjevima
- Angular – Upravlja klijentskim dijelom aplikacije i omogućuje izradu jedno-straničnih aplikacija (eng. Single Page Application - SPA)
- Node.js – Glavni poslužitelj koji povezuje ostale komponente

Ono po čemu se MEAN platforma znatno razlikuje od tipičnih Node.js projekata je izrada jedno-straničnih aplikacija što omogućuje Angular. Naravno, postoje druge tehnologije koje se mogu koristiti primjerice Vue i Ajax kako bi se izradile jedno-stranične aplikacije, no trenutno najpopularniji je Angular. Jedno-stranična aplikacija radi na principu da se koristi samo jedan html dokument koji se prikazuje u pregledniku korisnika. Angular uzima i mijenja

taj HTML dokument sa novim podacima dobivenim od strane Node.js-a i ponovo prikazuje klijentu [32].




Slika 20: Način rada MEAN platforme (Izvor: [32])

Na slici je prikazan rad tipične MEAN aplikacije prema Maximilianu Schwarzmülleru [32]. Angular služi za prikazivanje podataka i sam izgled aplikacije. Angular i Node.js izmjenjuju podatke putem zahtjeva u json formatu, dok se na Node.js poslužiteljskoj strani ti podaci šalju ili dobivaju iz baze, u ovom slučaju MongoDb. Sva logika vezana za aplikaciju koja uključuje autentifikaciju, validaciju podataka, spremanje, brisanje i slično se također nalazi na Node.js poslužitelju [32].

## 6.1. Implementacija MEAN platforme

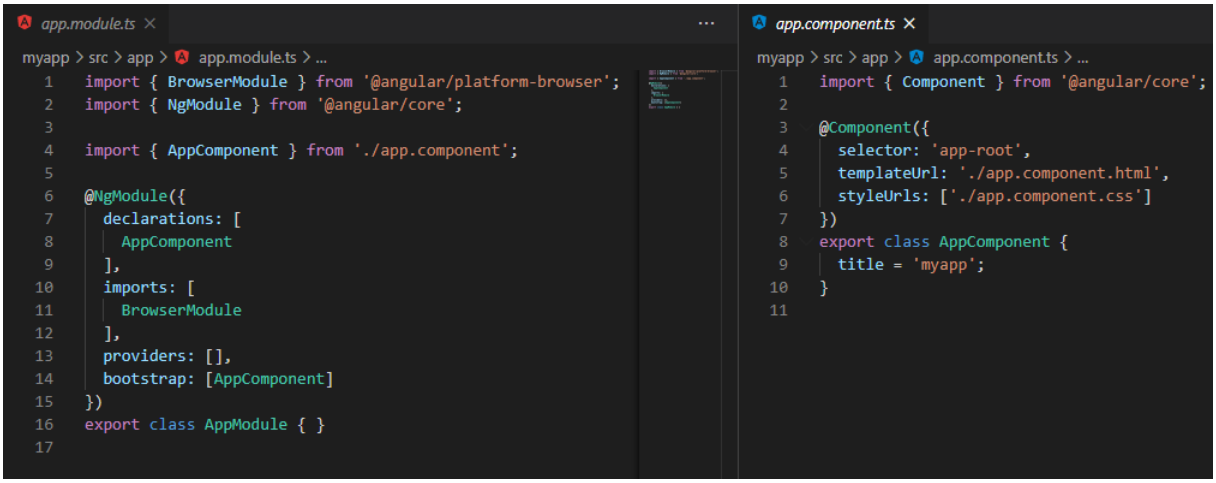
U ovom poglavlju prikazati će se primjer osnovne implementacije MEAN platforme i kako komponente te platforme komuniciraju. Prije svega, valja naglasiti da je Angular samo okvir i da bez obzira na MEAN platformu, treba Node.js kako bi prevodio i pokretao js kod bez Web preglednika. Node.js instalacija je prikazana u ranijim poglavljima pa se tu neće ponavljati. Angular se može instalirati putem npm-a pokretanjem naredbe [32]:

```
npm install -g @angular/cli
```

Nakon instalacije, stvaranje novog projekta putem Angular-a moguće je putem naredbe:

```
ng new ime_aplikacije
```

Projekt se stvara u direktoriju u kojemu je trenutno postavljeno sučelje naredbenog retka. Stvorena aplikacija se može pokrenuti putem naredbe „ng serve“, te će u slučaju novostvorene aplikacije, biti prikazana jednostavna aplikacija već zadana od strane Angular-a. Novostvorenu aplikaciju moguće je otvoriti putem Visual Studio Code-a ili bilo kojeg drugog uređivača koda. Struktura aplikacije je nalik običnoj Node.js aplikaciji zajedno sa direktorijem koji sadrži Node.js pakete. Unutar glavnog direktorija se nalazi direktorij „src“ koji sadrži Angular datoteke. U tom direktoriju se nalazi direktorij „app“ koji sadrži komponente aplikacije. Te komponente su nalik Node.js modulima i svaka komponenta definira jedan dio stranice ili jednu stranicu koju Angular prikazuje. Svaka komponenta se tipično sastoji od HTML datoteke koja definira sav HTML kod, CSS datoteke koja sadrži stil koji ta komponenta koristi, i TypeScript datoteke koja definira ponašanje određene stranice. Postoji još jedna datoteka sa ekstenzijom spec.ts, no ona nije obavezna i koristi se u svrhu testiranja Angular aplikacije [32].




Slika 21: Izgled Angular komponenti (Izvor: vlastita izrada)

Na slici je vidljiv izgled TypeScript datoteke za jednu komponentu i za glavnu komponentu Angular aplikacije. Unutar komponente (desno na slici iznad) se definira selektor, odnosno ime kojim je komponenta dostupna u HTML-u, URL do kojega se može doći do HTML datoteke te komponente i stil koji vodi do CSS datoteke te komponente. Komponenta „app.module.ts“ je glavna komponenta Angular aplikacije koja definira sve ostale komponente koje se uključuju u rad aplikacije. Također definira i pakete koji se koriste kroz aplikaciju. Komponente se jednostavno mogu stvarati ručno i organizirati u različite direktorije, no prije svega moraju se uključiti unutar „app.module“ komponente kako bi bile vidljive Angular-u. Komponente se u HTML referenciraju putem naziva koji je zadan unutar same komponente, primjerice, ukoliko se komponenta naziva “mojakomponenta” [32]:

```
<mojakomponenta></mojakomponenta>
```

Unutar klase koja se izvezuje u komponenti je moguće pisati funkcije koje se pozivaju putem osluškivača dostupnih u Angularu, primjerice:

```
<button (click)=”onButtonClick()”> Spremi </button>
<textarea [(ngModel)]=”text1”></textarea>
<p>{{ text1 }} </p>
```

Putem osluškivača “onButtonClick” dobavlja se input iz „textarea“ elementa i njegova se vrijednost postavlja u p elementu. S obzirom da je orijentacija poglavlja na MEAN platformi, tu će se završiti sa primjerom djelovanja Angular-a i njegovih komponenti, a dalje će se nastaviti sa ostalim komponentama platforme.

