Razvoj web aplikacija na MEAN platformi

Petrović, Mario

Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij Poslovna informatika

Mario Petrović

Razvoj web aplikacija na MEAN platformi

Diplomski rad

Mentor: prof. dr. sc. Mario Radovan

Rijeka, prosinac 2018.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 20.12.2017.

Zadatak za diplomski rad

Pristupnik: Mario Petrović

Naziv diplomskog rada: Razvoj web aplikacija na MEAN platformi

Naziv diplomskog rada na eng. jeziku: Developing web applications using the MEAN stack

Sadržaj zadatka:

Operativni sustav Linux, Web poslužitelj Apache, sustav za upravljanje bazama podataka MySQL i jezik PHP čine tzv. LAMP platformu koju koristi najveći broj web sjedišta. Međutim, posljednjih godina najbrže raste tzv. MEAN platforma, koja se sastoji od noSQL baze podataka MongoDB, JavaScript platformi Express.js i Angular.js, te Node.js poslužitelja. U diplomskom radu student će dati pregled navedenih tehnologija, te opisati kako svaka od njih funkcionira zasebno te kao dio MEAN cjeline. Na primjeru izrade jedne ogledne aplikacije opisati će se postupak razvoja web aplikacija na MEAN platformi. Nakon toga usporediti će se performanse dvaju istovjetnih oglednih aplikacija izrađenih na LAMP i MEAN platformi te dati analiza rezultata.

Mentor:

prof. dr. sc. Mario Radovan

_____

Voditelj za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović

_____

Zadatak preuzet:

_____

(potpis pristupnika)

Sadržaj

Popis tablica i slika

Sažetak

MEAN *stack* tj. platforma objedinjuje baze podataka MongoDB, JavaScript platforme Express.js i Angular.js, te Node.js poslužitelja. Omogućuje jednostavnu i lakšu izradu *web* tj. mrežnih aplikacija korištenjem samo jednog programskog jezika JavaScript. Novija i modernija tehnologija privlačna je programerima obzirom da iziskuje poznavanje samo jednog programskog jezika te sam krajnji proizvod je elegantna, brza i vizualno moćna aplikacija, ukoliko se tako želi izraditi. Najveći konkurent MEAN platformi je LAMP platforma koja je danas rekorder u broju aktivnih *web* stranica. Koristi kao poslužitelja Apache te PHP jezik za serverski dio, a za klijentski dio koristi XML (engl. *Extensible Markup Language*, skraćeno XML), HTML (engl. *Hypertext Markup Language*, skraćeno HTML) te CSS (engl. *Cascading Style Sheets*, skraćeno CSS) te po potrebi i JavaScript se može dodati kako bi klijentski dio komunicirao sa serverom. Ova tehnologija korištena na operacijskom sustavu Linux zaokružuje samu definiciju platforme.

Budući da sam savladao LAMP tehnologiju odlučio sam proširiti znanje MEAN platformom te ću kroz rad prvenstveno pokušati približiti JavaScript jezik čitatelju te objasniti svaku komponentu MEAN platforme. Na primjeru izrade aplikacije „Moja knjiga“ približiti ću tehnologije čitatelju i objasniti sam postupak korištenja tehnologije na primjeru. Aplikaciju će biti izrađena i u LAMP tehnologiji te na kraju rada će biti odrađeno „A/B“ testiranje koje će prikazati prednosti i nedostatke pojedine platforme na izrađenom primjeru.

Aplikacija „Moja knjiga“ je *web* aplikacija te mini društvena mreža gdje se svaki registrirani korisnik ima mogućnost unos te kreiranje liste najdražih knjiga za svaki pojedini mjesec korištenja aplikacije. Knjige rangira na način da im da ocjenu od 1 do 10 gdje je 10 najviša i najbolja ocjena. Također za svaku dodanu knjigu može dodati komentar. Za knjige dodane od strane drugih korisnika može dati ocjenu i komentar. Budući da se ocjenjuju i knjige drugih korisnika moguće je pogledati ukupnu mjesečnu rang listu te listu po grupama korisnika ovisno o tome koju grupu korisnik odabere prilikom registracije.

1. Uvod

U današnje vrijeme nezamisliv je život bez interneta kao komunikacijskog i informativnog sredstva u društvu. Kako bi na što bolji način razmjenjivali informacije, podatke internetske ili web stranice razvijale su se kroz nekoliko desetljeća te svakim razvojem poboljšavaju kvalitetu, sigurnost i povećavaju brzinu njihovog korištenja. Kako su napredovale tako su postajale sve složeniji sustavi te sama tehnologija na kojoj su bazirane se razvijala u skladu s potrebama. Danas najčešći primjer web aplikacije razvijan je s LAMP tehnologijom gdje se koriste svi dinamički elementi weba kao što su HTML, CSS i JavaScript. Sam pozadinski ili serverski dio najčešće je pisan PHP skriptnim jezikom. Objedinjeno na Linux operacijskim sustavima i s Apache HTTP poslužiteljem tj. serverom i MySQL (engl. *My Structured Query Language*, skraćeno MySQL) bazama podataka čine LAMP platformu. Za programiranje i izradu web aplikacija danas prednjače Linux operacijski sustavi kako je na njima i kreirana većina današnje tehnologije te kako su besplatni i otvorenog koda. Kako je dostupan svima proširio se utjecaj Linuxa na razvoj ovih tehnologija. Apache HTTP Server je također otvorenog koda te je sposoban raditi na svim poznatijim operacijskim sustavima te je danas najkorišteniji web poslužitelj uz Nginx. MySQL je također sustav otvorenog koda koji služi za upravljanjem nad bazom podataka, u ovom slučaju za web aplikacije. Distribuira se kao sastavni dio serverskih Linux distribucija no podržan je i na ostalim Linux, Mac Os i Windows distribucijama. PHP je danas najkorišteniji jezik uz JavaScript za programiranje web aplikacija. Kako postoje mnogi programski okviri PHP-a za programiranje koji sadrže strukturu za kod te mnoge funkcije koje pomažu u izradi i ubrzavaju sam proces stvaranja web aplikacija još uvijek dobro kotira i održava se na tržištu.

Budući da je JavaScript jezik podržan u svim poznatijim web preglednicima i njegovim razvojem te razvojem JavaScript programskih okvira postao je sastavni dio svake ozbiljnije web aplikacije. Širokih mogućnosti i fleksibilnosti postao je sve zastupljeniji te se s time razvila ideja da sam preuzme i serverski dio aplikacije kako bi se olakšalo programiranje i smanjio broj jezika i tehnologija pri samoj izradi web aplikacije. Brzim razvojem stvorena je MEAN platforma koja objedinjuje MongoDB, noSQL baze podataka, platformi Express.js i Angular.js te Node.js poslužitelja. Korištenjem samo jednog programskog jezika za razliku od LAMP platforme gdje je

nužno poznavanje programiranja s PHP i JavaScript jezikom olakšano je programiranje i proširene su mogućnosti i opcije svake web aplikacije koja je rađena s ovom tehnologijom. Najvažnija značajka ove MEAN platforme jest *real-time,* tj. izvođenje u realnom vremenu, komponenta gdje korisnik pravovremeno komunicira sa serverom te i na taj način dobiva povratne informacije koje su skladištene na serveru.

Velik broj web programera nisu ljubitelji JavaScript-a obzirom da ima neke svoje posebnosti no svakako treba biti u korak s vremenom i razvojem tehnologija koje pružaju bolje i jače mogućnosti u odnosu na postojeće i u tom trenutku najzastupljenije.

2. LAMP platforma

LAMP platformu sačinjava skup tehnologija Linux, Apache, MySQL i PHP. Linux je operacijski sustav s mnogim distribucijama što serverskih što neserverskih *desktop* distribucija za svakodnevno korištenje i programiranje. Ovaj operacijski sustav je najjednostavniji za korištenje pri web programiranju kako je najveći broj tehnologija koje su nužne i vezane za izradu web stranica podržane i izrađivane na ovom sustavu. Otvorenog je koda te besplatan što je utjecalo na opseg korištenja u ove svrhe. Sama konfiguracija tehnologija za web programiranje je jednostavnija na Linux sustavima i mogu se instalirati samo s nekoliko komandi u terminalu. Na Windowsima je malo drukčija priča gdje je često nužan nekakav program koji će omogućiti kompatibilnost i podržanost poput XAMPP-a za Apache.

Apache je HTTP poslužitelj koji omogućuje izvršavanje samog programskog koda. Podržava jezike Perl, Python, Tcl i PHP. Kod pisan u PHP izvršiti će Apache i vratiti podatke koji su zatraženi od strane klijenta. Trenutno je najkorišteniji HTTP (engl. *Hypertext Transfer Protocol* , skraćeno HTTP) server u svijetu. [1]

MySQL je sustav za upravljanjem nad bazom relacijskom podataka. On je višenitni i višekorisnički SQL sustav te pomoću njega možemo upravljati relacijskom bazom podataka pomoću naredbi i upita. Danas su neki od najpoznatijih MariaDB, PostgreSQL i MongoDB koji je NoSQL. [2]

PHP je skriptni objektni programski jezik kojemu je osnova C. Najčešće je korišteni jezik za web programiranje te su mnoge poznate web aplikacije pisane ovim jezikom kao što je npr. Facebook. Jezik je poprilično jednostavan za naučiti, a razvojem obrazaca ili programskog okvira olakšano je i pojednostavljeno programiranje. Vrlo često se koristi u kombinaciji s JavaScript-om u današnje vrijeme. Nedostatak ovoj programskoj jezika u odnosu na JavaSript je što je on samo na poslužiteljskoj strani. PHP kod može biti ugrađen u HTML ili se može pozivati vanjska PHP skripta. Rezultat nedostatka je situacija gdje se web stranica jednom učita i PHP kod izvrši podaci bez ponovnog učitavanja stranice neće se ažurirati. Jedino ukoliko smo uključili AJAX (engl. *Asynchronous JavaScript And XML*, skraćeno AJAX) JavaScript-a. Izrazito je razvijen programski jezik te trenutna zadnja verzija je PHP7.2. PHP ima sposobnost

izvršavanja iz komandne linije te pozivom preko PHP naredbe može se izvršiti set PHP naredbi.

3. JavaScript

JavaScript je objektni skriptni jezik koji se koristi kod izrade dinamičnih web stranica tj. aplikacija te mobilnih aplikacija. Osnovni dijelovi weba su HTML koji je osnovni kod pomoću kojeg oblikujemo i uređujemo elemente koji sačinjavaju cijelu web stranicu. Osnovni primjeri su tekstovi, tablice, forme i sl. Kako bismo definirali i odredili vizualni izgled elemenata koristi se CSS dok nam JavaScript omogućuje da upravljamo ponašanjem tih elemenata. Trenutno je najkorišteniji jezik te najtraženiji na tržištu rada. Veliki razvoj i mogućnost svih web preglednika da ga izvršavaju potpomogli su JavaScript-u da se probije na vrh u odnosu na druge jezike. JavaScript je skriptni interpretirani jezik više razine te ga je razvila tvrtka Netscape. Nije klasičan objektno orijentirani jezik već se temelji na prototipu. Prvenstveno se izvršavao samo u web preglednicima te pomoću AJAX-a mogao komunicirati sa serverom. Danas je potpuno druga priča gdje pomoću Node.js izvršava na serveru. Na taj način olakšano je programiranje kako je moguće raditi samo s jednim jezikom za razliku gdje se prije pojave Node.js nužno koristio najčešće PHP ili neki drugi jezici poput Java-e, C#-a. [3] [4]

Izvršavanje JavaScript-a u web preglednicima omogućava dodavanje novog HTML-a, promjena postojećeg te također isto za CSS pomoću kojeg se izmjenjuje i definira vizualni prikaz tj. izgled web stranice. Reagira na akcije korisnika kao što su klik mišem, pomicanje pokazivača, pritisak tipke. Također kao što je već navedeno preko AJAX-a može komunicirati sa serverom, slati mu zahtjeve kako bi se dobio odgovor, podizanje ili *upload* te dohvat ili *download* podataka. Radi kvalitetnijeg korištenja nekog web sjedišta JavaScript ima mogućnost spremanja kolačića (engl. *cookies*). To su podaci koji se lokalno spremaju na strani korisnika kao što su npr. podaci za prijavu, zamjena za sesijske podatke koji bilježe npr. da je korisnik prijavljen što uvelike rasterećuje servere pogotovo za velike društvene mreže. Već s ovakvim mogućnostima JavaScript je počeo zasjenjivati ostale programske jezike no sa serverskom stranom s vremenom će početi zamjenjivati ih. Kako su mnoge velike i poznate aplikacije poput Facebooka pisane u PHP-u on je još uvijek uvelike prisutan, te također zbog znanja programera gdje su mnogi naviknuli na PHP i njegove mogućnosti pogotovo u radu s programskim okvirima poput Laravela, Symfony-ja. Što se tiče njegovih mogućnosti na lokalnoj klijent strani ne može čitati, pisati i kopirati tj.

kreirati datoteke niti pokretati programe jer nema direktan pristup funkcijama operacijskog sustava što je dobro radi zaštite korisnika te zloupotrebe. Samo dodavanje datoteka je limitirano s učitavanjem tj. dodavanjem istih. Od *hardvera* može koristiti web kameru i mikrofon uz eksplicitnu dozvolu korisnika. [5]

Kako je JavaScript potpuno integriran s neizostavnim komponentama svakog web sjedišta HTML/CSS te je vrlo jednostavan interpretiran skriptni jezik više razine te podržan u svim najkorištenijim web preglednicima čini ga već neizostavnim dijelom svakog dinamičkog web sjedišta.

3.1. *Server side* JavaScript

1995. godine Netscape je izdao verziju JavaScripta za web preglednike te napravio uvod u implementaciju jezika za serversku stranu izvršavanja kao „Netscape Enterprise Server". Od 1996. IIS web server podržavao je Microsoftovu implementaciju JavaScripta kao jezika na serverskoj strani za ASP i .NET web sjedišta. 2000-ih su predstavljene serverske inačice jezika između kojih je 2009. Node.js kao predvodnik. [6]

3.2. Sintaksa JavaScript-a

Polazište korištenja svakog programskog jezika je rad s varijablama. Varijable definiramo s „var" ili „const" ovisno o tome definiramo li varijablu ili konstantu. Prije korištenja bilo koje varijable potrebno ju je inicijalizirati bilo da joj dodamo neku vrijednost na način da ju definiramo ili samo inicijaliziramo bez ikakve vrijednosti. Varijablu inicijaliziramo kroz sintaksu „var x;" gdje je x ime varijable, a znak „;" je nužan za korištenje kao znak za kraj naredbe. Ukoliko želimo dodati vrijednost varijabli x 5 to ćemo učiniti na sljedeći način: var x = 5; Osim numeričke vrijednosti možemo varijablu definirati i kao tip *string* tj. tekstualno-znakovni tip. var x = „Ovo je tekst!". Osnova korištenja svakog programskoj jezika je ispis nekakvog teksta. To ćemo u JavaScript-u izvršiti na sljedeći način pomoću konzole objekta sintaksom: „console.log („Tekst koji će se ispisati na ekranu");". [7]

Funkcije kao i u drugim jezicima mogu biti privatne, javne i zaštićene funkcije ovisno o tome kako ih želimo definirati. U JavaScript-u postoje dvije vrste opsega ili razine programskog koda. Lokalni i globalni opseg. Svaka funkcija stvara novi opseg. Opseg definira dostupnost varijabli iz funkcija. Varijable koje su definirane unutar funkcije su lokalne te one imaju lokalni opseg i njima se može pristupiti samo unutar funkcije. Kako su lokalne varijable dostupne samo unutar njihovih funkcija varijable s istim nazivom mogu se koristiti u različitim funkcijama. One se stvaraju kada započne izvršavanje dijela koda koji čini funkcija i brišu se kada se izvođenje funkcije završi. Primjer tj. sintaksu jednostavnije funkcije definiramo na sljedeći način:

```
1. function primjerFunkcije() {
2.     var primjerVarijabla = “tekst”;
3. }
```

Slika 1 primjer JavaScript funkcije

U tijelu funkcije koje je ograđeno vitičastim zagradama možemo koristiti našu definiranu varijablu. Varijable koje su definirane izvan funkcije su globalne. Njima se može pristupiti iz drugih funkcija te se mogu koristiti. Globalne varijable mogu se napraviti lokalno (privatno) s zatvaranjem. Zatvaranje (engl. *Closure*) je unutarnja funkcija koja ima pristup varijabli-opsega lanca vanjske zatvorene funkcije. Zatvaranje ima tri lanca opsega. Prvi ima pristup vlastitom opsegu. To su varijable definirane u tijelu funkcije. Imaju pristup varijablama vanjske funkcije i imaju pristup globalnim varijablama. Unutarnja funkcija ima pristup ne samo varijablama vanjske funkcije već i parametrima vanjske funkcije. Unutarnja funkcija ne može pozvati objekt argumenata vanjske funkcije, iako može izravno pozvati parametre vanjske funkcije. „This“ funkcija je često korištena funkcija i neophodna za programiranje. Ključna riječ ove funkcije ponaša se drugačije u JavaScript-u za razliku u drugim jezicima. Ona također ima neke razlike između strogog načina rada i nepristupačnog načina rada. U većini slučajeva, vrijednost toga određuje se funkcijom pozivanja. Nije moguće odrediti dodjelom tijekom izvršenja, a može se razlikovati svaki put kada se poziva funkcija. Uvedena je metoda vezanja kako bi se postavila vrijednost funkcije, bez obzira na to kako se zove, a uvedene su i funkcije sa strelicama koje ne daju vlastitu vezu nego zadržavaju vrijednost općeg konteksta. U globalnom kontekstu izvršenje izvan bilo koje funkcije

odnosi se na globalni objekt bez obzira jesu li uvjeti strogi ili ne. Unutar funkcije vrijednost ovisi o tome kako se ta funkcija naziva. Metode ili funkcije su radnje koje se mogu izvršiti na objektima. Svojstva objekta mogu biti i primitivne vrijednosti, drugi objekti i funkcije. Metoda objekta je objektni entitet koji sadrži definiciju funkcije. [7]