S obzirom da Angular zapravo stvara Node.js poslužitelj, on je već uključen u projekt. Express se dodaje jednostavno instaliranjem putem npm-a. Glavni Express poslužitelj se stvara u odvojenoj JavaScript datoteci i definiraju se neke od ruta koje će se koristiti u Angular-u. Tipično se Node.js i Angular povezuju na način da se namjesti „outputPath“ unutar angular.json konfiguracije i poziva naredba „ng build -prod“ kako bi se stvorila gotova Angular aplikacija unutar direktorija koji je zadan. Zatim, u Node.js poslužitelju definira se ruta na „/“ koja će jednostavno vraćati index.html kojeg servira Angular. Jednostavan princip koji djeluje jer se ne koristi Node.js za sve rute već samo kao API koji vraća podatke koje Angular poziva putem zahtjeva. Također, potrebno je definirati direktorij u kojemu je Angular aplikacija kao statičan direktorij kako bi bilo moguće učitati sve skripte potrebne za rad aplikacije. Postoji i drugi način, a to je jednostavno serviranje oba poslužitelja pojedinačno. Dakle, Angular poslužitelj se pokreće putem naredbe „ng serve“ i Node.js putem „npm start“. Ako se ovaj način koristi, potrebno je pokretati oba poslužitelja prilikom testiranja aplikacije [32].

MongoDb baza podataka se može stvoriti online na službenoj poveznici: https://www.mongodb.com/ . Besplatna verzija je dovoljna za manje ili srednje velike aplikacije. Nakon stvaranja baze podataka, potrebno je postaviti korisnika koji će biti administrator i imati pristup bazi podataka zajedno sa lozinkom te dopustiti pristup bazi sa naše IP adrese. Sve detaljnije upute nalaze se na službenoj stranici.

Kako bi se olakšao rad sa MongoDb bazom podataka, potrebno je instalirati paket Mongoose putem npm-a koji omogućuje definiranje modela i komuniciranje sa MongoDb bazom podataka. Mongoose se spaja sa MongoDb bazom pozivanjem naredbe „mongoose.connect()“ koja prima jedan argument, String koji sadrži URL sa podacima za povezivanje. Taj String je dostupan na samoj MongoDb stranici [32].

primjer.js > ...

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

const primjerSchema = mongoose.Schema({
    ime: { type: String, required: true},
    prezime: { type: String, required: true}
});

module.exports = mongoose.Model('Primjer', primjerSchema);
```

Slika 22: Primjer Mongoose modela (Izvor: vlastita izrada)

Iz primjera na slici je vidljivo da se model definira kao objekt sa svojstvima koji se izvozi u globalnom aspektu. Nove instance modela stvaraju se pozivanjem „new“ naredbe i jednostavno popunjavanjem svojstvima kao i kod kreiranja običnih JavaScript modela [32]. Spremanje i dohvaćanje podataka obavlja se sa naredbama „save()“, „find()“ i drugih, koje se pozivaju unutar Node.js aplikacije. Modeli unutar Angular aplikacije definiraju se zasebno, no uglavnom su istovjetni modelima definiranim za MongoDb.

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { Subject } from "rxjs";
import { map } from 'rxjs/operators';

import { Post } from "./post.model";

@Injectable({ providedIn: "root" })
export class PostsService {
  private posts: Post[] = [];
  private postsUpdated = new Subject<Post[]>();

  constructor(private http: HttpClient) {}

  getPosts() {
    this.http
      .get<{ message: string; posts: any }>(
        "http://localhost:3000/api/posts"
      )
      .pipe(map((postData) => {
        return postData.posts.map(post => {
          return {
            title: post.title,
            content: post.content,
            id: post._id
          };
        });
      }))
      .subscribe(transformedPosts => {
        this.posts = transformedPosts;
        this.postsUpdated.next([...this.posts]);
      });
  }
```

```
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, DELETE, OPTIONS"
  );
  next();
});

app.post("/api/posts", (req, res, next) => {
  const post = new Post({
    title: req.body.title,
    content: req.body.content
  });
  post.save();
  res.status(201).json({
    message: "Post added successfully"
  });
});

app.get("/api/posts", (req, res, next) => {
  Post.find().then(documents => {
    res.status(200).json({
      message: "Posts fetched successfully!",
      posts: documents
    });
  });
});
```

Slika 23: Primjer veze između Angular i Node.js (Izvor: vlastita izrada)

Na slici je vidljiv primjer koda za upravljanje stranicom koja prikazuje postove (lijevo) i kod Node.js poslužitelja (desno). Kod je napravljen prema predavanju Maximiliana Schwarzmüllera [32]. Glavnu vezu između Node.js i Angular-a predstavlja Angular paket HttpClient. Taj paket omogućava slanje zahtjeva na zadani URL koji može biti i relativan

projektu. Prije samog korištenja paketa, potrebno je uključiti u izvedbu unutar „app.module.ts“ datoteke, HttpClientModule i dodati u „imports“. Unutar konstruktora se zatim inicijalizira Http klijent te pozivanjem metoda primjerice get() i post() se šalju zahtjevi na rutu koja je definirana unutar Node.js-a. Na strani Node.js-a poziva se metoda za dohvaćanje podataka iz Mongo baze podataka i šalje se Angular-u u obliku json-a. Pozivanjem metode subscribe() na poslan zahtjev unutar Angular-a je nužno radi naznačavanja Angular-u da se žele primiti podaci i poslati sam zahtjev jer bez pozivanja te metode on se neće poslati. Ti se podaci onda mogu prikazati jednostavno dodjeljivanjem globalnim varijablama koje su definirane i prikazivanjem u HTML-u korištenjem jednih od Angular metoda. Ovo je jednostavan primjer komuniciranja okvira i poslužiteljskog dijela aplikacije.

# 7. Gotova aplikacija

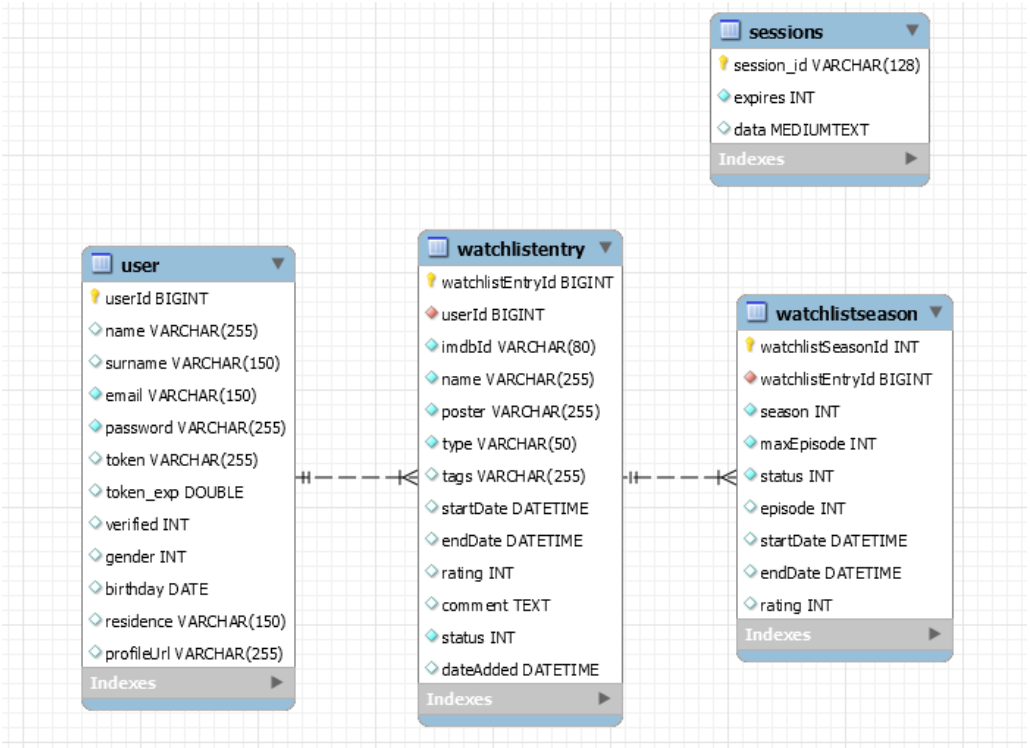
Na kraju, napravljena je funkcionalna aplikacija pomoću Node.js-a, Express-a i ostalih opisanih tehnologija u ovom radu. Dijelovi koda su također prikazani kao primjeri kroz rad. Aplikacija služi za praćenje serija i filmova, odnosno dodavanje u listu gledanja gdje je moguće pregledati sve serije i filmove koji su u nju dodani, sortirati, mijenjati. Moguće je i pretraživanje na svakom dijelu stranice, pregled statistike kao i vlastite kontrolne ploče, promjene postavki korisnika te pregled stranice o aplikaciji. Aplikacija je napravljena na engleskom jeziku i Windows operativnom sustavu uz pomoć Visual Studio Code-a. Izvorni kod aplikacije dostupan je na Github vlastitom repozitoriju na sljedećoj poveznici:

- Izvorni kod (Github): https://github.com/ItsNotACoffee/Showly

Za aplikaciju se koriste process.env varijable definirane unutar nodemon.json datoteke koja nije postavljena javno na Github-u radi senzitivnih podataka. Za postavljanje aplikacije na Web korišten je Heroku zajedno sa CrystalDB ekstenzijom koja omogućava besplatnu malu MySQL bazu podataka. Heroku je moguće spojiti na Github repozitorij, te je stoga aplikacija dostupna unutar „Environment“ taba na Github-u, klikom na „View deployment“, ukoliko je trenutno dostupna. Projekt se može klonirati i koristiti lokalno pri čemu je potrebno prvotno pokrenuti naredbu „npm install“ kako bi se instalirali svi paketi potrebni za aplikaciju. Nakon toga, potrebno je postaviti nodemon.json datoteku u korijenskom (eng. *root*) direktoriju koja sadrži potrebne „env“ varijable (potrebne varijable su izlistane unutar „Readme“ datoteke) sa vlastitim podacima. Također, potrebno je napraviti vlastitu MySQL bazu podataka temeljenu na strukturi ovog projekta koja će biti opisana u narednom poglavlju.

## 7.1. Struktura aplikacije

Za aplikaciju je korištena MySQL baza podataka koja sadrži ukupno tri povezane tablice i jednu nepovezanu tablicu koja služi za spremanje sesija.

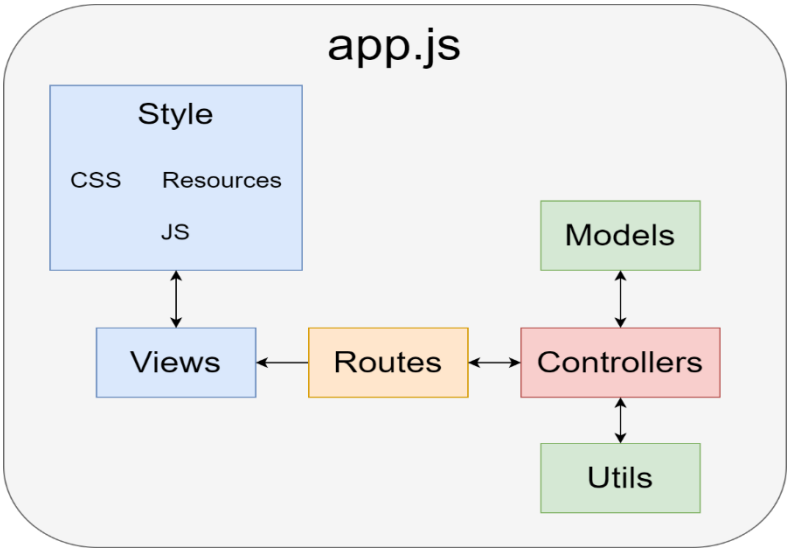



Slika 24: Shema aplikacije (Izvor: vlastita izrada)

Prva tablica, „user“, koristi se za spremanje korisnika te sadrži osnovne podatke kao što su email, ime, prezime, spol i sl. Polje za lozinku sadrži šifriranu lozinku putem bcryptjs paketa. Polja token i token_exp služe za spremanje tokena koji se koristi prilikom validacije korisnika i resetiranja lozinke. Polje profileUrl sadrži poveznicu do lokalno spremljene slike na strani poslužitelja koja se sprema pomoću Multer paketa. Tablica „user“ spaja se na „watchlistentry“ tablicu preko 1:M veze. Ta tablica služi za spremanje svih filmova i serija koje se dodaju u listu gledanja. Sama lista gledanja kao tablica ne postoji, već se dohvaća preko jedinstvenog userId-a, dakle jedan korisnik može imati više filmova ili serija dodanih koji se smatraju njegovom listom gledanja. Tablica „watchlistseason“ se veže na tablicu „watchlistentry“ također preko 1:M veze te služi za dodavanje pojedinih sezona određene serije u listu. Na taj način je moguće pratiti pojedine sezone određene serije. Određeni zapis u listi gledanja se dohvaća preko userId-a i imdbId-a čija je kombinacija jedinstvena. Zadnja tablica

se postavlja automatski od strane paketa express-mysql-session koji upravlja spremanjem sesija u bazu podataka.