Kada je riječ o nasljeđivanju, JavaScript ima samo jedan konstruktor za objekte. Svaki objekt ima privatni konstruktor koji ima vezu s drugim objektom koji se naziva prototipom. Taj prototipni objekt ima svoj vlastiti prototip, i tako dalje sve dok objekt ne izjednači s *null* kao prototipom. Prema definiciji, *null* nema prototip i djeluje kao konačna veza u ovom prototipnom lancu. Gotovo svi objekti u JavaScript su slučajevi objekta koji leži na vrhu prototipnog lanca. Iako se ova zbrka često smatra jednom od slabosti JavaScript-a, sam prototipni model nasljeđivanja zapravo je snažniji od klasičnog modela. [7]

Object Decorator Patterns ubrzavaju proces programiranja kroz korištenje ispitanih i u razvoju dokazanih razvojnih paradigmi. Koristi se kod ponovne upotrebe ili korištenja oblikovnih obrazaca tj. dijelova koda koje ćemo koristiti više od jednog puta te poboljšavaju čitljivost koda i reducira ga. Najčešća upotreba kod programskih okvira gdje je točno definirana programska logika te su na taj način već unaprijed definirani dekoratori tj. njihova logika. U primjeru vidimo objekt „Korisnik“ gdje ga kasnije dekoriranog možemo koristiti tj. pozivati za unos novog i prikaz podataka.

```
1. var Korisnik = function(ime) {
2.     this.ime = ime;
3.
4.     this.say = function() {
5.         log.add("Korisnik: " + this.ime);
6.     };
7. }
```

Slika 2 Primjer JavaScript objekta

Ovom funkcijom definirali smo tako da korisniku dodjeljujemo ime. „DecoratedKorisnik“ objekt koji u sebi sadrži vrijednosti korisnik(ime), ulicu i grad i funkciju za dodavanje

„*add“*, referenca na korisnika, definiramo sučelje za unos, implementacija funkcionalnosti.

```
1.  var DecoratedKorisnik = function(korisnik, ulica, grad) {
2.      this.korisnik = korisnik;
3.      this.ime = korisnik.ime;
4.      this.ulica = ulica;
5.      this.grad = grad;
6.
7.      this.say = function() {
8.          log.add("Decorated Korisnik: " + this.ime + ", " +
9.              this.ulica + ", " + this.grad);
10.     };
11. }
```

Slika 3 Primjer JavaScript objekta 2

„DecoratedKorisnik“ objekt koji u sebi sadrži vrijednosti korisnik(ime), ulicu i grad i funkciju za dodavanje „add“, referenca na korisnika, definiramo okruženje za unos, implementacija funkcionalnosti

```
1.  var DecoratedKorisnik = function(korisnik, ulica, grad) {
2.      this.korisnik = korisnik;
3.      this.ime = korisnik.ime;
4.      this.ulica = ulica;
5.      this.grad = grad;
6.
7.      this.say = function() {
8.          log.add("Decorated Korisnik: " + this.ime + ", " +
9.                      this.ulica + ", " + this.grad);
10.     };
11. }
```

Slika 4 Primjer JavaScript objekta 3

Za prijavu korisnika u sustav:

```
1.  function run() {
2.
3.      var korisnik = new Korisnik("Mario");
4.      korisnik.say();
5.
6.      var decorated = new DecoratedKorisnik(korisnik, "Strossamyerova", "Rijeka");
7.      decorated.say();
8.
9.      log.show();
10. }
```

Slika 5 Primjer JavaScript funkcije

Kreirali smo funkciju „run“ koja vrši prijavu korisnika u sustav.[7]

Na ovaj način smo proširili objekt „Korisnik“ pomoću dekoratora sa svojstvima „grad“ i „ulica“. Ime kao izvorni dio objekta mora ostati isti što je poanta cijele priče te po potrebi ćemo objekt „Korisnik“ na ovaj način nadograditi bez izmjena cijelog koda ili prilagodbi koje će nam zadati dodatnog posla.

Refactoring ne dodaje značajke ili funkcionalnosti u softverskom sustavu. To je vrlo dobra opcija za programiranje pri programiranju te održavanju koda. To olakšava razumijevanje softverskog sustava i jednostavnije i s time i jeftinije za izmjenu bez mijenjanja strukture koda i njegove prirode.

Prednosti:

• poboljšava dizajn softvera

• čini softver i kod razumljivim

• pomaže u pronalaženju nedostataka i *bug*-ova

• programiranje postaje brže

• kod je prilagodljiviji i lakši za održavanje ili nadogradnju

Dodavanje metoda konstruktorima je dodavanje funkcija tj. opcija. Konstruktor je klasa u kojoj se nalazi logika programa tj. *back-end*. U svakom konstruktoru su sve metode i funkcije koje upravljaju programom. Najpoželjnije je da postoji jedan bazični konstruktor u kojemu se nalaze osnovne i glavne metode programa te koji će produžiti druge kontrolere. Najjednostavnije je pozivati metode s „this“ funkcijom.

[8]

U tijelu klase izrazi se izvršavaju u strogom načinu rada. Ne mogu se pozvati iz druge klase ili neke metode van te klase. Metoda konstruktora je posebna metoda za stvaranje i inicijalizaciju objekta stvorenog pomoću klase. Postoji samo jedna specijalna metoda koja je konstruktor u klasi. Ukoliko ih imamo više od jedne kod se neće izvršiti te će biti prekinut. Konstruktor može upotrijebiti „super“ ključnu riječ za pozivanje konstruktora roditeljske klase.

```
1.  function run() {
2.
3.  class MojiMjeseci {
4.    constructor(godine, mjeseci) {
5.      this.godine = godine;
6.      this.mjeseci = mjeseci;
7.    }
8.
9.    get starost() {
10.     return this.izracunMj();
11.   }
12.
13.   izracunMj() {
14.     return this.godine * this.mjeseci;
15.   }
16. }
17.
18. const ja = new MojiMjeseci(20, 12);
19. console.log(ja.starost);
```

Slika 6 Primjer JavaScript klase i metoda

Kod izračunava starost u mjesecima. Definirane su varijable godine i mjeseci gdje mjeseci označavaju broj mjeseci u godini te godine za naše godine starosti. Konstruktor dodaje vrijednosti, tj. konstruira objekt dok funkcija „izracunMj“ množi mjesece s godinama te ju dobivamo s „*get“* funkcijom koja ju poziva. Konstanta „ja“ dodaje vrijednosti te se na kraju koda ispisuje koliko smo mjeseci stari. [9]

Kada govorimo o pseudoklasičnim obrascima (engl. *Pseudoclassical Patterns)* riječ je o sposobnosti jednog objekta da ima pristup svojstvima i metodama drugog objekta.

Od nadklasa klase nasljeđuju svojstva i metode. Podklase su sposobne nasljeđivati svojstva i metode njihovih nadklasa, a istodobno mogu stvarati svojstva i metode jedinstvene samo za njih. Podklase ne mogu mijenjati svojstva nadklasa već ih mogu „prepisati“ s metodom koja će biti istog naziva. Instance nadklasa i podklasa mogu delegirati metode iz prototipova. Prototipiranjem možemo nadklasi dodati iz podklase nekakvo novo svojstvo. Na taj način izbjegavamo redundantnost koda. Kreiranjem novih klasa prototipiranjem dobivamo pseudoklasične obrazac klase. [10]

Prethodno smo već rekli da je nadklasa ona klasa koja prosljeđuje svojstva i metode podklasama koje ju proširuju tj. nasljeđuju. Pseudoklasične podklase (engl. *Pseudoclassical Subclasses*) su tip klasa koje su najoptimiziraniji oblik klase. Prototipne metode naslijeđene su podklasama. U ovom tipu podklase stvaramo metodu u konstruktoru svaki puta kada pozivamo konstruktor za novu instancu. Na ovaj način prototipne metode ne nastaju svaki puta za svaku novu instancu ili poziv. Prototipne metode nasljeđuju se za svaku novu instancu ili poziv. Sintaksa je: Objekt “create“ za prototip. Svaka promjena u prototipu objekta odražava se na ostale kreirane instance.

```
var Osoba = function(ime, godine, visina) {
  Student.call(this, ime, godine, visina);
  this.rangVisine = null;
};

Osoba.prototype= Object.create(Student.prototype, {
  constructor: {value: Osoba}
});

Osoba.prototype.odjelzaInformatiku = function() {
  this.rangVisine += 1.80;
  this.odjelZaInformatiku = random;
}
```

Slika 7 Primjer JavaScript klasa

U primjeru sam pokazao kako se kreira jednostavna pseudoklasična podklasa gdje svaka „osoba“ postaje „student“ i dodjeljuje „visina“ te se rangira na odjelu za informatiku. U konačnici se dodjeljuje nasumičan broj radi pojednostavljenja primjera gdje bi se trebala vršiti usporedba s ostalim studentima. [10]

4. MEAN platforma

MEAN je akronim za MongoDB, Express.js, Angular.js, i Node.js koji čine skup JavaScript tehnologija za izradu dinamičkih web aplikacija. MEAN omogućuje izradu web aplikacija s jednim programskim jezikom. Bilo da je riječ o *back-endu* ili *front-endu*. Kako se koristi samo jedan programski jezik uvelike je olakšano programiranje i povezivanje svih dijelova web aplikacije. Kako je JavaScript i na serverskoj strani nisu potrebna ponovna učitavanja web stranica koje su izrađivane ovom tehnologijom.

PREDNOSTI

Kada je riječ o jednostraničnom web sjedištu tokom interakcije s web stranicom nije potrebno ni jedno ponovno učitavanje a kao odgovor na svaku našu radnju dobiti ćemo željene podatke ili akcije pravovremeno.

- MongoDB je dosta brži u odnosu na MySQL
- MEAN je skalabilniji u odnosu na LAMP
- Jedan programer obuhvaća *front-end* i *back-end* što nije nužno za LAMP
- Bolji rješenja u oblaku (engl. *Cloud*)

NEDOSTATCI

- Teško je pronaći eksperta u odnosu za LAMP
- Velik broj grešaka jer je novija tehnologija
- Slabije dokumentiran i manje iskustvo internet zajednica s radom u odnosu na LAMP.

[11]

Node.js je JavaScript okvir za *back-end* dio web aplikacija koji je izrađen na Chrome-ovom V8 JavaScript pogonu. Čini ga jedan od najvećih skupova JavaScript biblioteka u svijetu. Asinkron je i dizajniran za skalabilne web aplikacije. Node.js Foundation je organizacija koja se bavi razvojem ove tehnologije te pomaže zainteresiranima da

pridonesu svojim angažmanom u razvoju Node.js-a. Od kraja prvog desetljeća 21. stoljeća Node.js je jedan od najbrže rastućih projekata otvorenog koda. Neke od najpoznatijih web stranice izrađene na ovoj tehnologiji su „Netflix“, „Airbnb“, „Twitter“, „PayPal“, „Walmart“, „Yahoo“, koji je sada dio „Samsunga“, osnovao je originalni Node.js projekt. S vremenom projekt je rastao, sve veći broj programera se pridruživao te je i sam projekt poprimio odlike globalnog projekta. [11]

4.1. Node.js

Node.js je platforma koja podržava različite tipove aplikacija. Najčešće se koristi za izradu web aplikacija. Izgrađen je na „Google Chrome JSV8“ *engin-u* tj. motoru ili pokretaču. Njime možemo izgraditi brze, skalabilne *web* aplikacije na strani *servera*. Sadrži biblioteku za pokretanje JavaScript-a. Otvorenog je koda te potpuno besplatan za korištenje te sadrži pakete „npm“ koji čine najveći skup biblioteka otvorenog koda u svijetu današnjice. Postoje mnogi moduli koji podržavaju programiranje *web* aplikacija od kojih je „Connect“ modul najzastupljeniji danas u korištenju. „Connect“ modul sadrži mnoge omotače za Node.js-ov niži nivo i omogućuje programiranje s mnogim programskim okvirima.[12]

4.2. Express.js

Express.js je *web* programski okvir baziran na JavaScript-u. Sadrži neka od glavnih svojstava koja olakšavaju programiranje u Node.js tehnologiji. Omogućava da se programer drži strukture koda, pojednostavljuje programiranje i izradu aplikacije. Ugrađen je na konekciji na server te na taj način čini samu arhitekturu koda aplikacije. Express.js-a proširuje klasu „Connect*“* Node.js-a. Ovisan je o Node.js-u budući da ga samo proširuje. Express.js-om ćemo moći napisati cijeli kod vezan za *back-end*. Omogućava uključivanje modularnih HTML predložaka, proširuje odgovore koje vraća server u obliku objekta na način da podržava različite formate samog izlaza, upravlja rutama web stranice i još mnoge mogućnosti. [12]

4.3. MongoDB

MongoDB je predvodnik NoSQL tipa baze podataka. Jedna je on najčešće korištenih solucija za korištenje i upravljanje bazama podataka. NoSQL baze podataka nastale su kao odgovor na SQL baze podataka koje su relacijske i pisane u SQL jeziku.

Budući da su danas baze podataka sve veće i sve je teže upravljati njima NoSQL koncept sprema podatke na način da ih mapira te sprema u datoteke. U slučaju MongoDB-a koristi se samo JavaScript te se podaci spremaju u JSON formatu. Spremaju se na način da imaju svoje ključeve te u obliku objekata. Ovakav tip baze podataka je skalabilan te brži u svom radu u odnosu na SQL relacijske baze podataka. Upravljanje i kreiranje je brže i jednostavnije. Također je riječ o proizvodu otvorenog koda i besplatnom. Prva verzija se pojavila 2009. godine te je naziv dobio po engleskoj riječi „*humongous*“ što u prijevodu znači ogroman. Ovaj naziv je dobio budući da se sama potreba javila zbog sve većih baza podataka. [12]

4.4. Angular.js

Angular.js je platforma za jednostavnu izradu *web* aplikacija. Kombinira *template*-ove ili obrasce, skripte u ovisnosti koje proširuju kod, povezuje *back-end i front-end* te obrnuto te integrira najbolje prakse rješenja problema pri izradi *web* aplikacija. Omogućuje izradu bilo *desktop, web* ili mobilnih sučelja. Angular.js je JavaScript programski okvir nastao kao rješenje za jednostranične web aplikacije no koristi se i za višestranične. Izvršava se s klijentske strane u *web* pregledniku. Omogućuje lakše pisanje aplikacija te dolazi s rješenjima za vezanje podataka, promjena-detekcija, forme, rutanje tj. definiranje ruta aplikacije i navigacije te HTTP implementacija izvan okvira. S vremenom je nastala velika količina skripti vezane za ovaj programski okvir. Alati poput Angular-Cli, Angular universal i Angular Material dodani su u projekt i omogućuju izvršavanje serverske strane i izradu grafičkog sučelja. Koristi MVC arhitekturu koja objedinjava modele, poglede i kontrolere. Inače osim uz JavaScript može se ukomponirati i s drugim jezicima u *back-endu*. Dobar primjer gdje je Angular.js korišten u *front-end*-u, a PHP u *back-end*-u je web aplikacija domaće izrade Denti Croazia (http://www.denticroazia.it/). [13]

5. Instalacija

5.1. Instalacija na Linux Ubuntu operacijski sustav

Kako je riječ o operacijskom sustavu Linux, distribucija Ubuntu 18.04, instalacija MEAN platforme izvršava se preko terminala tj. komandne linije. Na jednostavan način upisom naredbi u terminal te kreiranjem konfiguracije lokalnog servera za naše buduće web sjedište osigurati ćemo potrebno radno okruženje sa svim potrebnim komponentama. Samo programiranje tj. programski kod će se pisati u IDE (engl. *Integrated development environment*, skraćeno IDE) alatu PhpStorm.

U prvom koraku dohvatiti ćemo skripte koje su potrebne za instalaciju naredbom „apt-get install build-essential git fontconfig libpng-dev ruby ruby-dev" te instalaciju pokrećemo naredbom „gem install sass".

Kako bismo instalirali MongoDB koristimo naredbu „apt-key adv –keyserver hkp://keyserver.ubuntu.com:80 –recv 0C49F3730359A14518585931BC711F9BA15703C6". Ona omogućuje dodavanje službene arhive skripti zadnje stabilne verzije. Pokretanje skripte tj. njezinu instalaciju pokrećemo naredbama za dodavanje arhive u na točno određenu putanju lokalno „echo „deb [arch=amd64,arm64] http://repo.MongoDB.org/apt/ubuntu xenial/MongoDB-org/3.4 multiverse" | tee /etc/apt/sources.list.d/MongoDB-org-3.4.list" i pokretanjem instalacije „apt-get install MongoDB-org". Nakon instalacije naredbama u terminalu možemo upravljati s našom bazom podataka prvenstveno za pokretanje naredbom „systemctl start mongod", te „ystemctl stop mongod" za zaustavljanje. Kako bismo provjerili je li MongoDB servis aktivan to možemo saznati upisivanjem naredbe „ystemctl status mongod" u terminal.

Sljedeći korak je instalacija NodeJS-a i NPM-a. Dodajemo resurse za instalaciju iz arhive skripti upisivanjem naredbe „curl -sL https://deb.nodesource.com/setup_8.x | -E bash -" te pokrećemo instalaciju s naredbom „apt-get install nodejs".

Yarn je upravljački alat za klijentsku stranu programiranja koji je ovisan o NodeJS i NPM-u. Instalirati ćemo ga naredbom „npm install -g yarn". Gulp je alat za izvršavanje velikog broja naredbi JavaScript jezika te se može instalirati upisom naredbe u terminal „npm install -g gulp".

Sama instalacija MEAN platforme će se izvršiti na način prvenstveno kloniranjem skripti iz online arhive naredbom „git clone https://github.com/meanjs/mean.git”. Kako bismo instalirali u terminalu ćemo se pozicionirati na adresu „mean” mape na računalu koja je kreirana tokom instalacije. Zatim pokrećemo instalaciju naredbom „npm install”. Nakon instalacije izvršiti ćemo naredbu koja će nam omogućiti pokretanje s dozvolama root tj. korijenskog ili glavnog korisnika. [14]

6. Zahtjevi za razvoj

6.1 Zahtjevi za dizajn sustava

U sklopu ovog rada, ukupno dvije web aplikacije će biti izrađene. Naime, jedna koristeći MEAN softverski paket, te druga koristeći LAMP softverski paket i Phalcon PHP programski okvir.