Aplikacija je temeljena na MVC strukturi opisanoj ranije u ovom radu uz dodatak „util“ direktorija u kojem su spremljene neke od češće korištenih funkcija te „routes“ direktorija u kojemu su dodatno izdvojene rute od kontrolera.



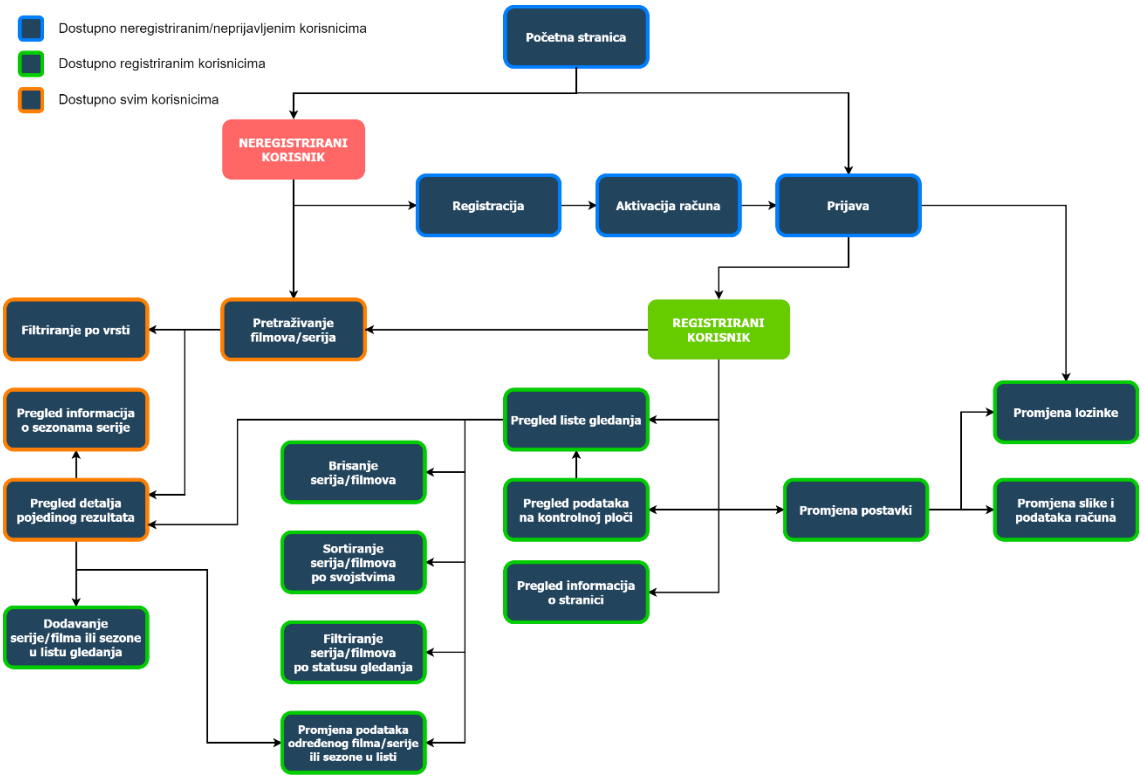

Slika 25: Struktura aplikacije (Izvor: vlastita izrada)

Sav kod obuhvaća datoteka app.js koja poziva sve potrebne rute, kontrolere i pakete. Modeli su zaduženi za komuniciranje s bazom podataka te prijenos tih podataka kontroleru koji to zahtijeva. Pogledi su dodatno definirani sa „style“ direktorijem koji sadrži css datoteke, statične resurse poput slika te JavaScript datotekom koja sadrži kod za upravljanje nekoliko elemenata na klijentskoj strani Web aplikacije.

## 7.2. Funkcionalnost aplikacije

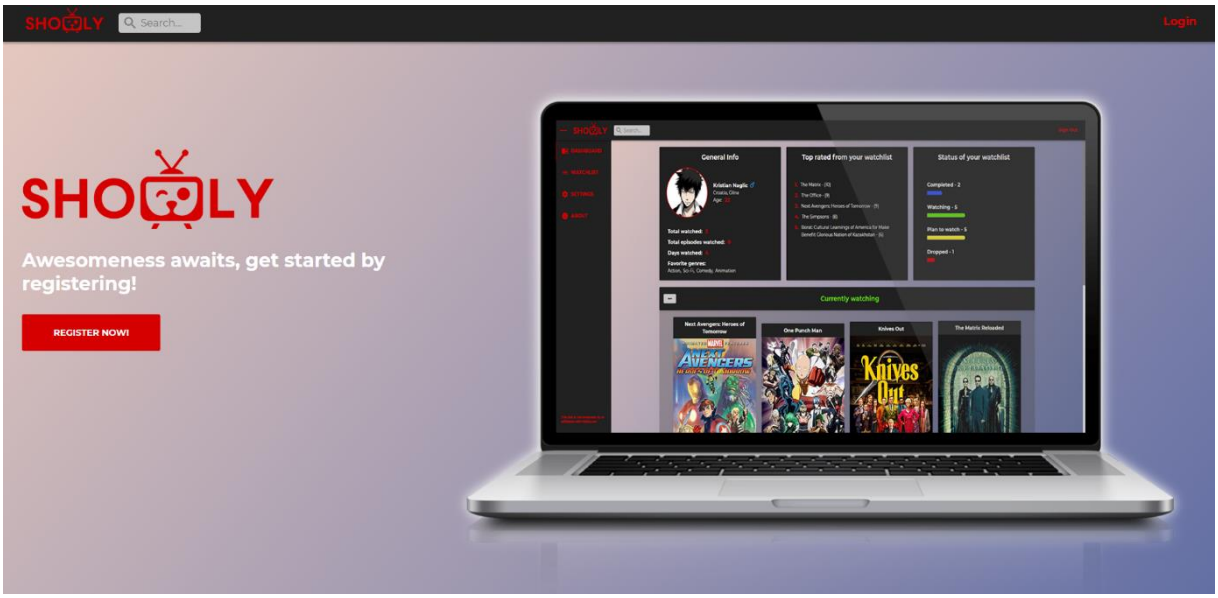
U aplikaciji postoje dvije uloge korisnika, neregistrirani i registrirani korisnici. Neregistrirani korisnici, što uključuje i neprijavljene korisnike, imaju mogućnost prijave ili registracije uz aktivaciju računa. Obje uloge korisnika mogu pretraživati filmove putem trake za pretraživanje na vrhu stranice i pregledavati rezultate pretraživanja uz filtriranje po tipu (serija ili film). Za svaki rezultat je moguć pregled detalja za određeni film ili seriju uz pregled informacija o sezonama ukoliko se radi o seriji. Registrirani korisnici, ukoliko su prijavljeni, imaju dodatnu mogućnost promjene podataka pojedine serije ili filma, ili pojedine sezone, te dodavanja u listu gledanja. Registrirani korisnici imaju nekoliko opcija. Prva je pregled