6.1.1 Softverski paket

Aplikacija koja će biti izrađena koristeći MEAN tehnologiju. Aplikacija će biti izrađena i u LAMP tehnologiji te će korisnik na osnovu A/B testiranja odabrati koju verziju aplikacije želi koristiti. Ona koja dobije ukupno bolju ocjenu same jednostavnosti izrade te kvalitete rada aplikacije biti će odabrana od strane korisnika. U LAMP tehnologiji koristiti će se predložak Phalcon te će aplikacija biti izrađena na MySQL relacijskoj bazi podataka.

6.1.2 Naziv aplikacije

Aplikacija će se zvati „Moja Knjiga“

6.1.3 Sigurnosni zahtjevi

Potrebno je napraviti aplikaciju na način da sama prijava i korištenje aplikacije bude osigurano potrebnom tehnologijom kako ne bi došlo do izloženosti podataka riziku od krađe i sl.

6.2 Korisnički zahtjevi

Aplikacija treba funkcionirati kao manja društvena mreža gdje će korisnici moći razmjenjivati knjige koje im se sviđaju te ih dijeliti, komentirati i ocjenjivati. Niže su navedeni korisnički zahtjevi koje aplikacija mora zadovoljavati te na taj način ispuniti korisničke zahtjeve. Oni omogućavaju ispravno i efikasno korištenje aplikacije i ispunjenje korisničkih ciljeva.

6.2.1 Unos knjiga

Korisnik treba moći unositi svoje najdraže knjige. Moći će dodavati knjige koje želi na način da unese u formu naziv knjige, ime autora te kratki opis same knjige. Biti će potrebno i dodati žanr gdje će izabrati jedan od ponuđenih žanrova.

6.2.3 Komentiranje knjiga

Korisnik će moći pisati komentare o knjigama te će ih ocjenjivati i tako dobiti uvid o popularnosti neke knjige na društvenoj mreži.

6.2.4 Promjena lozinke

Svaki korisnik imati će pristup svojim korisničkim podatcima te mogućnost izmjene lozinke.

6.2.5 Registracija

Svaki korisnik mora prethodno registrirati. Podatci za registraciju se sastoje od: imena, prezimena, e-mail adrese te lozinke.

6.2.6 Prijava

Nakon što se korisnik registrira moći će se prijaviti u aplikaciju sa svojom e-mail adresom i lozinkom.

6.2.7 Prikaz knjiga na korisničkom profilu

Korisnik će na svom profilu vidjeti popis svih knjiga koje je dodao te će ih moći ažurirati ukoliko je to potrebno. Svaka knjiga će se moći prikazati na njezinoj stranici te će tamo korisnici moći ocijeniti knjigu te upisati komentar.

6.2.8 Pretraživanje knjiga

Aplikacija će sadržavati pretragu. Korisnik će moći pretražiti knjige po ključnim riječima naziva knjige ili autora. Također rezultate će moći filtrirati po žanru. Iz liste rezultata korisnik će sa svakog rezultata moći pristupiti stranici za prikaz podataka te knjige. Knjige će biti sortirane silazno od najviše prosječne ocjene dodijeljene od korisnika.

7. Konfiguracija projekta

Struktura aplikacije je podijeljena u dva glavna dijela. Jedan dio se odnosi na serverski dio aplikacije u kojemu se nalazi sva logika i kod koji komunicira izravno sa serverom. Ovaj dio aplikacije ćemo smjestiti u mapu pod nazivom „nodejs". Drugi dio je klijentska strana aplikacije gdje koristimo Angular.js i u njega ćemo spremiti sve vezano za klijentski dio aplikacije koji je u konačnici vidljiv krajnjem korisniku i kojeg će pokretati internetski preglednik korisnika. Mapa u kojoj se nalazi ovaj dio imenovana je nazivom „MojaKnjiga".

U mapi „nodejs" se nalazi JavaScript skripta „app.js" koja je glava skripte serverske strane aplikacije i iz koje se pozivaju sve drugi skripte izravno ili neizravno. Osim nje imamo i skriptu „package.json" u koju dodajemo listu svih zavisnih biblioteke koje će aplikacija koristiti. Osim ovih skripti u „nodejs" folderu ćemo imati još nekoliko mapa. U mapi "node_modules" će se nalaziti sve biblioteke koje smo uključili preko „package.json". To su gotove biblioteke koje omogućavaju lakše programiranje i izradu aplikacije. U mapi „routes" ćemo kreirati skriptu s definiranim rutama aplikacije po kojima ćemo reći serveru gdje se nešto nalazi tj. na kojoj ruti nešto treba prikazati kako bi iz te rute klijentski dio aplikacije mogao dohvatiti što mu je potrebno. U mapi „models" ćemo kreirati sve modele naše baze podataka. Mapa „controllers" će sadržavati sve kontrolere iliti skripte koje sadržavaju svu logiku ovog dijela aplikacije. U skriptama koje će biti kreirane će se sadržavati sve potrebne klase i metode koje će davati upute serveru što da radi. i konačno imamo „config" mapu u koju ćemo smjestiti konfiguracijske skripte koje će sadržavati sve konstante i postavke aplikacije po kojima ćemo definirati osnovne konfiguracijske parametre za aplikaciju.

U mapi „MojaKnjiga" ćemo imati također „package.json" koja ima istu ulogu kao i u serverskom dijelu aplikacije. Osim nje imamo i „Angular.json" koja sadrži sve instalirane skripte za Angular.js i ovdje definiramo postavke poput naziva aplikacije, css skripti i sl. Također imamo i „tsconfig.json" i „tslint.json" koje definiraju skripte za strukturu koda te navigaciju. Te skripte nalaze se u mapi „e2e", a ostale biblioteke se instalirane se nalaze u „node_modules". Sve skripte koje budemo pisali ćemo smjestiti u mapu „src". U njoj ćemo kreirati glavnu konfiguracijsku skriptu za ovaj dio aplikacije „main.ts". U „app" folderu ćemo kreirati sve komponente ovog dijela aplikacije, dok se u „assets" nalaze dodatne js i css skripte koje budemo htjeli uključiti. Ovdje uključujemo „HTML5up" gotovu css skriptu koju ćemo uključili kako bi imali gotovo rješenje za

vizualni dio aplikacije poput gumbova, naslova, linkova i sl. U mapi „enviroments” ćemo definirati konfiguracijsku skriptu za okruženje u kojemu radimo. To može npr. biti produkcija, test i sl. „app” folder će sadržavati sve podmape komponenti aplikacije klijentskog dijela uključujući skripte klasa, metoda, servisa i sl. O ovoj mapi ćemo i kreirati skriptu za rute aplikacije koje će u konačnici biti dostupne i krajnjem korisniku. Svaka komponenta se sastoji od nekoliko skripti. „component.ts” skripta definira klase i metode vezane za komponentu. „component.spec.ts” je generirana skripta koja izvršava samo početni test te nije potrebna aplikaciji za rad. Još u ovom dijelu imamo i „component.html” skriptu koja sadržava HTML kod koji će se na kraju prikazati klijentu u vizualnom obliku.

Nakon instalacije MEAN platforme potrebno je napraviti konfiguraciju aplikacije. Aplikacija će se sastojati od dva dijela. Serverski dio za kojega ćemo kreirati mapu „nodejs” te klijentski dio za kojega izrađujemo mapu „MojaKnjiga”. U serverskom dijelu kreirati ćemo sve potrebne servise i klase potrebne za rad koji će se izvršavati na strani servera te komunicirati s bazom podataka MoongoDB.

```
require('./config/config');
require('./models/db');
require('./config/passportConfig');
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const passport = require('passport');
const rtsIndex = require('./routes/index.router');
var app = express();

app.use(bodyParser.json());
app.use(cors());
app.use(passport.initialize());
app.use('/mk', rtsIndex);
// error handler
app.use((err, req, res, next) => {
    if (err.name === 'ValidationError') {
        var valErrors = [];
        Object.keys(err.errors).forEach(key => valErrors.push(err.errors[key].message));
        res.status(422).send(valErrors)
    }
    else{
        console.log(err);
    }
})
// pokretanje nodejs servera
app.listen(process.env.PORT, () => console.log(`Server je pokrenut na portu : ${process.env.PORT}`));
```

Slika 8 "app.js"skripta serverskog dijela aplikacije

Aplikacija se pokreće sa skriptom app.js u koju smo uključili modele iz naše baze podataka te smo uključili i obrazac Express.js kako bismo mogli koristiti njegove biblioteke i sintaksu. Također uključujemo sve dodatne biblioteke poput Passport kojega ćemo objasniti u dijelu aplikacije za prijavu korisnika. Kako se aplikacija sastoji od dva dijela oba će se pokretati na različitim portovima. Serverski dio biti će dostupan na HTTP portu 3000 a klijentski dio na HTTP portu 4200. Kako bi klijentski dio mogao komunicirati sa serverskim koji je na drugom portu, koristiti ćemo biblioteku „cors" koja zaobilazi sigurnosno pravilo „Cross-Origin Resource Sharing" ugrađeno u sam web preglednik. Sama konfiguracija komunikacije dvaju dijelova aplikacija sa „cors-om"

često zna biti poprilično kompleksna. Ovisno o Angular.js verziji zna biti problema te je bolje koristiti čim noviju verziju. U izradi ove aplikacije korištena je verzija broj 6 koja nije najnovija no bolje je dokumentirana od najnovije verzije 7. Aplikaciju ćemo postaviti na rutu „/mk" što su inicijali za naziv aplikacije „Moja knjiga". U app.js definiramo pokretanje servera te dodajemo kod za prikaz greški.

Za definiranje ruta aplikacije kreirali smo mapu „routes" u kojoj kreiramo skriptu „index.router.js".

```
1. const express = require('express');
2. const router = express.Router();
3. const ctrlUser = require('../controllers/user.controller');
4.
5. //USER ROUTES
6. router.post('/register', ctrlUser.register);
7.
8. module.exports = router;
9. router.get('/libraryData', jwtHelper.verifyJwtToken, ctrlBook.libraryData);
```

Slika 9 Primjer "index.router.js" skripte

U kodu iznad vidimo našu skriptu za definiranje ruta u koju smo uključili kontroler „user" kojeg ćemo kreirati kasnije te smo definirali rutu za registriranje i za podatke dodanih knjiga od strane korisnika. Ovo je samo primjer te ćemo kasnije u svakom potrebnom koraku ažurirati ovu skriptu za svaku novu dodanu stranicu za našu aplikaciju.

Također biti će potrebno kreirati mapu „models" u kojoj će se nalaziti modeli za bazu podataka. Modele ćemo objasniti kroz daljnju izradu aplikacije kroz izradu komponenti. U mapi „controllers" kreirati ćemo kontrolere koji će upravljati kreiranim modelima.

Kreiramo mapu „config" u kojoj ćemo kreirati skriptu „config.js".

```
1. var env = process.env.NODE_ENV || 'development';
2. var config = require('./config.json');
3. var envConfig = config[env];
4. Object.keys(envConfig).forEach(key => process.env[key] = envConfig[key]);
```

Slika 10 primjer "config.js" konfiguracijske skripte

Iznad se nalazi kod skripte „config.js” U njoj smo definirali verziju aplikacije kao „development” tj. lokalni način rada aplikacije koji koriste programeri prije nego li aplikacija bude u nekom drugom načinu kao npr. testiranje ili produkcija. Dohvatili smo našu skriptu „config.json” koja sadrži podatke u obliku konstanti.

```
{
    "development": {
        "PORT" : 3000,
        "MONGODB_URI": "mongodb://localhost:27017/mojaknjigaDB",
        "JWT_SECRET": "sigurnost#4545",
        "JWT_EXP": "20m"
 }
```

Slika 11 Primjer "config.json" konfiguracijske skripte

Iznad se nalazi kod skripte „config.json” koju smo također smjestili u mapu „config”. U njoj redom definiramo port na kojemu će se izvršavati serverski dio aplikacije, MongoDB adresu, sigurnosni kod aplikacije pomoću kojega će se generirati token koji omogućuje sigurnosne provjere tokom korištenja aplikacije. Ovaj dio će detaljnije biti objašnjen u dijelu izrade komponente za prijavu korisnika. Zadnje definiramo vrijeme za koje će token isteći tj. postati nevažeći ukoliko se sa strane korisnika aplikacija ne koristi. Definirano je kao 20 minuta. Baza podataka MongoDB je nazvana „mojaknjigaDB”.

Klijentski konfiguracijski dio ćemo definirati u nekoliko skripti. U „main.ts” uključiti ćemo sam Angular.js te definirano okruženje u kojemu radimo.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';
if (environment.production) {
  enableProdMode();
}
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

Slika 12 Primjer "main.ts" skripte

Okruženje definiramo u mapi „environments” u skripti „enviroment.ts“.

```
1. export const environment = {
2.   production: false,
3.   apiBaseUrl: 'http://localhost:3000/mk'
4. };
```

Slika 13 Primjer "environments.ts" skripte

Ovdje smo definirali da projekt nije u produkciji te port i glavnu rutu aplikacije koja je definirana u serverskom dijelu kako bi mogao klijentski dio komunicirati s njim. Ostatak konfiguracije vezan je za same komponente pa ćemo ga objasniti na tom dijelu rada. Ostale konfiguracijske skripte i dodatne biblioteke su dodane na način da smo ih uključili u json konfiguracijske skripte te su se one instalirale preko naredbe „npm install”.

U klijentskom dijelu aplikacije za početak ćemo samo dodati ccs stil. Kopirati ćemo gotovu css skriptu u folder „assets” koji se nalazi u folderu „MojaKnjiga”. Ostatak razvoja klijentskog dijela biti će objašnjene kroz izradu svih komponenti aplikacije. Ono što je bitno u samoj globalnoj konfiguraciji u klijentskom dijelu da definiramo rute. To ćemo napraviti u mapi na ruti „MojaKnjiga/scr/app”. Tu kreiramo skriptu „routes.ts“

```
1. import { Routes } from '@angular/router';
2. import { UserComponent } from './user/user.component';
3. import { RegisterComponent } from './user/register/register.component';
4. import { LoginComponent } from './user/login/login.component';
5. import { UserDataComponent } from './user-data/user-data.component';
6. import { AddBookComponent } from './add-book/add-book.component';
7. import { HomePageComponent } from './home-page/home-page.component';
8. import { ViewBookComponent } from './view-book/view-book.component';
9. import { SecureGuard } from './secure/secure.guard';
10. export const appRoutes: Routes = [
11.     {
12.         path: 'register', component: UserComponent,
13.         children: [{ path: '', component: RegisterComponent }]
14.     },
15.     {
16.         path: 'login', component: UserComponent,
17.         children: [{ path: '', component: LoginComponent }]
18.     },
19.     {
20.         path: 'userdata', component: UserDataComponent, canActivate: [SecureGuard]
21.     },
22.     {
23.         path: '', redirectTo: '/login', pathMatch: 'full'
24.     },
25.     {
26.         path: 'addbook', component: AddBookComponent, canActivate: [SecureGuard]
27.     },
28.     {
29.         path: 'viewbook/:bookid', component: ViewBookComponent, canActivate: [SecureGuard]
30.     },
31.     {
32.         path: 'home', component: HomePageComponent, canActivate: [SecureGuard]
33.     }
34. ];
```

Slika 14 "routes.ts" skripta

Uključujemo sve komponente koje smo izradili u aplikaciji te definiramo rute kao što je npr. ruta za registraciju. Uvozimo tj. dodajemo klasu komponente „user” što vidimo u liniji 2 koda iznad. Zatim definiramo rutu za registraciju što vidimo na linijama od 10 do 14.

Pomoću dvije naredbe pokrenuti ćemo oba dijela aplikacije. Za serverski dio aplikacije ćemo na ruti „nodejs“ mape pokrenuti naredbu „ng app.js“. Njome šaljemo serveru zahtjev da čita „app.js“ skriptu iz koje će se pokrenuti dalje cijela aplikacija. Što se tiče klijentske strane aplikacije pokrećemo ju naredbom „ng serve --open“ na ruti mape „MojaKnjiga“. Ova naredba će kompajlirati cijeli kod „Angular.js“ dijela aplikacije. te će ju pokrenuti pri završetku procesa. Ukoliko imamo nekakvu grešku u kodu aplikacija se neće pokrenuti uspješno te ćemo vidjeti grešku u konzoli gdje smo pokrenuli naredbu.

Kada su riješene rute možemo kreirati izbornik u HTML skripti komponente „app“ tj. glava komponenta aplikacije.

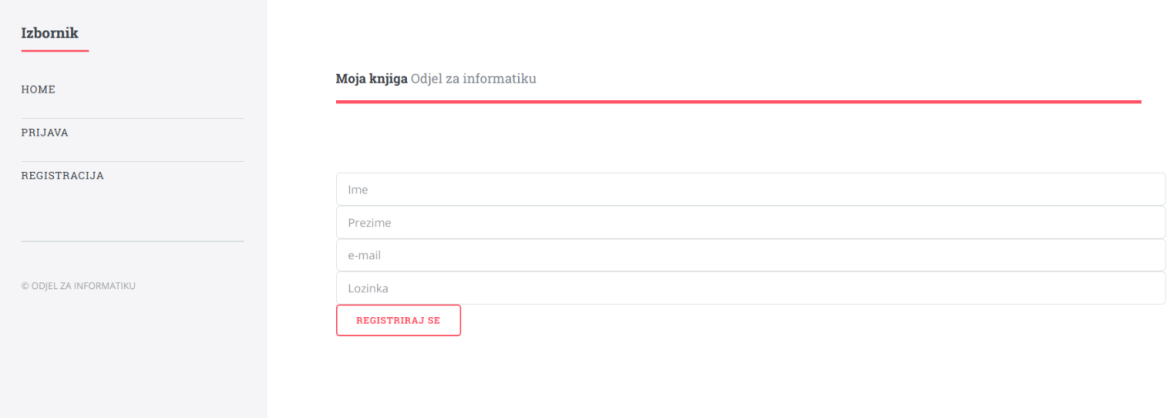
```
1. <li *ngIf="userService.checkLogin()">
2.     <p class="underlineHover" routerLink="/addbook"
3.         routerLinkActive="active">Dodaj Knjigu</p>
4. </li>
```

Slika 15 Primjer HTML skripte Angular.js komponente

U kodu iznad imamo primjer poveznice u izborniku za dodavanje nove knjige. Rutu mu definiramo preko parametra „routerLink“.