podataka na kontrolnoj ploči uz mogućnost odlaska na listu gledanja direktno sa ploče. Zatim, pregled liste gledanja koja sadrži dodane filmove i serije zajedno sa sezonama i mogućnostima filtriranja po statusu gledanja i sortiranja po svojstvima. Iz liste je moguće obrisati pojedine filmove ili serije, mijenjati podatke i pregledati detalje. Registrirani korisnik ima i mogućnost promjene postavki računa uz promjenu slike prikazane na kontrolnoj ploči te mogućnost resetiranja lozinke. Zadnja dostupna poveznica je stranica o aplikaciji gdje su dostupne informacije o aplikaciji, autoru i korištenim modulima. Na slici niže je prikazan navigacijski dijagram koji prikazuje sve moguće radnje koju su opisane, a dalje će se one detaljnije prikazati uz odgovarajuće slike.




Slika 26: Navigacijski dijagram aplikacije (Izvor: vlastita izrada)

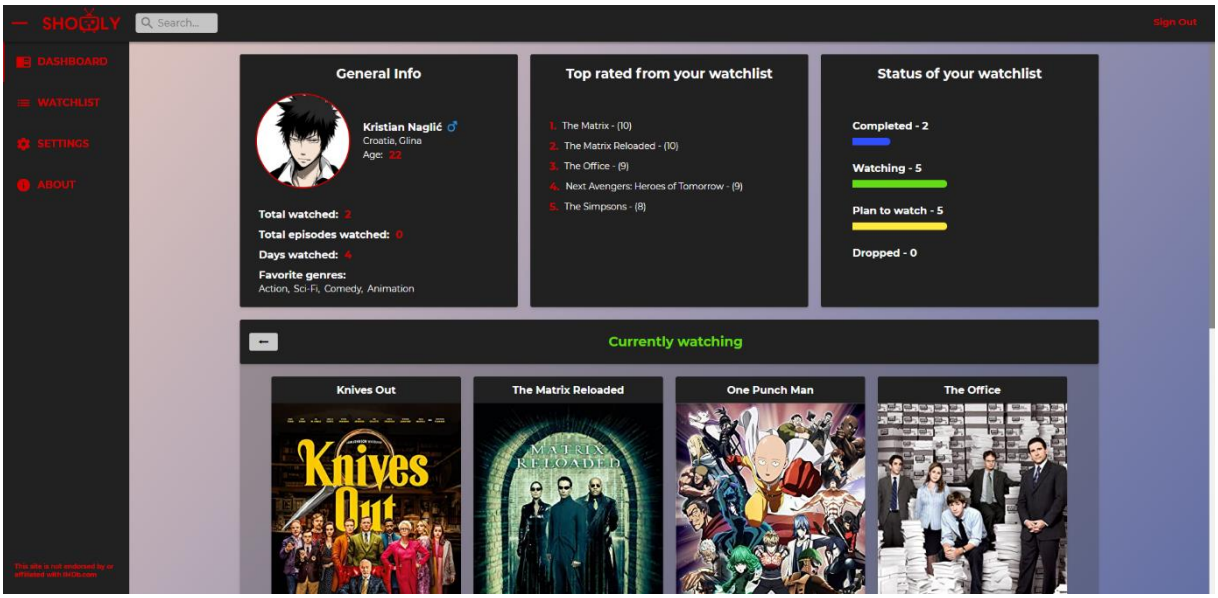
Aplikacija je kompatibilna i sa mobilnim uređajima jer koristi grid i fleksibilan dizajn. CSS je pisan samostalno te nije korišten Bootstrap. Početna stranica je dostupna samo korisnicima koji nisu prijavljeni u sustav te je vrlo jednostavnog dizajna.




Slika 27: Aplikacija – početna stranica (Izvor: vlastita izrada)

Nakon prijavljivanja u sustav, na početnoj stranici korisnik ima uvid u statistiku vezanu za njegovu listu gledanja kao i dvije sekcije koje mu preporučuju koje filmova ili serije može pogledati sljedeće. Statistika se računa i šalje od strane kontrolera temeljeno na dobavljenim podacima iz baze podataka, a kartice za preporuku uzimaju četiri nasumična zapisa iz korisnikove liste gledanja sa određenim statusom.

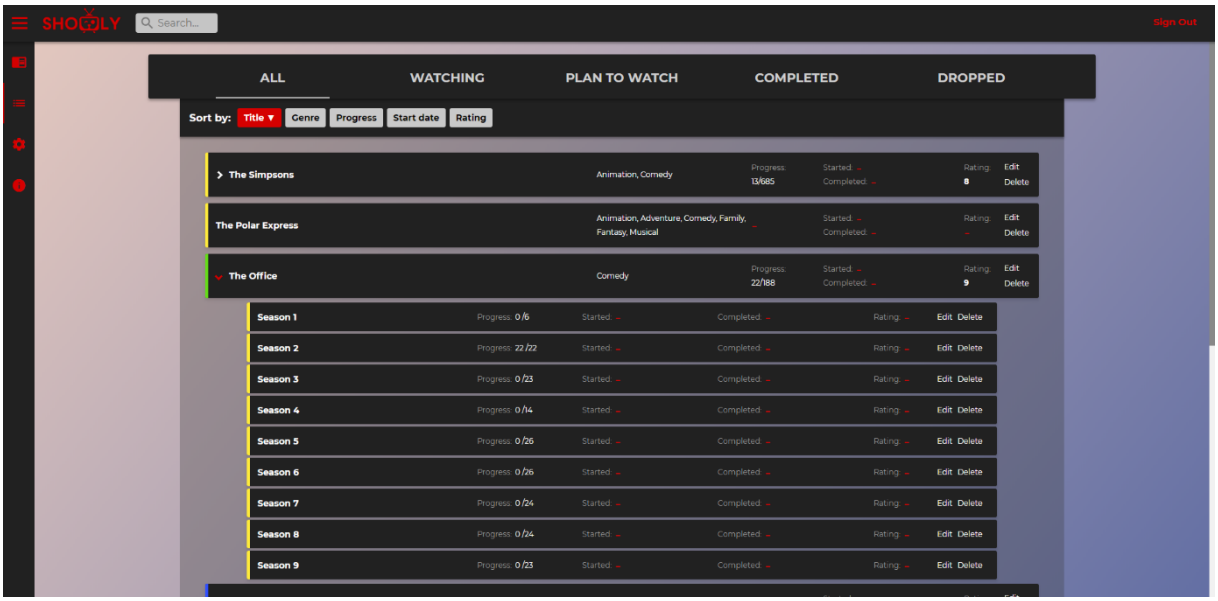
Na lijevoj strani se nalazi dinamičan meni koji sadržava sve moguće opcije. Izgled i ponašanje menija je postignuto čisto uporabom JavaScript-a na strani klijenta i mijenjanja CSS klasa po potrebi.