8. Registracija korisnika

8.1. Serverski dio komponente



Slika 16 Stranica za registraciju korisnika

Nakon završetka instalacije MEAN platforme možemo započeti s procesom programiranja te izrade aplikacije. Prva izrađena komponenta biti će registracija korisnika. Pri registraciji korisnik će trebati upisati u formu za registraciju ime, prezime, e-mail adresu te lozinku. Sva logika za ovaj proces biti će smještena u mapi nodejs servera, a klijent strana u mapi koja je za to predviđena tj. Angular.js mapa. Node.js mapu sam nazvao „nodejs“ dok Angular.js mapu „MojaKnjiga“.

Najprije ćemo napraviti model koji će sadržavati sve navedene podatke potrebne za registraciju te ćemo tamo definirati tip podatka također kako bi se svi podatci mogli uspješno spremiti u našu MongoDB bazu podataka. Kako bi smo to napravili kreirati ćemo skriptu u Node.js folderu u mapi models gdje će nam se nalaziti svi modeli. Model za korisnike nazvati ćemo „user” te skriptu nazvati user.model.js. Kako bi naš model mogao komunicirati s MongoDB trebamo uključiti potrebnu biblioteku Mongoose koja će obavljati taj posao. To ćemo učiniti kod svakog modela kojeg ćemo izraditi na sljedeći način:

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

Slika 17 Primjer uključivanja biblioteke "mongoose"

Definiranjem konstante u koju smo uključili „Mongoose“ biblioteku moći ćemo koristiti njezine metode u ovoj skripti. Sljedeći korak biti će definiranje objekta u kojemu će se nalaziti svi potrebni podatci tj. atributi koje će biti moguće spremiti za svakog korisnika u našu „MongoDB“ bazu podataka.

```
var userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: 'Ime je obavezno.'
    },
    surname: {
        type: String,
        required: 'Prezime je obavezno.'
    },
    email: {
        type: String,
        required: 'e-mail je obavezan',
        unique: true
    },
    password: {
        type: String,
        required: 'Lozinka je obavezna',
        minlength: [6, 'Lozinka se mora sadržavati od najmanje 6 simbola.']
    },
    saltSecurity: String
});
```

Slika 18 "userShema"

Iznad se nalazi naš objekt koji je shema za MongoDB tj. predstavlja novu deklaraciju objekta „Shema” koja je objekt biblioteke Mongoose koja omogućava komunikaciju s našom bazom podataka. Za svaki podatak koji će biti potrebno unijeti pri registraciji definiramo kao svojstvo našeg objekta. Za ime je definirano svojstvo „name” te je tip podatka tekst tj. *string* i definirali smo ga kao obvezno polje na način tako da smo mu dodali svojstvo „required”. Korisnik se neće moći registrirati bez da unese ovo polje. U registraciji sva polja će biti obavezna tako da smo na isti način definirali za sva polja ovog objekta. Prezime smo definirali pod nazivom „surname” te je također tip podatka

tekst. U oba polja smo definirali i poruku koja će se vratiti korisniku ukoliko ne unese tekst za bilo koje od ovih polja. E-mail adresa je definirana na isti način. Dodajemo joj svojstvo jedinstvenosti koje će pri registraciji onemogućiti da se korisnik registrira više puta s istom e-mail adresom. To smo napravili na način da smo u svojstvo „email" dodali „unique: true". Za lozinku smo definirali svojstvo „password". Dodali smo da se ona mora pri unosu sadržavati od minimalno 6 simbola. To smo učinili definiranjem „minlenghth" uvjeta s porukom na način kako se nalazi u kodu iznad. Osim svih potrebnih unosa tj. atributa u objektu smo definirali i dodatno svojstvo koje je nazvano „saltSecurity". Ovo svojstvo je tipa tekst i u njega ćemo spremiti sigurnosni kod preko kojega će lozinka biti kriptirana kako bi smo povećali sigurnost korisničkih računa.

Pri registraciji dobro je provjeriti je li korisnik unio ispravnu e-mail adresu. Kako bi smo izbjegli situaciju u kojoj korisnik može izbjeći provjeru u klijent dijelu, provjeru ćemo napraviti i u serverskom dijelu.

```
1. userSchema.path('email').validate((val) => {
2.     emailRegex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))
3.                     @((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|
4.                     (([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
5.     return emailRegex.test(val);
6. }, 'Neispravna e-mail adresa.');
```

Slika 19 Metoda za validaciju e-mail adrese

Dodali smo metodu provjere za svojstvo našeg objekta „email". Pozivom te metode pri unosu e-maila ispituje se ispravnost preko uzorka koji odgovara ispravnoj e-mail adresi. Ukoliko korisnik unese neispravnu e-mail adresu vratiti će se poruka koja će dati informaciju o tome. Sama metoda se poziva pri učitavanju registracije te provjerava polje „email", uzima njegovu vrijednost te provjerava je li ispravna. Uzorak koji predstavlja ispravnu e-mail adresu spremljen je u varijablu „emailRegex". Metoda koje će testirati je metoda „test" dok dohvaćena vrijednost dobiva naziv „val".

Na samom kraju komuniciramo s MongoDB-om te mu prosljeđujemo model tj. naš novokreirani objekt sa svim podatcima.

```
1. mongoose.model('User', userSchema);
```

Slika 20 Primjer uključivanja klase "User" u skriptu

U kontrolerima definiramo klase i njihove metode koje će komunicirati sa serverom. Za model korisnika definiramo kontroler „User“. Kreiramo skriptu u mapi „controllers” „user.controller.js”. Uključujemo na isti način biblioteku Mongoose kao u modelu te definiramo metodu za registraciju i uključujemo model koji smo prethodno napravili. Model uključujemo na sljedeći način:

```
1. const User = mongoose.model('User');
```

Slika 21 Instanciranje objekta "User"

Kada ga uključimo možemo ga koristiti te obrađivati.

```
1. module.exports.register = (req, res, next) => {
2.     var user = new User({
3.         name : req.body.name,
4.         surname : req.body.surname,
5.         email : req.body.email,
6.         password : req.body.password
7.     });
8.     user.save((err, doc) => {
9.         if (!err) {
10.            res.send(doc);
11.        } else {
12.            if (err.code == 11000) {
13.                res.status(422).send(['Ova e-mail adresa je već registrirana.']);
14.            } else {
15.                return next(err);
16.            }
17.        }
18.    });
19. }
```

Slika 22 Metode za spremanje novog korisnika u DB

Koristeći metode biblioteke našeg serverskog dijela kreirana je metoda „register” za registraciju korisnika. U njoj inicijaliziramo naš objekt „User” te mu dodjeljujemo vrijednost za svako svojstvo objekta. Dohvaćamo iz forme unesene podatke koja će naknadno biti kreirana u klijentskom dijelu. Redom za svako svojstvo dohvaćamo unos i definiramo ga u našoj novoj klasi „User” te nakon što smo objekt definirali pozivamo metodu „save” za spremanje te ukoliko je poslana tj. ukoliko je prošla provjera svih podataka uspješno podatci će biti spremljeni u bazu. Ukoliko su jedna ili više provjera prošle neuspješno vratiti će se poruka korisniku ovisno o tome do koje je pogreške kod unosa došlo. Kako bi smo ispravno vratili korisniku pogrešku potrebno je dohvatiti kod pogreške i definirati tekst poruke te ga vratiti u odgovor servera korisniku. Testiranjem unošenja neispravnih podataka u internet pregledniku možemo vidjeti koji kod greške se vraća kao odgovor te možemo za iste definirati poruke koje će se vratiti korisniku prilikom pogrešnih unosa. Za kod 11000 vraćamo status 422 koji vraća poruku da je e-mail adresa već registrirana. Naravno uvijek se može pojaviti i neka druga greška stoga i njih treba uzeti u obzir. Ukoliko se pojavi jedna od njih potrebno ih je vratiti u odgovoru servera („next(err)”). Ukoliko nema grešaka uneseni podatci će biti proslijeđeni serveru na spremanje metodom „send” koja šalje podatke „doc”.

```
1. userSchema.pre('save', function(next) {
2.     bcrypt.genSalt(10, (err, salt) => {
3.         bcrypt.hash(this.password, salt, (err, hash) => {
4.             this.password = hash;
5.             this.saltSecurity = salt;
6.             next();
7.         });
8.     });
9. });
```

Slika 23 Hashiranje lozinke

Metoda za kriptiranje lozinke je prikazana iznad u slici koda. Koristimo „bcrypt“ biblioteku koju smo uključili u model te pri spremanju korisnika lozinka se kriptira i se sprema u tom obliku. Sprema se i „saltSecurity” atribut pomoću kojeg će lozinka biti dekriptirana prilikom prijave korisnika.

Kako bi korisnik mogao komunicirati sa serverom potrebno je u server dijelu definirati rute preko kojih ćemo pristupiti kontroleru. U folderu „routes” kreiramo skriptu „index.router.js”. U njoj uključujemo Express.js.

```
1. const express = require('express');
2. const router = express.Router();
3. const ctrlUser = require('../controllers/user.controller');
```

Slika 24 Uključivanje dodatnih biblioteka, kontrolera i klasa

Pozivamo njezinu Express.js metodu „Router” koja omogućuje komunikaciju sa serverom. Zatim uključujemo naš kontroler. Nakon toga definiramo našu URL rutu na kojoj će se korisniku prikazati stranica za registraciju.

```
1. router.post('/register', ctrlUser.register);
```

Slika 25 Primjer rute

Ovime smo završili serversku stranu dijela aplikacije za registraciju.

8.2. Klijentski dio komponente

Nakon što smo definirali sve potrebno za serverski dio aplikacije vezano za registraciju krećemo u izradu klijent strane tj. „Angular.js” dio aplikacije.

Prvo ćemo kreirati u folderu „MojaKnjiga” koji sadržava klijentsku stranu aplikacije. U podfolderu „src” u kojemu se nalaze naše skripte za klijentsku stranu aplikacije kreiramo još jedan folder s nazivom „user” koji će sadržavati sve skripte vezane za naš model „User” i obradu istog s klijent strane. Sve potrebne skripte neophodne za rad Angular.js-a možemo izgenerirati naredbom *„ng g c user“* koju upisujemo u konzoli kada smo pozicionirani u folderu „MojaKnjiga”. Na ovaj način generirale su se skripte i folder „user”. U njemu sada možemo pronaći skripte „user.component.html” i „user.component.ts”. U prvoj skripti definiramo HTML kod za

registraciju. U drugoj skripti ćemo pisati potrebne kontrole i metode za ovu komponentu.

U HTML skripti nećemo ništa dodatno upisivati. Ova skripta će samo pozivati potrebne HTML skripte podkomponenti za registraciju te kasnije i za prijavu korisnika. Uključiti ćemo ih pomoću jedne linije koda.

```
1. <router-outlet></router-outlet>
```

Slika 26 HTML kod za dohvat proširenog HTML koda

Ono što nam preostaje za kreirati je skripta koja sadrži kontrole. „user.component.ts" skriptu ćemo uključiti Angular.js.

```
1. import {Component, OnInit} from '@angular/core';
```

Slika 27 Primjer uključivanja Angular.js skripti

Zatim uključujemo HTML skriptu ove komponente.

```
1. @Component({
2.     selector: 'app-user',
3.     templateUrl: './user.component.html'
4. })
```

Slika 28 Primjer uključivanja HTML komponente

Samu klasu nećemo mijenjati. U nju će se uključiti klase i metode njezinih podkomponenti.

```
1. export class UserComponent implements OnInit {
2.     constructor() {
3.     }
4.     ngOnInit() {
5.     }
6. }
```

Slika 29 Primjer klase sa inicijalizacijskom metodom

Klasa dobiva naziv po komponenti kako smo ju definirali pri izradi. Ima konstruktor i metodu „ngOnInit“ koja definira što će se događati tokom instanciranja klase. U ovom slučaju ostavljamo prazno jer nije potrebno postavljati nikakva svojstva ili dohvaćati podatke iz baze podataka pri učitavanju komponente. Na isti način kreiramo podkomponentu za registraciju „register” u mapi „user”. Krećemo s izradom njezine prve skripte za definiranje kontrola tj. skripte „register.component.js”.

```
import {Component, OnInit} from '@angular/core';
import {NgForm} from '@angular/forms';
import {UserService} from '../../shared/user.service';
@Component({
    selector: 'app-register',
    templateUrl: './register.component.html',
    styleUrls: ['../../../assets/css/main.css' ]
})
export class RegisterComponent implements OnInit {
    emailRegex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|
        (".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|
        (([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    showSuccessMessage: boolean;
    serverErrorMessages: string;
    constructor(private userService: UserService) {
    }
    ngOnInit() {
    }
    onSubmit(form: NgForm) {
        this.userService.registerUser(form.value).subscribe(
            res => {
                this.showSuccessMessage = true;
                setTimeout(() => this.showSuccessMessage = false, 4000);
                this.resetForm(form);
            },
            err => {
                if (err.status === 422) {
                    this.serverErrorMessages = err.error.join('<br/>');
                } else {
                    this.serverErrorMessages = 'Nešto nije uredu.';
                }
            }
        );
    }
```

Slika 30 "register.component.js" skripta 1/2

```
34.     }
35.     resetForm(form: NgForm) {
36.         this.userService.selectedUser = {
37.             id: '',
38.             name: '',
39.             surname: '',
40.             email: '',
41.             password: ''
42.         };
43.         form.resetForm();
44.         this.serverErrorMessages = '';
45.     }
46. }
```

Slika 31 "register.component.js" skripta 2/2

Iznad je kod skripte „register.component.ts”. Kao i u svakoj skripti ove vrste i ovdje smo uključili Angular.js. Osim Angulara.js uključujemo i njegovu klasu „NgForm”. Uključivanje iste omogućuje korištenje Angular.js formi i logike za njih. Na ovaj način se pojednostavljuje cijeli proces izrade aplikacije. Uključujemo klasu „UserService” koju ćemo kreirati i koja će sadržavati svu logiku vezanu za komponentu „user” pa tako i za njezine podkomponente. Definiramo *selektor* kako bi Angular.js znao gdje se nalazi pri čitanju skripte tj. dodjeljujemo naziv. Također uključujemo CSS stilove i HTML komponentu. Isto kako smo definirali provjeru ispravnosti e-mail adrese koja se unosi od strane korisnika pri registraciji u serverskom dijelu aplikacije, tako smo napravili i na klijentskoj strani i definirali uzorak koji predstavlja ispravnu e-mail adresu. Definiramo kojeg tipa će biti poruka za uspješnu i neuspješnu registraciju. Uspjeh ili „showSuccessMessage” biti će tipa „boolean”. Ovaj tip nam vraća kao odgovor servera istinu tj. „true” ili laž „false”. U slučaju uspjeha kao odgovor ćemo dobiti „true”. Neuspjeh će vratiti tekstualni oblik odgovora tj. „string”. Konstruktor prima servis „user” komponente. Kako bismo ga mogli koristili potrebno je prvo kreirati isti što će biti objašnjeno u sljedećem odjeljku. Klasa „RegisterComponent” vrši slanje podataka serveru za registraciju, obrađuje te podatke te ih na kraju sprema. Server će vratiti kao odgovor je li registracija obavljena uspješno ili ne. Klasa „RegisterComponent” sadrži

metode slanja „onSubmit” te „resetForm” koje upravljaju formom iliti obrascem za registraciju sa svim njegovim komponentama.

U „app” podmapi „shared” kreirati ćemo sve servise pa tako i servis „user.service.ts”.

```
1. import { Injectable } from '@angular/core';
2. import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
3. import { environment } from '../../environments/environment';
4. import { User } from './user.model';
5. @Injectable({
6.   providedIn: 'root'
7. })
```

Slika 32 "user.service.ts" uključivanje biblioteka

U njega ćemo uključiti potrebne skripte MEAN platforme za ovaj servis kao što je model „user” te pomoćne skripte za Angular.js-a. U servisu ćemo u klasi definirati sve metode koje nam budu bile potrebne.

```
1. registerUser(user: User){
2.   console.log(user);
3.   return this.http.post(environment.apiBaseUrl + '/register', user, this.noSecureHeader);
4. }
```

Slika 33 "user.service.ts" metoda za registraciju

Osim metode potrebno je kreirati i varijablu koja će sadržavati registracijske podatke koji se budu unosili u obrazac pri registraciji korisnika.

```
selectedUser: User = {
  id:'',
  name: '',
  surname: '',
  email: '',
  password: '',
  groups:[]
};
```

Slika 34 "user.service.ts" objekt "selectedUser"

U varijablu „selectedUser” dodajemo klasu „User” koja nema vrijednosti za pojedina svojstva te će ona biti definirana nakon što korisnik pri registraciji upiše podatke.

Definirali smo u klasi „UserService” metodu za registraciju koja šalje podatke na server preko definirane rute.

Ono što nam je preostalo je kreirati HTML formu tj. HTML kod koji će se ispisati prilikom učitavanja stranice tj. registracijske komponente.