Slika 28: Aplikacija – kontrolna ploča (Izvor: vlastita izrada)

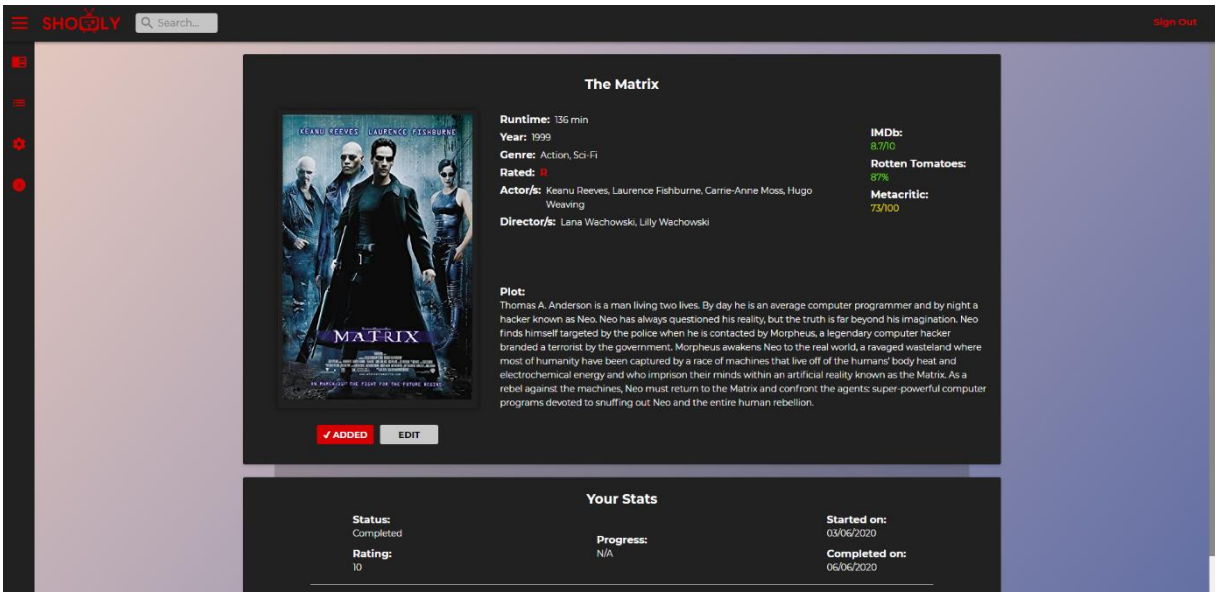
Ukoliko korisnik odabere listu gledanja, može pregledati sve zapise koje ima dodane u listu te sortirati po određenim svojstvima. Gornja traka omogućava filtriranje po statusu gledanja, a s desne strane se dinamično pojavljuje straničenje ukoliko korisnik ima više od 30 zapisa u listi. Filtriranje je postignuto jednostavno postavljanjem „hidden input“ polja unutar HTML-a prema čijim vrijednostima se šalje GET zahtjev sa zadanim uvjetima. Prije samog slanja odgovora, kontroler natrag šalje i nove vrijednosti varijabli koje se dinamično upisuju u HTML kako bi se zadržalo svojstvo sortiranja prilikom straničenja ili filtriranja statusa. Uz to, svaka forma je odvojena kako bi imala vlastite „hidden“ elementa sa vrijednostima.

Straničenje u listi gledanja, kao i u pretraživanju, se računa unutar samog HTML dokumenta putem EJS-a i poslanih varijabli za maksimalan broj zapisa i trenutnu stranicu. Kontrole za promjenu stranice se generiraju putem jednostavne for petlje.



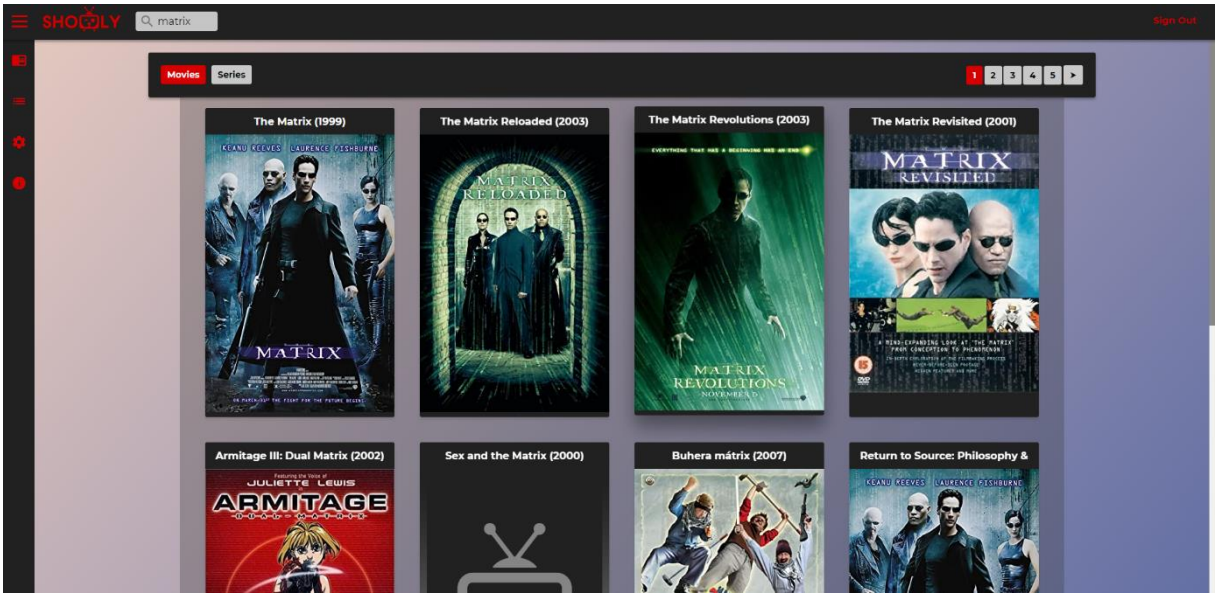

Slika 29: Aplikacija – lista gledanja (Izvor: vlastita izrada)

Svaki od zapisa prikazuje određene podatke koje je korisnik postavio, te akcije brisanja i mijenjanja podataka zapisa. Ukoliko se radi o seriji, pojavljuje se strelica koja otvara padajući meni koji prikazuje dodatno pojedine sezone dodane u listi.