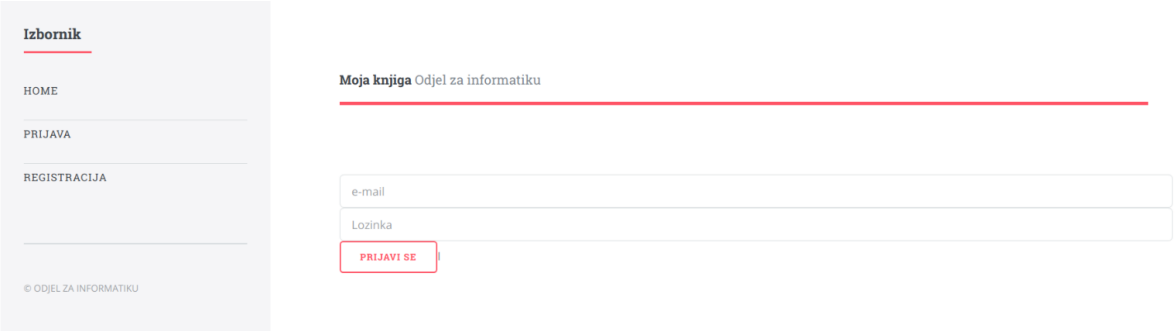
```
<form #registerForm="ngForm" (ngSubmit)="registerForm.valid && onSubmit(registerForm)">
  <input type="text" #name="ngModel" [(ngModel)]="userService.selectedUser.name" name="name"
    placeholder="name"
  required  [ngClass]="{'invalid-textbox' :registerForm.submitted && !name.valid }">
  <div *ngIf="registerForm.submitted && !name.valid">
    <label class="validation-message">This field is required.</label>
  </div>
  <input type="text" #surname="ngModel" [(ngModel)]="userService.selectedUser.surname" name="surname"
    placeholder="surname"
         required  [ngClass]="{'invalid-textbox' :registerForm.submitted && !surname.valid }">
  <div *ngIf="registerForm.submitted && !surname.valid">
    <label class="validation-message">This field is required.</label>
  </div>
  <input type="text" #email="ngModel" [(ngModel)]="userService.selectedUser.email" name="email"
    placeholder="Email"
  required [pattern]="emailRegex"  [ngClass]="{'invalid-textbox' :registerForm.submitted && !email.valid }">
  <div *ngIf="registerForm.submitted && email.errors">
    <label *ngIf="email.errors.required" class="validation-message">This field is required.</label>
    <label *ngIf="email.errors.pattern" class="validation-message">Invalid email address.</label>
  </div>
  <input type="password" #password="ngModel" [(ngModel)]="userService.selectedUser.password" name="password"
    placeholder="Password"
  minlength="4" required [ngClass]="{'invalid-textbox' :registerForm.submitted && !password.valid }">
  <div *ngIf="registerForm.submitted && password.errors">
    <label *ngIf="password.errors.required" class="validation-message">This field is required.</label>
    <label *ngIf="password.errors.minlength" class="validation-message">Enter atleast 4 characters.</label>
  </div>

  <input type="text" #group="ngModel" [(ngModel)]="userService.selectedUser.group" name="group"
    placeholder="group"
  required  [ngClass]="{'invalid-textbox' :registerForm.submitted && !group.valid }">
  <div *ngIf="registerForm.submitted && !group.valid">
    <label class="validation-message">This field is required.</label>
  </div>
  <input type="submit" value="Register">
</form>
<!-- Poruka - uspjeh -->
<div class="success" *ngIf="showSucessMessage">
  Saved successfully
</div>
<!-- Poruka - greška -->
<div class="alert" *ngIf="serverErrorMessages">
  {{serverErrorMessages}}
</div>
```

Slika 35 HTML skripta registracijske komponente

U Angular.js-u kreiramo HTML formu koristeći Angular.js specifikaciju za definiranje formi. Svaka forma izravno komunicira s Angular.js-om i tipa je „ngForm” po kojemu će biti prepoznata kao forma. U ovom slučaju „#registerForm” je identifikacija forme po kojoj će biti prepoznata. Definiramo model za svako polje unosa i njegovo ime. Model je definiran preko servisa kojeg smo definirali u prethodnom koraku. Uključene su i validacije poput obaveznih polja, dužine unosa teksta te poruka za uspješnu ili neuspješnu registraciju. Forma će komunicirati s klijentskim dijelom aplikacije preko servisa te pomoću metoda će komunicirati sa serverskim dijelom koji će u konačnici proslijediti bazi podataka da spremi podatke ukoliko je to potrebno. Kreiranje same HTML forme nešto je kompliciranije u odnosu na LAMP tehnologiju. Sintaksa je kompleksnija što možemo vidjeti iz primjera koda iznad. Sve komponente HTML elementa za unos „input” ili imena za unos „label” imaju prefiks „ng“ te možemo uključiti uvjetovanja, poruke neispravnog unosa i uključiti validacije. Prilikom svakog inputa trebamo se referencirati na servis komponente „user”.

9. Prijava



Slika 36 Stranica za prijavu korisnika

9.1. Serverski dio komponente

Prijava korisnika u sustav će funkcionirati na način da korisnik unosi svoju e-mail adresu te lozinku koju je definirao. Lozinka je spremljena u kriptiranom obliku te će se ona dekriptirati i serverski dio aplikacije će provjeriti ispravnost podataka. Ukoliko nisu ispravni vratiti će poruku o tome, a ukoliko su ispravni, na računalo korisnika će se spremiti token pomoću kojeg će korisnik imati pristup stranicama aplikacije na koje se ne može pristupiti bez prijave korisnika. Token će biti dostupan korisniku i može ga pročitati no ta mogućnost nije štetna za sigurnost korisnika obzirom da je vidljiva samo tom korisniku nakon prijave. Kada se korisnik odjavi on će se obrisati i generirati ponovo drukčiji prilikom sljedeće prijave.

U skriptu „index.router.js" definiramo novu rutu za prijavu korisnika.

```
1. router.post('/authenticate', ctrlUser.authenticate);
```

Slika 37 Definiranje rute za prijavu

U kontroleru kreiramo metodu za prijavu korisnika.

```
module.exports.authenticate = (req, res, next) => {
    console.log(req.body);
    // poziv za prolaz autorizacije
    passport.authenticate('local', (err, user, info) => {
        // greška passport autorizacije
        if (err) {
            return res.status(400).json(err);
        }// registrirani korisnik
        else if (user) {
            return res.status(200).json({ "token": user.generateJwt() });
        }// nepoznati korisnik ili pogrešna lozinka
        else {
            return res.status(404).json(info);
        }
    })(req, res);
}
```

Slika 38 Metoda za projavu korisnika

Ova metoda prikuplja unesene podatke korisnika za prijavu i verificira ih. Ukoliko provjera bude uspješna generirati će se token pozivom na odgovarajuću metodu koju ćemo također kreirati „user.generateJwt”. Kako bi smo mogli izvršiti verifikaciju unesenih podataka za prijavu korisnika uključiti ćemo dvije biblioteke kako bi smo ih ukomponirali u Angular.js i mogli koristiti njihove metode. Uključiti ćemo ih u model „User”.

```
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
```

Slika 39 Dodavanje biblioteka za kriptiranje i kreiranje tokena

Riječ je o bibliotekama „cryptjs” koja će nam omogućiti jednostavno kriptiranje i dekriptiranje lozinke te „jsonwebtoken” koja će nam omogućiti upravljanje našim token-om. U skriptu „user.model.js” ćemo dodati dvije nove metode.

```
1.  userSchema.methods.verifyPassword = function(password) {
2.      console.log(password);
3.      console.log(this.password);
4.      return bcrypt.compareSync(password, this.password);
5.  };
6.  userSchema.methods.generateJwt = function() {
7.      return jwt.sign({_id: this._id},
8.          process.env.JWT_SECRET,
9.          {
10.             expiresIn: process.env.JWT_EXP
11.         });
12. }
```

Slika 40 Metoda za provjeru lozinke i generiranje tokena

Prva metoda služi za provjeru lozinke na način da ju dekriptira. Kao rezultat vratiti će dekriptiranu lozinku. Druga metoda će izgenerirati token koji istječe kroz definirano vrijeme u konfiguraciji. U metodi za prijavu „module.exports.authenticate” se pozivaju ove dvije metode te se vrši prijava korisnika. U konfiguracijski folder serverskog dijela aplikacije ćemo kreirati konfiguracijsku skriptu za prijavu korisnika „passportConfig.js”.

```
const passport = require('passport');
const localStrategy = require('passport-local').Strategy;
const mongoose = require('mongoose');
var User = mongoose.model('User');
passport.use(
    new localStrategy({usernameField: 'email'},
        (username, password, done) => {
            User.findOne({email: username},
                (err, user) => {
                    if (err) {
                        return done(err);
                    }// nepoznati korisnik
                    else if (!user) {

                        return done(null, false, {message: 'e-mail nije registriran'});
                    }// pogrešna lozinka
                    else if (!user.verifyPassword(password)) {
                        return done(null, false, {message: 'Pogrešna lozinka.'});
                    }// uspješna autorizacija
                    else {
                        return done(null, user);
                    }
                });
        })
);
```

Slika 41 "passportConfig.js" skripta

U nju ćemo uključiti biblioteku „passport” koja omogućuje lakšu prijavu korisnika. Koristimo klasu „Strategy” i biblioteke pomoću koje ćemo obaviti validaciju unesenih podataka. Konstruiranjem definiramo podatke prijave. U našem slučaju identifikacijsko ime je „email” te lozinka „password”. Ova klasa će pretražiti model „User” i pronaći odgovarajuću kombinaciju unesenih podataka. Pomoću metode „findOne” pronaći ćemo korisnika. Ukoliko pretraga bude uspješna vratiti će nam se odgovor servera „done”. Ova metoda vraća objekt korisnika iz naše baze podataka. Ukoliko pronađe korisnika i lozinka ne bude odgovarajuća dobiti ćemo o tome poruku, a ukoliko je pogrešan e-mail tj. nije prethodno registriran također kao odgovor servera vraća se odgovarajuća poruka. Pomoću ove skripte ostvarili smo komunikaciju s bazom podataka. Pomoću metode za provjeru lozinke „verifyPassword” i pretrage baze

podataka možemo provjeriti i dohvatiti pronađenog korisnika sa svim njegovim podatcima koji se vraćaju kao odgovor servera u obliku objekta.

Nakon prijave generira se token. U tom trenutku korisnik će biti preusmjeren na početnu stranicu dijela aplikacije za prijavljenog korisnika. Naravno, uvijek nešto može poći u krivom smjeru te je potrebno provjeriti je li token generiran u ispravnom obliku. Kako bismo ga mogli provjeriti kreirati ćemo potrebnu skriptu s kojom ćemo to ostvariti. Ponovno u konfiguracijskom folderu kreiramo skriptu, sada pod nazivom „jwtHelper.js”.

```
1.  const jwt = require('jsonwebtoken');
2.  module.exports.verifyJwtToken = (req, res, next) => {
3.
4.      var token;
5.      if ('authorization' in req.headers)
6.          token = req.headers['authorization'].split(' ')[1];
7.      if (!token){
8.          return res.status(403).send({ secure: false, message: 'Nema tokena.' });
9.      }
10.     else {
11.         jwt.verify(token, process.env.JWT_SECRET,
12.             (err, decoded) => {
13.                 if (err)
14.                     return res.status(500).send({ secure: false, message:
15.                         'Token autorizacija je neuspješna.' });
16.                 else {
17.                     req._id = decoded._id;
18.                     next();
19.                 }
20.             }
21.         )
22.     }
23. }
```

Slika 42 "jwtHelper.js" skripta

Koristimo pomoćnu biblioteku „jsonwebtoken” te kreiramo metodu kojoj prosljeđujemo upit, odgovor i metodu koja slijedi. Iz „headers-a” koje šalje server dohvaćamo token

i provjeravamo postoji li i je li ispravan. Ova skripta omogućuje komunikaciju sa serverom tako da dohvaća sve podatke iz zaglavlja i čita token.

9.2. Klijentski dio komponente

Klijentski dio aplikacije komponente prijave korisnika ćemo kreirati na isti način u mapi „MojaKnjiga" kao kod registracije. Komponentu kreiramo u već kreiranoj mapi „user" kao podkomponentu definiramo „login" za prijavu korisnika. U folderu „login" ćemo pronaći sve dijelove komponente za prijavu, a u „shared" folder koji smo spominjali u registraciji ćemo u „user.service.ts" skriptu dodati metode koje će omogućavati prijavu. Za prijavu ćemo imati i dodatnu komponentu za sigurnost i provjeru korisnika pri prijavi te ćemo ju nazvati „secure". Kako bismo kreirali komponentu koja će vršiti provjeru i omogućavati obavljanje sigurnosti kroz rad u cijeloj aplikaciji naredbom u terminalu ćemo ju generirati. „ng g g secure" naredba će generirati skripte koje su nam potrebne.

```
import {
    HttpInterceptor,
    HttpRequest,
    HttpHandler,
    HttpEvent
} from '@angular/common/http';
import {Injectable} from '@angular/core';
import {tap} from 'rxjs/operators';
import {Router} from '@angular/router';
import {UserService} from '../shared/user.service';
@Injectable()
export class SecureInterceptor implements HttpInterceptor {
    constructor(private userService: UserService, private router: Router) {
    }
    intercept(req: HttpRequest<any>, next: HttpHandler) {
        if (req.headers.get('nosecure')) {
            return next.handle(req.clone());
        } else {
            const cloneRequest = req.clone({
                headers: req.headers.set("Authorization", "Bearer " + this.userService.getToken())
            });
            return next.handle(cloneRequest).pipe(
                tap(
                    event => {
                    },
                    err => {
                        if (err.error.secure === false) {
                            this.router.navigateByUrl('/login');
                        }
                    })
            );
        }
    }
}
```

Slika 43 "secure.interceptor.ts" skripta

U „secure.interceptor.ts” skrpti kreirana je klasa koja implementira „HttpInterceptor”. Proširili smo ju tako da smo joj uključili naše novokreirane metodu „getToken” koja će dohvatiti korisnički token i služiti kao *interceptor*. *Interceptor* je uzorak dizajna softvera koji prihvaća unos, obrađuje ga i potom prosljeđuje dalje. U našem slučaju, ova klasa dodaje autorizacijski token u svaki HTTP zahtjev koji se šalje prema serveru. Ono što

će nam osiguravati je to da korisnici mogu pristupiti svim stranicama aplikacije ukoliko su se uspješno prijavili te posjeduju token. Jedine stranice kojima će se moći pristupiti bez prijave su stranica za prijavu i registraciju. Druga skripta koja se generirala je „secure.guard.ts”.

```
1. import {Injectable} from '@angular/core';
2. import {
3.     CanActivate,
4.     ActivatedRouteSnapshot,
5.     RouterStateSnapshot
6. } from '@angular/router';
7. import {Observable} from 'rxjs';
8. import {UserService} from '../shared/user.service';
9. import {Router} from '@angular/router';
10. @Injectable({
11.     providedIn: 'root'
12. })
13. export class SecureGuard implements CanActivate {
14.     constructor(private userService: UserService, private router: Router) {
15.     }
16.     canActivate(
17.         next: ActivatedRouteSnapshot,
18.         state: RouterStateSnapshot): boolean {
19.         if (!this.userService.checkLogin()) {
20.             this.router.navigateByUrl('/login');
21.             this.userService.removeToken();
22.             return false;
23.         }
24.         return true;
25.     }
26. }
```

Slika 44 secure.guard.ts skripta

Kao i u prošloj skripti uključujemo sve potrebne klase i biblioteke te modificiramo postojeću klasu „CanActivate” u kojoj za svaku rutu provjeravamo je li korisnik ulogiran.

Provjeravamo s metodom „checkLogin”. Ukoliko korisnik nije prijavljen biti će preusmjeren na stranicu za prijavu. Ako je token spremljen može biti u nekoj situaciji biti neispravan i on će tada biti obrisan metodom „removeToken” što je zapravo metoda za odjavu korisnika.

U „user.service.ts” dodajemo nove metode od kojih smo već neke naveli u prethodnom odjeljku.

```
1. login(authCredentials) {
2.   return this.http.post(environment.apiBaseUrl +
3.     '/authenticate', authCredentials, this.noSecureHeader);
4. }
```

Slika 45 Metoda za prijavu - klijentski dio

Metoda „login” očekuje podatke za prijavu. Provjerava unesenu e-mail adresu i lozinku. Ukoliko su ispravni korisnik će biti prijavljen. Definirali smo rutu koja završava s „/authenticate”.

```
1. setToken(token: string) {
2.   localStorage.setItem('token', token)
3. }
```

Slika 46 Metoda za postavljanje tokena

Metoda „setToken” će izgenerirati token koji će služiti kao verifikacijski ključ pri kretanju po aplikaciji.

```
1. getToken() {
2.   return localStorage.getItem('token');
3. }
```

Slika 47 Metoda za dohvat tokena

Metoda „getToken” omogućuje dohvat tokena koji je spremljen lokalno kod korisnika na računalu.

```
removeToken() {
  localStorage.removeItem('token');
}
```

Slika 48 Metoda za brisanje tokena

„removeToken” je metoda za odjavu korisnika tj. briše token koji je spremljen lokalno i na taj način se onemogućuje korisniku da se kreće po dijelu aplikacije za kojeg je potrebna prijava.

```
getPayload() {
  var token = this.getToken();
  if (token) {
    var userPayload = atob(token.split('.')[1]);
    return JSON.parse(userPayload);
  } else {
      return null;
  }
}
```

Slika 49 "getPayload" metoda

Metoda „getPayload” dohvaća token i parsira ga tako da od odgovora servera dohvatimo dio koji je njegova vrijednost.

```
checkLogin() {
  var userPayload = this.getPayload();
  if (userPayload) {
      return userPayload.exp > Date.now() / 1000;
  } else {
      return false;
  }
}
```

Slika 50 Metoda za provjeru prijave

„checkLogin” metoda dohvaća vrijednost tokena i na taj način provjerava je li korisnik prijavljen u sustav.

Nakon što smo ažurirali korisnički servis i kreirali komponentu koja rješava problem sigurnosti nadograditi ćemo kreiranu komponentu za prijavu koja je podkomponenta komponente „user”.

```
import {Component, OnInit} from '@angular/core';
import {NgForm} from '@angular/forms';
import {Router} from '@angular/router';
import {UserService} from '../../shared/user.service';
@Component({
    selector: 'app-login',
    templateUrl: './login.component.html'
})
export class LoginComponent implements OnInit {
    constructor(private userService: UserService, private router: Router) {
    }
    model = {
        email: '',
        password: ''
    };
    emailRegex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))
        @((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)
        +[a-zA-Z]{2,}))$/;
    serverErrorMessages: string;
    ngOnInit() {
        if (this.userService.checkLogin())
            this.router.navigateByUrl('/home');
    }
    onSubmit(form: NgForm) {
        this.userService.login(form.value).subscribe(
            res => {
                this.userService.setToken(res['token']);
                this.router.navigateByUrl('/home');
            },
            err => {
                this.serverErrorMessages = err.error.message;
            }
        );
    }
}
```

Slika 51 "login.component.ts" skripta

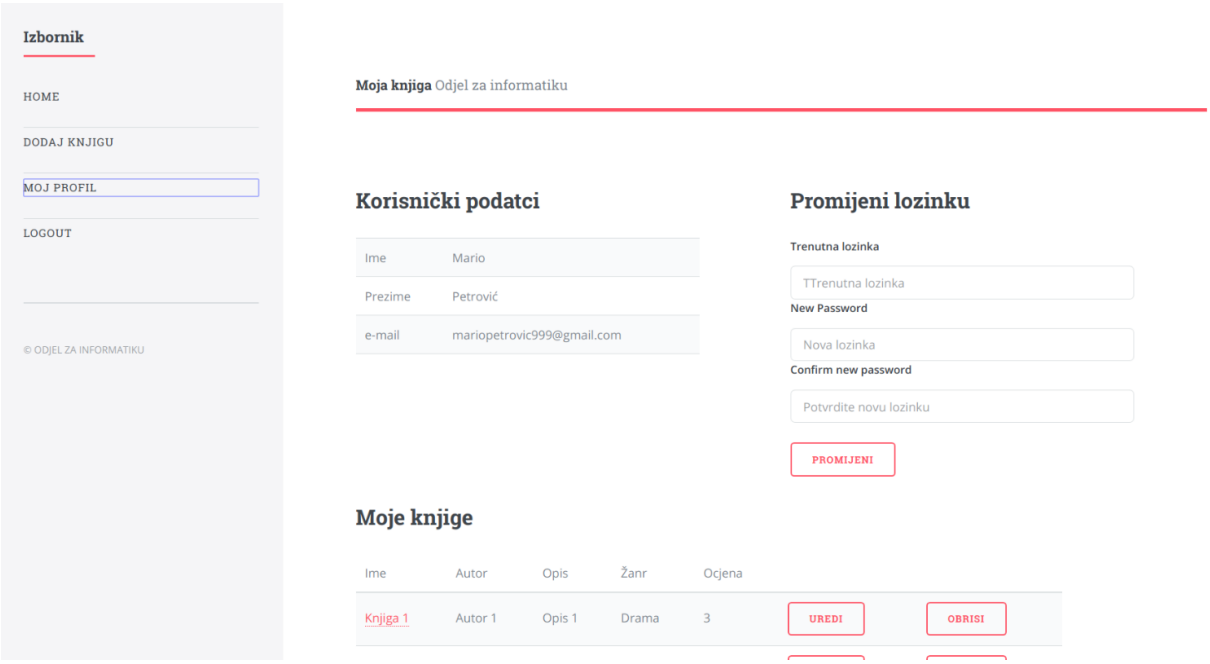
U mapi „login” smo kreirali „login.component.ts” skriptu. Uključili smo Angular.js forme, ostale klase, korisnički servis i komponente koje su nam potrebne. Definirali smo rutu, predložak za našu HTML formu i klasu za prijavu korisnika. U njoj smo definirali model za unos podataka za prijavu korisnika te provjeru ispravnosti e-mail adrese tako da poštuje ispravnost strukture e-mail adresa.

```
<form #loginForm="ngForm" (ngSubmit)="loginForm.valid && onSubmit(loginForm)">
  <input type="text" name="email" #email="ngModel" [(ngModel)]="model.email"
    placeholder="Email" [pattern]="emailRegex"
    required [ngClass]="{'invalid-textbox' :loginForm.submitted && !email.valid }">
  <div *ngIf="loginForm.submitted && email.errors?.pattern">
    <label class="validation-message">Neispravna e-mail adresa.</label>
  </div>
  <input type="password" name="password" #password="ngModel" [(ngModel)]="model.password"
    placeholder="Password"
    required minlength="4" [ngClass]="{'invalid-textbox' :loginForm.submitted &&
        !password.valid }">
  <div *ngIf="loginForm.submitted && password.errors?.minlength">
    <label class="validation-message">Lozinka mora sadržavati minimalno 6 simbola.</label>
  </div>
  <input type="submit" value="Prijava">I
</form>
<!-- Ispis grešaka -->
<div class="alert" *ngIf="serverErrorMessages">
  {{serverErrorMessages}}
</div>
```

Slika 52 "login.component.html" skripta

Na kraju kao što vidimo u kodu iznad kreirali smo HTML formu za prijavu korisnika. Struktura i logika je ista kao i kod registracije korisnika samo što koristimo definirani model za prijavu. Obrazac za promjenu lozinke sastoji se od polja u koje unosimo staru lozinku te dva polja za unos nove lozinke. Dva polja za unos nove lozinke su definirana kako bi korisnik dva puta upisao novu lozinku i bio siguran da nije pogriješio pri upisu.

10. Korisnički profil



Slika 53 Stranica korisničkog profila

Kada se korisnik prijavi u sustav kroz navigaciju se može kretati po aplikaciji i pristupati stranicama koje su zaštićene. Prva stranica kojoj će korisnik pristupati biti će početna „Home" stranica na kojoj će se prikazivati popis knjiga koje će korisnici pretraživati. Prije nego li kreiramo sam prikaz knjiga i početnu stranicu napraviti ćemo stranicu koja prikazuje korisničke podatke prijavljenog korisnika. Također na toj stranici korisnik će moći i promijeniti lozinku po želji. Osim podataka korisnika i opcije za promjenu lozinke korisnik će na ovoj stranici vidjeti i popis svih dodanih knjiga, no prije ove komponente ćemo kreirati komponentu za dodavanje knjiga. U ovom koraku ćemo napraviti dio ove stranice gdje prikazujemo korisničke podatke i opciju za promjenu lozinke.

10.1. Serverski dio komponente

Prvo krećemo s izradom serverske strane dijela ove aplikacije. U sve rute dalje koje budemo definirali u „index.router.js" sada ćemo uključiti i provjeru kako bismo znali je li korisnik prijavljen. Ukoliko nije korisnik će se preusmjeriti na stranicu za prijavu.

```
1. router.get('/userData', jwtHelper.verifyJwtToken, ctrlUser.userData);
```

Slika 54 Ruta za prikaz korisničkog profila

U liniji koda iznad smo definirali rutu za prikaz korisničkih podataka. Uključili smo joj kontroler „user" komponente te metodu „userData" koju ćemo kreirati kao i metodu za provjeru tj. verifikaciju tokena.

```
1. module.exports.userData = (req, res, next) => {
2.     User.findOne({ _id: req._id },
3.         (err, user) => {
4.             if (!user) {
5.                 return res.status(404).json({ status: false, message: 'Korisnik nije pronađen' });
6.             } else {
7.                 Rating.find({ user_id: req._id }).exec().then(function (ratings) {
8.                     if (ratings) {
9.                         let ratedBooksIds = [];
10.                        let rating = ratings;
11.                        console.log(rating.length);
12.                        for (let i = 0; i < rating.length; i++) {
13.                            ratedBooksIds.push(rating[i].book_id);
14.                        }
15.                        console.log(ratedBooksIds);
16.                        return res.status(200).json({
17.                            status: true,
18.                            user: _.pick(user, ['id', 'name', 'surname', 'email', 'groups']),
19.                            ratedBooks: ratedBooksIds
20.                        });
21.                    }
22.                });
23.            }
24.        }
25.    );
26. }
```

Slika 55 metoda „userData"

U kodu iznad je metoda „userData" koja omogućuje prikaz korisničkih podataka. Pomoću metode „findOne" pretražujemo bazu podataka po korisničkom ID-ju (engl. *Identifier*, skraćeno ID) koji je dohvaćen prilikom prijave korisnika te pokušavamo vratiti

korisničke podatke. Ukoliko nešto nije u redu server će vratiti odgovor da korisnik nije pronađen i kod 404. Ova metoda dalje traži ocjenjene knjige od strane korisnika, ukoliko ih ima. To ćemo objasniti u drugom dijelu. Kao odgovor servera ukoliko uspješno dohvati podatke vraćaju se ocjene knjiga te korisnički podatci koji su nam u ovom trenutku bitni.

10.2. Klijentski dio komponente

Krećemo na sljedeći korak i kreiramo klijentski dio aplikacije za ovu komponentu. U „app" mapi kreiramo mapu „user-data". Na taj način smo kreirali novu komponentu Angular.js dijela. Kreiramo ju jednostavnije upisivanjem u terminal naredbu „ng g c user-data".

```
1.  import { Component, OnInit } from '@angular/core';
2.  import { UserService } from '../shared/user.service';
3.  import { BookService } from '../shared/book.service';
4.  import { Router } from '@angular/router';
5.  import { NgForm } from '@angular/forms';
6.  import { TextFilterPipe } from '../shared/text-filter.pipe';
7.  import { StringFilterPipe } from '../shared/string-filter.pipe';
8.  import { BookRatingComponent } from '../book-rating/book-rating.component';
9.  @Component({
10.     selector: 'app-user-data',
11.     templateUrl: './user-data.component.html'
12. })
```

Slika 56 Uključivanje svih potrebnih biblioteka, klasa u skriptu "user-data.component.ts"

U kodu iznad u skriptu „user-data.component.ts" uključili smo potrebne servise koje ćemo koristiti te ostale potrebne komponente. Ono što nas od njih trenutno zanima je servis „UserService", forma „NgForm", „Router" te HTML obrazac koji ćemo koristiti za prikaz vizualnog dijela.

```
export class UserDataComponent implements OnInit {
    data;
    successMessage: string;
    showSuccessMessage: boolean;
    serverErrorMessages: string;
    userDataLoaded: boolean;
```

Slika 57 "UserDataComponent" klasa 1/3

U kodu iznad vidimo početni dio nove klase „UserDataComponent”. Definirali smo svojstva klase. Svojstvo „data” u sebi će sadržavati podatke o korisničkom profilu te također smo definirali poruke koje će vratiti server kao odgovor te „userDataLoaded” koji vraća je li istina da smo dobili korisničke podatke. Dalje ćemo u klasi definirati metode potrebne za prikaz podataka i promjenu lozinke. Definiramo i konstruktor klase.

```
constructor(private userService: UserService,
    private bookService: BookService, private router: Router)
}
ed: boolean;
```

Slika 58 "UserDataComponent" klasa 2/3

Dodijelili smo mu sve potrebne servise i „Router” klasu koja nam omogućuje manipuliranje i definiranje ruta.

```
ngOnInit() {
    this.getProfileData();
}
```

Slika 59 "UserDataComponent" klasa 3/3

U kodu iznad u metodi koja se poziva pri inicijalizaciji pozivamo metodu „getProfileData”.

```
getProfileData() {
    this.userService.getUserData().subscribe(
        res => {
            this.userService.selectedUser = res['user'];
            this.data = res['user'];
            this.userDataLoaded = true;
        },
        err => {
            console.log(err);
        }
    );
}
```

Slika 60 "getprofileData“ metoda

„getProfileData” metoda će nam vratiti podatke prijavljenog korisnika pomoću metode „userService” iz servisa. Pomoću nje dohvaćene podatke ćemo prikazati na definiranoj ruti. Prikazati će se podatci tj. svojstvo „selectedUser” klase „UserService” kojeg smo prethodno definirali kod registracije.

```
getUserData() {
  return this.http.get(environment.apiBaseUrl + '/userData');
}
```

Slika 61 "getUserdata“ metoda

Vratiti ćemo se dalje u skriptu „user-data.component.js” te definirati metodu za promjenu lozinke.

```
onChangePassword(form: NgForm) {
    this.userService.changePassword(this.data.id, form.value).subscribe(
        res => {
            setTimeout(() => {
                form.resetForm();
                this.successMessage = 'Lozinka je promijenjena!';
            }, 3000);
        },
        err => {
            if (err) alert(err.error.message);
            if (err.status === 422) {
                this.serverErrorMessages = err.error.join('<br/>');
            } else {
                this.serverErrorMessages = 'Nešto nije u redu.';
            }
        }
    );
}
```

Slika 62 "onChangePassword" metoda

Metoda poziva metodu iz servisa „changePassword” te će pri uspjehu osvježiti podatke u profilu. Bilo da uspješno ili neuspješno obavimo proces promjene lozinke server će vratiti odgovarajuću poruku. Metoda „resetForm” se nalazi također u skripti „user-data.component.js”.

```
resetForm(form: NgForm) {
    this.userService.selectedUser = {
        id: '',
        name: '',
        surname: '',
        email: '',
        password: ''
    };
    form.resetForm();
    this.serverErrorMessages = '';
}
```

Slika 63 NgForm "selectedUser"

Ona će osvježiti ažurirane podatke korisnika. Metodu „changePassword” definiramo u „user” servisu.

```
changePassword(id: string, passwords) {
  return this.http.post(environment.apiBaseUrl + '/changePassword/'
    + id, { current: passwords.currentpwd, newPassword: passwords.newpwd,
    newConfirm: passwords.confirmnewpwd} );
}
```

Slika 64 "changePassword" metoda

Ovom metodom dohvaćena je upisana trenutna i nova lozinka korisnika te se nova lozinka kriptira i ažurira u bazi podataka. Dalje ćemo kreirati HTML kod za vizualni stranice u Internet pregledniku. Uređujemo skriptu „user-data.component.html”.

```
<div class="row">
    <div class="col-5">
        <div class="table-info">
            <h2>Korisnički podatci</h2>
            <table *ngIf="data" class="table-fill">
                <tbody>
                    <tr>
                        <td>Ime</td>
                        <td>{{data.name}}</td>
                    </tr>
                    <tr>
                        <td>Prezime</td>
                        <td>{{data.surname}}</td>
                    </tr>
                    <tr>
                        <td>e-mail</td>
                        <td>{{data.email}}</td>
                    </tr>
                </tbody>
            </table>
        </div>
        <!-- Poruka o uspjehu -->
        <div class="alert" *ngIf="successMessage">
            {{successMessage}}
        </div>
    </div>
    <div class="col-5 off-1">
```

Slika 65 HTML skripta "user-data" komponente 1/2

```
28.         <form #changePasswordForm="ngForm" (ngSubmit)="changePasswordForm.valid
29.             && onChangePassword(changePasswordForm)">
30.             <h2>Change Password</h2>
31.             <label>trenutna lozinka</label>
32.             <input type="password" [(ngModel)]="password.current"
33.                 #current="ngModel" name="currentpwd" placeholder="Trenutna lozinka"
34.                 required [ngClass]="{'invalid-textbox' :changePasswordForm.submitted
35.                     && !group.valid }">
36.             <label>Nova lozinka</label>
37.             <input type="password" [(ngModel)]="password.newPassword"
38.                 #newPassword="ngModel" name="newpwd" placeholder="Nova lozinka"
39.                 required [ngClass]="{'invalid-textbox' :changePasswordForm.submitted
40.                 && !group.valid }">
41.             <label>Potvrdite novu lozinku</label>
42.             <input type="password" [(ngModel)]="password.newConfirm"
43.                 #newConfirm="ngModel" name="confirmnewpwd"
44.                 placeholder="Nova lozinka" required
45.                 [ngClass]="{'invalid-textbox' :changePasswordForm.submitted && !group.valid }">
46.             <br />
47.             <input type="submit" value="Promijeni lozinku">
48.         </form>
49.     </div>
50. </div>
```

Slika 66 HTML skripta "user-data" komponente 2/2

HTML skripta prikazuje dohvaćene korisničke podatke koje je vratio server te je kreirana forma za promjenu lozinke. Korisnik unosi staru lozinku te dva puta novu lozinku kako ne bi došlo do pogrešnog upisa nove lozinke. Nakon potvrde server će vratiti poruku o uspješnosti promjene lozinke.

11. Knjige

Sljedeći korak biti će kreiranje mogućnosti dodavanja knjiga prijavljenim korisnicima. Knjiga će se dodavati na način da na stranici za dodavanje knjiga korisnik popuni formu gdje će unijeti podatke o knjizi. Biti će potrebno upisati naziv knjige, ime autora, kratki opis i dodijeliti žanr. Žanrovi su unaprijed definirani. Korisnik putem „select“ elementa odabire jedan od ponuđenih žanrova. Nakon što smo definirali model korisnika sada će biti potrebno kreirati i model za knjige kako bi se podatci o njima mogli spremati u bazu podataka.

11.1. Serverski dio aplikacije

11.1.1. Model

Kreiramo model u mapi „models” za knjige. Dodajemo kod u skriptu „book.model.js”

```
const mongoose = require('mongoose');
var bookSchema = new mongoose.Schema({
    userId: {
        type: String,
        required: true
    },
    bookName: {
        type: String,
        required: 'Ime je obavezno.'
    },
    authorName: {
        type: String,
        required: 'Autor je obavezan.'
    },
    description: {
        type: String,
        required: 'Opis je obavezan'
    },
    genre: {
        type: String,
        required: 'Žanr je obavezan'
    },
    ratingCount: {
        type: Number
    },
    ratingValue: {
        type: Number
    },
    ratingAverage: {
        type: Number
    },
```

Slika 67 Model Knjige 1/2

```
32.     userId: String,
33.     name: String,
34.     surname: String,
35.     text: String,
36.     date: Date
37. }],
38.     saltSecurity: String
39. });
40. mongoose.model('Book', bookSchema);
```

Slika 68 Model knjige 2/2

Kako će se moći komentirati i ocjenjivati knjige osim što će sadržavati osnovna svojstava poput ID-ja korisnika koji je knjigu dodao, imena knjige, autora, opisa i žanra, dodajemo i polje za brojanje komentara, ocjena te prosječnu ocjenu svih ocjena koje su korisnici dodijelili knjizi. Osim toga dodajemo i komentare koji imaju svoja svojstva poput ID-ja korisnika, teksta komentara te datuma dodavanja komentara. Na kraju dodajemo i sigurnosni atribut. Za komentiranje knjige ćemo kreirati posebni model kako bi se mogao pohraniti svaki komentar dodijeljen od bilo kojeg korisnika.

```
const mongoose = require('mongoose');
var ratingSchema = new mongoose.Schema({
    user_id: {
        type: String,
        index:true
    },
    book_id: {
        type: String,
        index:true
    },
    value: {
        type: Number
    }
},
{ collection : 'Ratings' });
mongoose.model('Rating', ratingSchema);
```

Slika 69 Model za ocjene

Skriptu u kojoj se nalazi shema za ocjenjivanje nazvali smo „rating.model.js”. Nakon kreiranja ova dva modela završili smo kreiranje svih modela u našoj aplikaciji.

11.1.2. Dodavanje, ažuriranje i prikaz knjige

U sljedećem koraku definirati ćemo kontroler pomoću kojega ćemo upravljati s knjigama. Skriptu kontrolera nazivamo „book.controller.js” te je smještamo u mapu „controllers”.

```
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');
const Book = mongoose.model('Book');
const Rating = mongoose.model('Rating');
```

Slika 70 Primjer uključivanja dodatnih biblioteka i klasa u kontroler

Uključujemo u kontroler biblioteku koja omogućuje komuniciranje s bazom podataka „Mongoose” te za rad s tokenima i novokreirane modele za knjigu i ocjenjivanje „Book” i „Rating”.