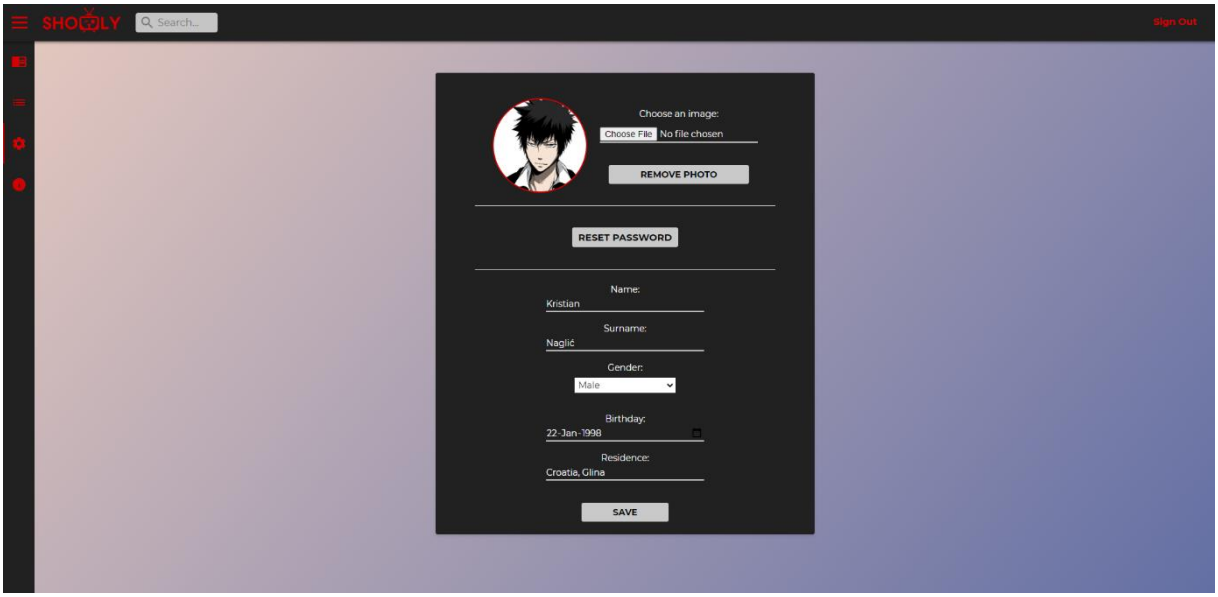

Slika 30: Aplikacija – detalji o zapisu (Izvor: vlastita izrada)

Klikom na naziv unutar liste gledanja ili na karticu preko pretraživanja se otvara prozor za detalje o pojedinom filmu ili seriji. Podaci se popunjavaju iz objekata dohvaćenih putem modela i EJS paketom koji dinamično popunjava poglede. Tu je moguće dodati zapis u listu i direktno mijenjati podatke o zapisu u listi. Dodatno, se ispod prikazuje kartica sa podacima o zapisu korisnika te ukoliko se radi o seriji, kartica sa tablicom i pregledom svih sezona serije od kojih se pojedine mogu dodati u listu. Ako se dodaje serija direktno, automatski se dodaju sve sezone te serije, no ako se dodaje sezona pojedinačno, dodaje se i sama serija u listu gledanja zajedno sa odabranom sezonom.




Slika 31: Aplikacija – pretraživanje (Izvor: vlastita izrada)

Pretraživanju je moguće pristupiti sa bilo koje stranice dostupne prijavljenom korisniku te prikazuje listu pronađenih rezultata u obliku kartica koje je moguće kliknuti da bi se otvorili detalji. Moguće je kretanje kroz stranice rezultata te, u lijevom kutu rezultata, filtriranje po seriji ili filmu.




Slika 32: Aplikacija – postavke korisnika (Izvor: vlastita izrada)

Korisnik ima mogućnost promjene svojih podataka putem postavki. Sve forme za brisanje i promjenu podataka kontroliraju se unutar kontrolera sa POST i GET metodama. Prilikom otvaranja, otvara se forma gdje je moguće postaviti podatke kao i profilnu sliku koja je vidljiva na kontrolnoj ploči. Gumb za brisanje slike pojavljuje se tek kada je postavljena slika, i nestaje ukoliko je izbrisana. Tu se jednostavno koristi EJS if else svojstvo kako bi se generirao gumb po uvjetu lokalne varijable. Sam prijenos slike na poslužitelj kontrolira Multer paket, a poveznica na sliku se sprema u bazu podataka. Također, postoji mogućnost resetiranja lozinke koja otvara novi prozor gdje je moguće upisati email adresu na koju će biti poslana poveznica sa tokenom za resetiranje lozinke tog korisnika. Slanje email-a moguće je putem modula Nodemailer koji je postavljen na korištenje Gmail servisa i vlastitog računa.

Na stranici o aplikaciji se nalaze statični podaci koji prikazuju opis aplikacije, informacije o autoru i paketima koji su korišteni zajedno sa vrstom licence te poveznicom ukoliko je potrebna.

# 8. Zaključak

Cilj rada je bio pokazati koliko je Node.js jednostavan za korištenje te u isto vrijeme vrlo ekstenzivan i iznimno moćan programski jezik. Uz to, može se vidjeti da je brži od većine ostalih tehnologija i da je pisanje strane poslužitelja i strane klijenta gotovo identično jer se temelji na JavaScript-u. Dodaci poput Express-a i body-parsera, koji su opisani u ovom radu, čine ga još lakšim za uporabu. Vidljivo je da MVC struktura igra veliku ulogu kod definiranja ne samo Node.js projekata, već se smatra dobrom praksom i kod programiranja općenito. Prije svega, prikazano je instaliranje i pokretanje jednostavnog Node.js poslužitelja iz čega se može zaključiti da takva radnja traje svega desetak minuta. Opisano je i nekoliko radnji mogućih sa Node.js-om kao što je rad sa podacima iz obrazaca, gdje su prikazani primjeri općenitog korištenja kao i većine funkcija koje su korištene u izradi same aplikacije. Pokazani su i načini obrade pogrešaka za koje se smatra da nisu pravila već ih je moguće obrađivati, odnosno rješavati na način koji najbolje odgovara korisniku i projektu na kojemu isti radi. Ono po čemu se Node.js ističe od ostalih tehnologija nije čak ni njegova brzina ili asinkron rad, već mogućnost korištenja raznih paketa koji olakšavaju rad. Sve radnje koje su opisane kroz ovaj rad, poput generiranja dinamičnog sadržaja ili rad sa formama, su moguće i bez dodatnih paketa jer su i oni pisani u Node.js-u odnosno JavaScript-u, no njihovo korištenje sprječava vraćanje osnovama i ponovo izmišljanje nečega što već postoji.