Prvo ćemo kreirati metodu za dodavanje knjige.

```
module.exports.addBook = (req, res, next) => {

    var token = req.headers['authorization'].split(' ')[1];
    if (token != null) {
        var actorID = jwt.decode(token)._id;
        console.log('ACTOR ID: ', actorID);
        var book = new Book({
            userId: actorID,
            bookName: req.body.bookName,
            authorName: req.body.authorName,
            description: req.body.description,
            genre: req.body.genre,
            ratingValue: 0,
            ratingCount: 0,
            ratingAverage: 0,
            comments: []
        });
        console.log('ADD BOOK REQ.BODY: ', req.body);
        if (req.body._id) {
            book._id = req.body._id;
            Book.findByIdAndUpdate({ _id: req.body._id }, { $set: { bookName: req.body.bookName,
                authorName: req.body.authorName, description: req.body.description,
                genre: req.body.genre}}, (err, doc) => {
                if (!err) {
                    Book.find({}, function (err, books) {
                        console.log(books);
                        if (err) {
                            return res.status(404).json({ status: false, message: 'Lista je prazna' });
                        } else {
                            return res.status(200).json({ status: true, books: books });
                        }
                    });
```

Slika 71 "book.controller.js" skripta 1/2

```
                    });
                } else {
                    return res.status(404).json({ status: false, message: 'Knjiga nije ažurirana' });
                }
            });
        } else {
            book.save((err, doc) => {
                if (!err) {
                    Book.find({}, function (err, books) {
                        console.log(books);
                        if (err) {
                            return res.status(404).json({ status: false, message: 'Lista je prazna' });
                        } else {
                            return res.status(200).json({ status: true, books: books });
                        }
                    });
                } else {
                    if (err.code == 11000) {
                        res.status(422).send(['Ova knjiga vec postoji']);
                    } else {
                        return next(err);
                    }
                }
            });
        }
    } else {
        return res.sendStatus(403);
    }
}
```

Slika 72 "book.controller.js" skripta 2/2

Kreirali smo metodu gdje smo definirali objekt „book” preko kojega ćemo kreirati knjigu tj. inicijalizirali smo objekt „Books” iz naših modela. Kako bi definirali tko je autor knjige preko dekodiranja tokena smo definirali ID korisnika. Definirane su poruke grešaka koje se mogu pojaviti ukoliko npr. knjiga već postoji u bazi podataka ili je jednostavno spremanje bilo neuspješno. Ovu metodu ćemo koristiti i kod ažuriranja knjige. Ukoliko smo potvrdili podatke knjige pokrenuti će se metoda ažuriranja. Sam poziv dodavanja knjige ili ažuriranja biti će drukčiji u Angular.js dijelu aplikacije ovisno o tome koju akciju smo pozvali.

Za prikaz knjige kreirana je metoda „getBookData”.

```
module.exports.getBookData = (req, res) => {
    Book.aggregate([{
        $match: {
            _id: req.params.id
        }
    }, {
        $lookup: {
            from: "users",
            localField: "userId",
            foreignField: "userId",
            as: "name"
        }
    }]);
    Book.findOne({ _id: req.params.id },
        (err, book) => {
            if (!book) {
                return res.status(404).json({ status: false, message: 'Korisnik nije pronađen' });
            } else {
                return res.status(200).json({ status: true, book: book });
            }
        }
    );
}
```

Slika 73 Metoda "getBookData"

Ovom metodom jednostavno dohvaćamo podatke bilo koje knjige te će server kao odgovor vratiti te podatke. Agregiramo s modelom „users” kako bismo mogli dohvatiti podatke korisnika koji su komentirali knjigu. Same podatke knjige dohvaćamo preko njezinog ID-ja. Server će vratiti tražene podatke u odgovoru ili će vratiti poruku greške.

11.1.3. Ocjenjivanje knjige

Ocjenjivanje knjige nešto je složenije obzirom da trebamo prilikom svakog ocjenjivanja izračunati prosječnu ocjenu za knjigu koja se ocjenjuje.

```
module.exports.rateBook = (req, res, next) => {
    console.log(req.body);
    var ratingModel = new Rating();
    ratingModel.book_id = req.body.book_id;
    ratingModel.user_id = req.body.user_id;
    ratingModel.value = req.body.rating;
    var new_book_count = req.body.book_count + 1;
    var new_book_value = Number(req.body.book_value) + Number(req.body.rating);
    var new_book_average = new_book_value / new_book_count;
    console.log(new_book_count, new_book_value, new_book_average);
    Rating.findOne({ user_id: ratingModel.user_id, book_id: ratingModel.book_id })
        .exec()
        .then(function (success) {
            if (success) { console.log("return"); return res.status(422).json({
                status: false, message: 'Knjiga je vec ocijenjena!' }); }
            else {
                ratingModel.save((err, doc) => {
                    if (!err) {
                        Rating.find({}, function (err, rating) {
                            if (err) {
                                return res.status(404).json({ status: false,
                                    message: 'Ocjena ne postoji' });
                            } else {
                                Book.update(
                                    { _id: ratingModel.book_id },
                                    {
                                        $inc: { 'ratingCount': 1,
                                            'ratingValue': ratingModel.value },
                                        $set: { 'ratingAverage': new_book_average }
                                    }
                                )
                                    .exec().then(function (success) {
                                        console.log(success);
                                        return res.status(200).json({ status: true,
                                            rating: success,
```

Slika 74 "rate-book.component,ts" skripta 1/2

```
                                        newRating: new_book_average });
                                })
                                .catch(
                                    function (err) {
                                        console.log(err);
                                        return res.status(422).json({ status: false,
                                            message: 'Azuriranje ocjene nije uspjelo' });
                                    });
                            }
                        });
                    } else {
                        if (err.code == 11000) {
                            res.status(422).send(['Knjiga je već ocijenjena']);
                        } else {
                            return next(err);
                        }
                    }
                });
            }
        });
}
```

Slika 75 "rate-book.component,ts" skripta 2/2

Metoda dohvaća potrebne podatke. Zanima nas ID knjige, korisnika i sama ocjena. Vrši se izračun prosječne ocjene jednostavnim dijeljenjem te spremamo u bazu novu prosječnu ocjenu. Ukupni broj ocjena do sada povećavamo za jedan kako bi smo pri ponavljanju ove metode mogli računati novu prosječnu ocjenu. Svaki korisnik može ocijeniti samo jednom jednu knjigu. Ukoliko dođe do greške ili neuspješnog ocjenjivanja server će vratiti odgovarajuću poruku.

```
module.exports.libraryData = (req, res, next) => {
    Book.find({}, function (err, books) {
        if (err) {
            return res.status(404).json({ status: false, message: 'Lista je prazna' });
        } else {
            return res.status(200).json({ status: true, books: books });
        }
    });
}
```

Slika 76 "libraryData" pretraga knjiga

11.1.4. Pretraga knjiga

Metoda „libraryData” dohvaća knjige te nam služi kao tražilica. Kao odgovor server vraća podatke knjiga koje pretražujemo. Ukoliko ne postoji ni jedna knjiga također dobivamo odgovarajuću poruku.

```
module.exports.getBooks = (req, res, next) => {
    var token = req.headers['authorization'].split(' ')[1];
    if (token != null) {
        var actorID = jwt.decode(token)._id;
        Book.find({
            userId: actorID
        }, function (err, books) {
            if (err) {
                return res.status(404).json({ status: false, message: 'Nemate svojih knjiga' });
            } else {
                return res.status(200).json({ status: true, books: books });
            }
        });
    } else {
        return res.sendStatus(401);
    }
}
```

Slika 77 "getBooks" pretraga knjiga

Metodom „getBooks” dohvaćamo knjige prijavljenog korisnika na njegovom profilu. Metoda funkcionira slično kao prethodna metoda samo ovdje imamo dodatni parametar ID korisnika kojeg dohvaćamo preko tokena kako bi znali koje su knjige od korisnika.

Pretposljednja metoda serverskog dijela aplikacije je metoda za dodavanje komentara.

```
module.exports.commentBook = (req, res, next) => {
    var token = req.headers['authorization'].split(' ')[1];
    console.log('COMMENT PARAMS: ', req.body);
    var bookID = req.body.bookId;
    var commentText = req.body.text.comment;
    if (token != null) {
        var actorID = jwt.decode(token)._id;
        var comment = { userId: actorID, name: '', surname: '',
            text: commentText, date: new Date() };
        User.findById({ _id: actorID }, function (err, user) {
            comment.name = user.name;
            comment.surname = user.surname;
            if (err) {
                return res.status(404).json({
                    status: false, message: 'Korisnik nije pronađen'});
            } else {
                Book.findByIdAndUpdate({ _id: bookID },
                    { $push: { comments: comment } }, { new: true },
                    function (err, doc) {
                    if (err) {
                        console.log(err);
                        return res.status(404).json({ status: false,
                        message: 'Knjiga nije pronađena' });
                    } else {
                        return res.status(200).json(doc);
                    }
                });
            }
        });
    } else {
        return res.sendStatus(401);
    }
}
```

Slika 78 Metoda "commentBook"

Preko tokena i ID-ja knjige spremamo komentar. Metoda dohvaća sve te podatke te ih sprema u model. Također po modelu spremamo i ime i prezime korisnika koji je upisao

komentar. Ukoliko dođe do pogreške prilikom komentiranja server će vratiti odgovarajuću poruku. Komentari se spremaju metodom ažuriranja „Book” modela.

11.1.5. Brisanje knjige

Metoda za brisanje knjige ja zadnja metoda u serverskom dijelu aplikacije.

```
module.exports.deleteBook = (req, res, next) => {
    var token = req.headers['authorization'].split(' ')[1];
    var bookID = req.body.bookId;
    if (token != null) {
        var actorID = jwt.decode(token)._id;
        Book.remove({ _id: bookID }, function (err) {
            if (err) {
                console.log(err);
                return res.status(500).send(err);
            } else {
                Book.find({
                    userId: actorID
                }, function (err, books) {
                    if (err) {
                        return res.status(404).json({ status: false,
                            message: 'Nemate knjiga' });
                    } else {
                        return res.status(200).json({ status: true,
                            books: books });
                    }
                });
            }
        });
    } else {
        return res.sendStatus(401);
    }
}
```

Slika 79 Metoda "deleteBook"

Preko provjere tokena i ID-ja knjige traži se odgovarajuća knjiga te ju brišemo iz baze podataka. Ukoliko dođe do greške server će vratiti pouku o neuspjehu u odgovoru.

11.1.6. Rute

Osim metoda potrebno je i definirati ostale rute u „index.router.js" skripti za novokreirane metode. Sve rute su definirane na način da smo im dodijelili zadnji segment URL adrese na engleskom jeziku te smo uključili provjeru tokena pomoću „jwtHelpera". Dodijeljen je i kontroler koji obrađuje podatke.

```
1. router.get('/libraryData', jwtHelper.verifyJwtToken, ctrlBook.libraryData);
2. router.get('/getBooks', jwtHelper.verifyJwtToken, ctrlBook.getBooks);
3. router.post('/addBook', jwtHelper.verifyJwtToken, ctrlBook.addBook);
4. router.post('/rateBook', jwtHelper.verifyJwtToken, ctrlBook.rateBook);
5. router.get('/bookData/:id', jwtHelper.verifyJwtToken, ctrlBook.getBookData);
6. router.post('/commentBook', jwtHelper.verifyJwtToken, ctrlBook.commentBook);
7. router.post('/deleteBook', jwtHelper.verifyJwtToken, ctrlBook.deleteBook);
```

Slika 80 "index.router.js" skripta - definiranje ruta

11.2. Klijentski dio aplikacije

Nakon što smo završili sa serverskim dijelom aplikacije krećemo s izradom klijentskog dijela aplikacije gdje ćemo obuhvatiti nekoliko komponenti. Redom ćemo kreirati komponente za kreiranje tj. dodavanje knjige, ocjenjivanje knjige, pregled knjige te komponentu naslovne. Naslovna stranica će imati tražilicu za sve dodane knjige koja će ih prikazivati poredane po prosječnim ocjenama silazno bilo da su filtrirane po žanru ili pretraživane po ključnoj riječi.

11.2.1. Servis

Prvo ćemo kreirati u „shared" mapi „book.model.ts".

```
1. import { text } from "@angular/core/src/render3/instructions";
2. export class Book {
3.     _id: string;
4.     bookName: string;
5.     authorName: string;
6.     description: string;
7.     genre: string;
8.     ratingCount :number;
9.     ratingValue:number;
10.    ratingAverage:number;
11.    comments: Array<{userId: string, text: string, date: Date}>;
12. }
```

Slika 81 Model knjige u servisu

U ovoj skripti smo definirali objekt „Book” sa svim njegovim parametrima kako bismo u komunikaciji sa serverskim dijelom aplikacije i serverom koji komunicira s bazom podataka mogli obrađivati podatke. Također ćemo kreirati skriptu „book.service.ts” u kojoj ćemo definirati metode koje će obrađivati potrebne podatke.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { environment } from '../../environments/environment';
import { Book } from './book.model';
@Injectable({
  providedIn: 'root'
})
export class BookService {
  selectedBook: Book = {
    _id: '',
    bookName: '',
    authorName: '',
    description: '',
    genre: '',
    ratingCount: null,
    ratingValue: null,
    ratingAverage: null,
    comments: null
  };
  noSecureHeader = { headers: new HttpHeaders({ 'NoSecure': 'True' }) };
  constructor(private http: HttpClient) { }
  //HttpMethods to manage books
  addBook(book: Book) {
    console.log(book);
    console.log(this.selectedBook);
    return this.http.post(environment.apiBaseUrl + '/addBook', book);
  }
  updateBook(book: Book, bookId) {
    book._id = bookId;
    return this.http.post(environment.apiBaseUrl + '/addBook', book);
  }
```

Slika 82 "book.service.ts" skripta 1/2

```
32.   getBookData(id) {
33.     console.log('bookid', id);
34.     return this.http.get(environment.apiBaseUrl + '/bookData/' + id);
35.   }
36.   getLibraryData() {
37.     return this.http.get(environment.apiBaseUrl + '/libraryData');
38.   }
39.   rateBook(item) {
40.     console.log(item.book_id);
41.     return this.http.post(environment.apiBaseUrl + '/rateBook', item);
42.   }
43.   getMyBooks() {
44.     return this.http.get(environment.apiBaseUrl + '/getBooks');
45.   }
46.   commentBook(comment) {
47.     return this.http.post(environment.apiBaseUrl + '/commentBook', comment);
48.   }
49. }
```

Slika 83 "book.service.ts" skripta 2/2

Kreirana je klasa „BookService” kojoj su definirana svojstva. Onemogućen je pristup neprijavljenim korisnicima za sve metode u ovoj komponenti aplikacije. Metoda „addBook” proslijediti će podatke koje smo joj dodijelili. Proslijedili smo joj objekt „Book” sa svim potrebnim podatcima knjige koju smo upisali. Podatci će biti proslijeđeni serverskom dijelu aplikacije preko definirane rute te će se dalje u tom dijelu aplikacije obrađivati. Metoda „updateBook” radi na sličan način i prosljeđuje podatke knjige serverskom dijelu aplikacije metodi „addBook” no s ID parametrom te će serverski dio aplikacije izvršiti ažuriranje knjige, a ne kreiranje.

„getBookData” dohvaća podatke o knjizi iz baze podataka preko ID-ja knjige. Definirana je ruta putem koje se metodom serverskog dijela aplikacije „getLibraryData” dohvaćaju kreirane knjige. Ova metoda je u svrsi dohvata podataka knjiga za tražilicu. Metoda „rateBook” ažurira dio objekta knjige za ocjenjivanje. Prosljeđuje podatke o ocjenjivanju serverskom dijelu aplikacije. „getMyBooks” metoda dohvaća knjige prijavljenog korisnika koje će se prikazati na njegovom profilu. Metoda „commentBook”

komunicira s metodom serverskog dijela aplikacije za komentiranje gdje joj prosljeđuje podatke komentara.

11.2.2. Rute

U skriptu „routes.ts” ćemo dodati rute za preostale komponente aplikacije vezane za rad s knjigama.