Na kraju rada ukratko je opisana MEAN platforma gdje se ponajviše fokusiralo na vezu između komponenti i kako se ona postiže umjesto njihove sintakse. Može se reći da je MEAN platforma jedan od rezultata odabira tehnologija koje najbolje rade zajedno i upotpunjuju si mogućnosti međusobno. U izradi aplikacije koja je na kraju rada i opisana, korištena je vrlo jednostavna kombinacija tehnologija i paketa kako bi se što bolje prikazao rad sa Node.js-om bez orijentacije na tehnologije klijentske strane. Kod uporabe Node.js-a, brojne prednosti nadmašuju mane, a lakoća izrade i jednostavna sintaksa omogućuju izradu funkcionalnih i brzih aplikacija u kratkom vremenskom roku.

# Popis literature


[1] MDN contributors, „Getting started with HTML“, 2020. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started [pristupano 05.07.2020.].

[2] MDN contributors, „What is CSS?“, 2020. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS [pristupano 05.07.2020.].

[3] MDN contributors, „JavaScript“, 2020. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript [pristupano 04.07.2020.].

[4] MDN contributors, „DOM (Document Object Model)“, 2020. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Glossary/DOM [pristupano 06.06.2020.].

[5] Jason Orendorff, „ES6 In Depth: let and const“, 2015. [Na internetu]. Dostupno: https://hacks.mozilla.org/2015/07/es6-in-depth-let-and-const/ [pristupano 04.07.2020.]

[6] Priyesh Patel, „What exactly is Node.js?“. 2018. [Na internetu]. Dostupno: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/ [pristupano 04.07.2020.].

[7] OpenJS Foundation, „Blocking vs Non-Blocking“, (bez dat.). [Na internetu]. Dostupno: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/ [pristupano 06.06.2020.].

[8] W3Schools, „What is npm?“, (bez dat.). [Na internetu]. Dostupno: https://www.w3schools.com/whatis/whatis_npm.asp [pristupano 06.06.2020.].

[9] TutorialsTeacher, „Express.js“, (bez dat.). [Na internetu]. Dostupno: https://www.tutorialsteacher.com/nodejs/expressjs [pristupano 06.06.2020.].

[10] Matthew Eernisse, „EJS“, (bez dat.). [Na internetu]. Dostupno: https://ejs.co/ [pristupano 07.06.2020.].

[11] „imdb-api“ (03.08.2019.). [Na internetu]. Dostupno: https://www.npmjs.com/package/imdb-api [pristupano 07.06.2020.].

[12] Remy Sharp, „nodemon“, 2020. [Na internetu]. Dostupno: https://www.npmjs.com/package/nodemon [pristupano 07.06.2020.].

[13] Douglas Wilson, „body-parser“, 2019. [Na internetu]. Dostupno: https://www.npmjs.com/package/body-parser [pristupano 07.06.2020.].

[14] Daniel Wirtz, „bcryptjs“, 2017. [Na internetu]. Dostupno: https://www.npmjs.com/package/bcryptjs [pristupano 07.06.2020.].

[15] Douglas Wilson, „express-session“, 2020. [Na internetu]. Dostupno: https://www.npmjs.com/package/express-session [pristupano 07.06.2020.].

[16] Charles Hill, „express-mysql-session“, 2020. [Na internetu]. Dostupno: https://www.npmjs.com/package/express-mysql-session [pristupano 07.06.2020.].

[17] Andris Reinman, „nodemailer“, (bez dat.). [Na internetu]. Dostupno: https://nodemailer.com/about/ [pristupano 07.06.2020.].

[18] „moment.js“, (bez dat.) [Na internetu]. Dostupno: https://momentjs.com/docs/ [pristupano 07.06.2020.].

[19] Hage Yaapa, Jaret Pfluger, Linus Unnebäck, „multer“, 2019. [Na internetu]. Dostupno: https://www.npmjs.com/package/multer [pristupano 07.06.2020.].

[20] Jared Hanson, „connect-flash“, 2013. [Na internetu]. Dostupno: https://www.npmjs.com/package/connect-flash [pristupano 07.06.2020.].

[21] Oracle Corporation, „What is MySQL?“, (bez dat.). [Na internetu]. Dostupno: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html [pristupano 06.06.2020.].

[22] Guru99, „SQL vs NoSQL: What's the difference?“, (bez dat.). [Na internetu]. Dostupno: https://www.guru99.com/sql-vs-nosql.html [pristupano 06.06.2020.].

[23] Andrey Sidorov, „mysql2“, 2019. [Na internetu]. Dostupno: https://www.npmjs.com/package/mysql2 [pristupano 06.06.2020.].

[24] Meghan Nelson, „An Intro to Git and GitHub for Beginners“, 2015. [Na internetu]. Dostupno: https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners [pristupano 06.07.2020.].

[25] „Node.js v14.4.0 Documentation“, (bez dat.). [Na internetu]. Dostupno: https://nodejs.org/dist/latest-v14.x/docs/api/synopsis.html [pristupano 07.06.2020.].

[26] Maximilian Schwarzmüller, „NodeJS - The Complete Guide (incl. MVC, REST APIs, GraphQL)“ 2020. [Na internetu]. Dostupno: https://www.udemy.com/course/nodejs-the-complete-guide/ [pristupano 14.05.2020.].

[27] Express, „Express.js Guide“, (bez dat.). [Na internetu]. Dostupno: http://expressjs.com/en/guide/writing-middleware.html [pristupano 16.05.2020.].

[28] „Post-Redirect-Get“, (bez dat.) u *RyteWiki.* Dostupno: https://en.ryte.com/wiki/Post-Redirect-Get [pristupano 09.06.2020.].

[29] Codecademy, „MVC: Model, View, Controller“, (bez dat.). [Na internetu]. Dostupno: https://www.codecademy.com/articles/mvc [pristupano 10.06.2020.].

[30] Oleg Romanyuk, „NodeJS vs Python: How to Choose the Best Technology to Develop Your Web App's Back End“, (14.01.2020.). [Na internetu]. Dostupno: https://www.freecodecamp.org/news/nodejs-vs-python-choosing-the-best-technology-to-develop-back-end-of-your-web-app/ [pristupano 12.06.2020.].

[31] Swapnil Banga, „PHP vs Node.js: Differences you need to Know“, (09.04.2020.). [Na internetu]. Dostupno: https://hackr.io/blog/php-vs-node-js [pristupano 12.06.2020.].

[32] Maximilian Schwarzmüller, „Angular & NodeJS - The MEAN Stack Guide“ 2020. [Na internetu]. Dostupno: https://www.udemy.com/course/angular-2-and-nodejs-the-practical-guide/ [pristupano 06.05.2020.].

# Popis slika