```
import { Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
import { RegisterComponent } from './user/register/register.component';
import { LoginComponent } from './user/login/login.component';
import { UserDataComponent } from './user-data/user-data.component';
import { AddBookComponent } from './add-book/add-book.component';
import { HomePageComponent } from './home-page/home-page.component';
import { ViewBookComponent } from './view-book/view-book.component';
import { SecureGuard } from './secure/secure.guard';
export const appRoutes: Routes = [
    {
        path: 'register', component: UserComponent,
        children: [{ path: '', component: RegisterComponent }]
    },
    {
        path: 'login', component: UserComponent,
        children: [{ path: '', component: LoginComponent }]
    },
    {
        path: 'userdata', component: UserDataComponent,
            canActivate: [SecureGuard]
    },
    {
        path: '', redirectTo: '/login', pathMatch: 'full'
    },
    {
        path: 'addbook', component: AddBookComponent,
            canActivate: [SecureGuard]
    },
    {
        path: 'viewbook/:bookid', component: ViewBookComponent,
            canActivate: [SecureGuard]
    },
```

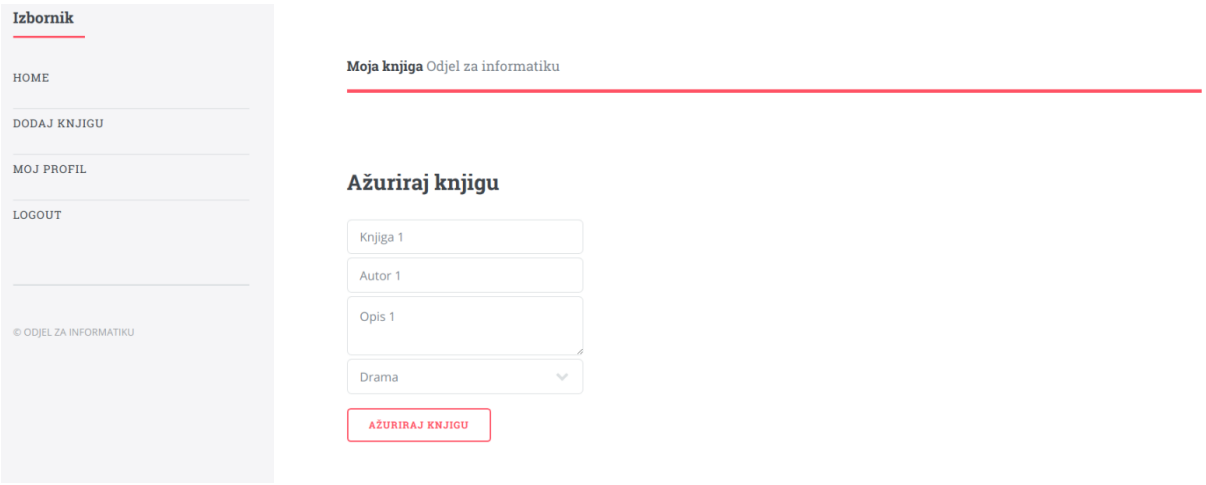
Slika 84 "routes.ts" skripta 1/2

```
34.     {
35.         path: 'editBook/:bookid', component: AddBookComponent,
36.             canActivate: [SecureGuard]
37.     },
38.     {
39.         path: 'home', component: HomePageComponent,
40.             canActivate: [SecureGuard]
41.     }
42. ];
```

Slika 85 "routes.ts" skripta 2/2

Uključili smo i potrebne klase koje su potrebne kako bi naše rute radile ispravno.

11.2.3. Dodavanje i ažuriranje knjiga



Slika 86 Stranica za ažuriranje knjiga

Prvo kreiramo komponentu za dodavanje knjiga „add-book”. Kao i u prošlim kreiranjima komponenti mapa u kojoj se nalaze skripte dobiva naziv po komponenti. U skriptu „add-book.component.ts” dodajemo svu potrebnu logiku.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import { UserService } from '../shared/user.service';
3.  import { BookService } from '../shared/book.service';
4.  import { Router } from '@angular/router';
5.  import { NgForm } from '@angular/forms';
6.  import { ActivatedRoute } from '@angular/router';
7.  import { Book } from '../shared/book.model';
8.  @Component({
9.    selector: 'app-add-book',
10.   templateUrl: './add-book.component.html',
11.   styleUrls: ['../../assets/css/main.css']
12. })
13. export class AddBookComponent implements OnInit {
14.   serverErrorMessages: string;
15.   libraryData;
16.   bookId;
17.   title: string = '';
18.   isEdit: boolean = false;
19.   constructor(private userService: UserService, private bookService:
20.         BookService, private router: Router, private route: ActivatedRoute) { }
```

Slika 87 "add-book.component.ts" skripta 1/3

```
21.  ngOnInit() {
22.    this.bookId = this.route.snapshot.paramMap.get("bookid");
23.    if (this.bookId != null) {
24.      this.title = 'Ažuriraj knjigu';
25.      this.isEdit = true;
26.      this.bookService.getBookData(this.bookId).subscribe(
27.        res => {
28.          console.log(res);
29.          this.bookService.selectedBook = res['book'];
30.        },
31.        (err) => {
32.          console.log(err);
33.        }
34.      );
35.    } else {
36.      this.title = 'Dodaj knjigu';
37.      this.bookService.selectedBook = new Book;
38.    }
39.  }
40.  onAddBook(form: NgForm) {
41.    if (this.isEdit) {
42.      this.bookService.updateBook(form.value, this.bookId).subscribe(
43.        res => {
44.          this.libraryData = res['books'];
45.          form.resetForm();
46.        },
47.        err => {
48.          console.log(err);
49.          this.serverErrorMessages = 'Nešto nije u redu';
50.        }
51.      );
52.    } else {
53.      this.bookService.addBook(form.value).subscribe(
54.        res => {
55.          console.log(res);
```

Slika 88 "add-book.component.ts" skripta 2/3

```
          this.libraryData = res['books'];
          form.resetForm();
        },
        err => {
          if (err) console.log(err);
          if (err.status === 422) {
            this.serverErrorMessages = err.error.join('<br/>');
          } else {
            this.serverErrorMessages = 'Nešto nije uredu.';
          }
        }
      );
    }
  }
  resetForm(form: NgForm) {
    this.bookService.selectedBook = {
      _id: '',
      bookName: '',
      authorName: '',
      description: '',
      genre: '',
      ratingCount: null,
      ratingValue: null,
      ratingAverage: null,
      comments: null
    };
    form.resetForm();
    this.serverErrorMessages = '';
  }
}
```

Slika 89 "add-book.component.ts" skripta 3/3

Uključili smo potrebne biblioteke i definirali smo klasu komponente „AddBookComponent”. Kao i svaka klasa u sebi sadrži svojstva te konstruktor sa servisima. Metodom „ngOnInit” dohvaćamo sve potrebne podatke knjige obzirom da

može biti riječ o dodavanju ili ažuriranju knjige. Metodu ažuriranja knjige ćemo pozvati ukoliko nam je proslijeđen njezin ID „bookid”. Ukoliko nema ID-ja pozvati će se metoda za dodavanje nove knjige. Definirana je i forma sa svojim parametrima koji definiraju polja za unos podataka. Ova skripta pomoću svojih metoda komunicira sa serverskim dijelom aplikacije te na taj način ostvaruje pristup bazi podataka i omogućava nam dodavanje i ažuriranje knjiga. Naslov na HTML stranici će biti definiran po varijabli „title” koja će po pozivu ažuriranja ili dodavanja dobiti odgovarajuću tekstualnu vrijednost.

U „add-book.component.html” ćemo dodati formu za unos i ažuriranje knjige

```
<form #addBookForm="ngForm" (ngSubmit)="addBookForm.valid && onAddBook(addBookForm)">
  <h2>{{title}}</h2>
  <div class="add-book">
      <input type="text" #bookName="ngModel"
        [(ngModel)]="bookService.selectedBook.bookName" name="bookName"
          placeholder="Ime knjige" required
            [ngClass]="{'invalid-textbox' :addBookForm.submitted && !bookName.valid }">
      <div *ngIf="addBookForm.submitted && !bookName.valid">
          <label class="validation-message">Ovo polje je obavezno.</label>
      </div>
      <input type="text" #authorName="ngModel"
        [(ngModel)]="bookService.selectedBook.authorName" name="authorName"
          placeholder="Autor" required
             [ngClass]="{'invalid-textbox' :addBookForm.submitted && !authorName.valid }"
      <div *ngIf="addBookForm.submitted && !authorName.valid">
          <label class="validation-message">Ovo polje je obavezno.</label>
      </div>
      <textarea type="text" #description="ngModel"
        [(ngModel)]="bookService.selectedBook.description" name="description"
          placeholder="Opis" required
            [ngClass]="{'invalid-textbox' :addBookForm.submitted && !description.valid }"
      </textarea>
      <div *ngIf="addBookForm.submitted && !description.valid">
          <label class="validation-message">Ovo polje je obavezno.</label>
      </div>
      <select #genre="ngModel" [(ngModel)]="bookService.selectedBook.genre" name="genre">
          <option>Drama</option>
          <option>Horor</option>
          <option>Poezija</option>
          <option>Roman</option>
      </select>
      <div *ngIf="addBookForm.submitted && !genre.valid">
          <label class="validation-message">Ovo polje je obavezno.</label>
      </div>
  </div>
```

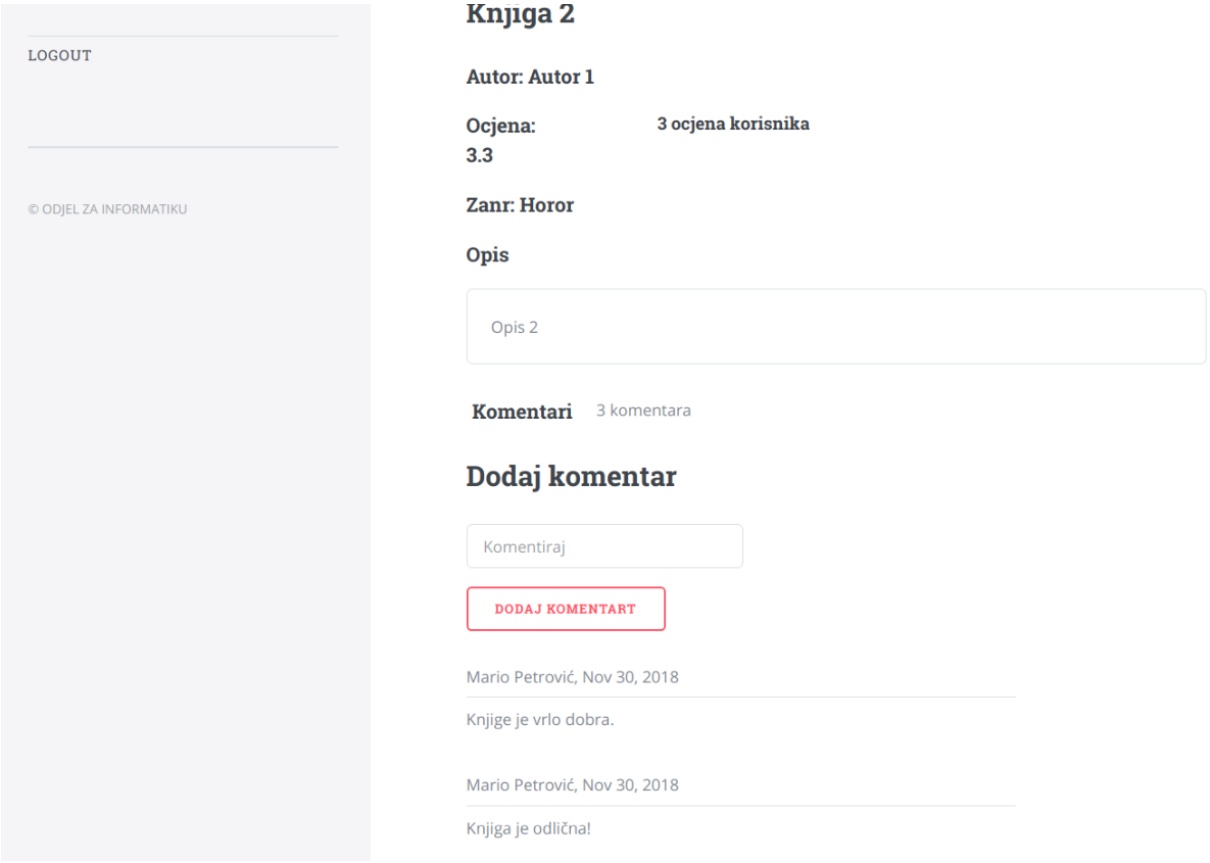
Slika 90 "add-book.component.html" skripta 1/2

```
36.    <input type="submit" value="{{title}}">
37. </form>
```

Slika 91 "add-book.component.html" skripta 2/2

HTML skripta započinje naslovom koji je definiran u „add-book.component.ts” . Definiramo sva polja za unose podataka i odabir žanra u padajućem izborniku . Sva polja su obavezna i forma je definirana kao i prethodne. Ukoliko je u modelu „selectedBook” imamo podatke knjige koje se dohvaćaju preko ID-ja knjige oni se prikazuju i možemo ih ažurirati.

11.2.4. Pregled knjige



Slika 92 Stranica za pregled knjige

Za pregled knjige kreirati ćemo komponentu „view-book”.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import { ActivatedRoute } from '@angular/router';
3.  import { UserService } from '../shared/user.service';
4.  import { BookService } from '../shared/book.service';
5.  import { NgForm } from '@angular/forms';
6.  import { Router } from '@angular/router';
7.  @Component({
8.    selector: 'app-view-book',
9.    templateUrl: './view-book.component.html',
10.   styleUrls: ['../../assets/css/main.css']
11. })
12. export class ViewBookComponent implements OnInit {
13.   private bookId;
14.   public bookData: Array<{ _id: string, authorName: string,
15.     description: string, genre: string, bookName: string,
16.     ratingAverage: number, ratingCount: number, ratingValue: number,
17.     v: number, comments: Array<{ userId: string, name: string,
18.     surname: string, text: string, date: any }> }> = [];
19.   public comment = '';
20.   public userDataLoaded: boolean;
21.   constructor(private userService: UserService, private bookService:
22.     BookService, private router: Router, private route: ActivatedRoute) { }
23.   ngOnInit() {
24.     this.bookId = this.route.snapshot.paramMap.get("bookid");
25.     this.bookService.getBookData(this.bookId).subscribe(
26.       res => {
27.         console.log(res);
28.         this.bookData = res['book'];
29.
30.         this.userService.getUserData().subscribe(
31.           res => {
32.             this.userService.selectedUser = res['user'];
33.             this.userService.ratedBooks = res['ratedBooks'];
```

Slika 93 "view-book.component.ts" skripta 1/2

```
34.             this.userDataLoaded = true;
35.           },
36.           err => {
37.             console.log(err);
38.           }
39.         );
40.       },
41.       (err) => {
42.         console.log(err);
43.       }
44.     );
45.   }
46.   addComment(form: NgForm) {
47.     this.bookService.commentBook({ bookId: this.bookId,
48.         text: form.value }).subscribe(
49.       res => {
50.         console.log(res);
51.         this.bookData['comments'] = res['comments'];
52.       },
53.       err => {
54.         console.log(err);
55.       }
56.     );
57.   }
58. }
```

Slika 94 "view-book.component.ts" skripta 2/2

U kodu iznad koji je napisan u skripti „view-book.component,ts” uključili smo potrebne klase, biblioteke i servise. Pomoću ove komponente ćemo prikazati u Internet pregledniku podatke knjige koju je korisnik odlučio pogledati. Definirali smo svojstva klase „ViewBookComponent” gdje imamo „bookId” varijablu koja nosi vrijednost ID-ja knjige preko kojega ćemo dohvatiti podatke iz baze podataka. Dohvaćamo sve podatke knjige te njezinu prosječnu ocjenu i sve komentare. Preko „userService” dohvaćamo korisnika te ocjene knjige. Također pozivamo i formu za dodavanje komentara gdje će svaki korisnik moći unijeti komentar za knjigu.

```
<div class="row">
  <div class="col-8">
    <h2>{{bookData.bookName}}</h2>
    <h3>Autor: {{bookData.authorName}}</h3>
    <div class="row">
      <div class="col-3">
        <!-- <h3>Ocjena: {{bookData.ratingAverage | number: '1.0-1'}}</h3> -->
        <h3>Ocjena: <app-book-rating [book]="bookData"
            *ngIf="userDataLoaded"></app-book-rating></h3>
      </div>
      <div class="col-9">
        <h4>{{bookData.ratingCount}} Ocjena korisnika</h4>
      </div>
    </div>
    <h3>Zanr: {{bookData.genre}}</h3>
  </div>
</div>
<div class="row">
  <div class="col-8">
    <h3>Opis</h3>
    <div class="box">
      <p>{{bookData.description}}</p>
    </div>
  </div>
</div>
<div class="row">
  <div class="col-4">

  </div>
</div>
<div class="row">
  <h3>Komentari</h3>
  <p>{{bookData.comments.length}} komentara</p>
</div>
```

Slika 95 "view-book.component.html" skripta 1/2

```
<div class="row">
  <form #commentForm="ngForm" (ngSubmit)="commentForm.valid
        && addComment(commentForm)">
    <h2>Dodaj komentar</h2>
    <div class="add-group">
      <input type="text" #commentText="ngModel"
        [(ngModel)]="comment" name="comment" placeholder="Comment" required
        [ngClass]="{'invalid-textbox' :commentForm.submitted && !commentText.valid }">
      <div *ngIf="commentForm.submitted && !commentText.valid">
        <label class="validation-message">This field is required.</label>
      </div>
    </div>
    <input type="submit" value="Add Comment">
  </form>
</div>
<div class="row">
  <div class="col-6 col-12-small">
    <ul class="alt" *ngFor="let com of bookData.comments">
      <li>{{com.name}} {{com.surname}}, {{com.date | date}}</li>
      <li>{{com.text}}</li>
    </ul>
  </div>
</div>
```

Slika 96 "view-book.component.html" skripta 2/2

Pomoću HTML koda komponente za pregled knjige „view-book.component.html” prikazujemo dohvaćene podatke o knjizi uključujući i komentare. Napisana je forma za unos komentara. Forma će vratiti podatke pri akciji slanja u „view-book.component.ts” koji će dalje obraditi podatke te ih preko servis metode koju smo prethodno objasnili poslati podatke metodi serverskog dijela aplikacije koja će spremiti komentar u model odgovarajuće knjige.

11.2.5. Ocjenjivanje knjiga

Ocjenjivanje knjige je nešto složenije od samih komentara te je kreirana komponenta „book-rating”.

```
import { Component, OnInit, Input } from '@angular/core';
import { BookService } from '../shared/book.service';
import { UserService } from '../shared/user.service';
import { Book } from '../shared/book.model';
@Component({
  selector: 'app-book-rating',
  templateUrl: './book-rating.component.html',
  styleUrls: ['../../assets/css/main.css']
})
export class BookRatingComponent implements OnInit {
  @Input() book: Book;
  isRateOpen: boolean;
  isRated: boolean;
  ratedBooks: String[];
  constructor(private bookService: BookService,
    private userService: UserService) { }
  ngOnInit() {
    console.log(this.book);
    this.ratedBooks = this.userService.ratedBooks;
    this.isRated = this.ratedBooks.includes(this.book._id);
  }
  toggleRateOption(value) {
    this.isRateOpen = !this.isRateOpen;
    if (value) console.log(value);
  }
  rateBook(rating) {
    let item = {
      book_count: this.book.ratingCount,
      book_value: this.book.ratingValue,
      user_id: this.userService.selectedUser.id,
      book_id: this.book._id,
      rating: rating
    }
```

Slika 97 "rate-book.component.ts" skripta 1/2

```
34.     console.log(item);
35.     this.bookService.rateBook(item).subscribe(
36.       res => {
37.         this.book.ratingAverage = res['newRating'];
38.         this.isRated = true;
39.         console.log(res);
40.       },
41.       err => {
42.         if (err) alert(err.error.message);
43.       }
44.     );
45.   }
46. }
```

Slika 98 "rate-book.component.ts" skripta 2/2

Komponenta dohvaća ocjene knjiga iz modela „ratedBooks” te za pojedinu knjigu provjerava je li već ocjenjena. Metoda „toggleRateOption” omogućuje prikaz padajućeg izbornika kod prikaza knjige. On prikazuje popis mogućih ocjena od kojih korisnik može odabrati jednu kako bi ocijenio knjigu. Metoda „rateBook” se poziva prilikom ocjenjivanja knjige i prosljeđuje joj se ocjena te se vrši novi izračun prosječne ocjene za knjigu.

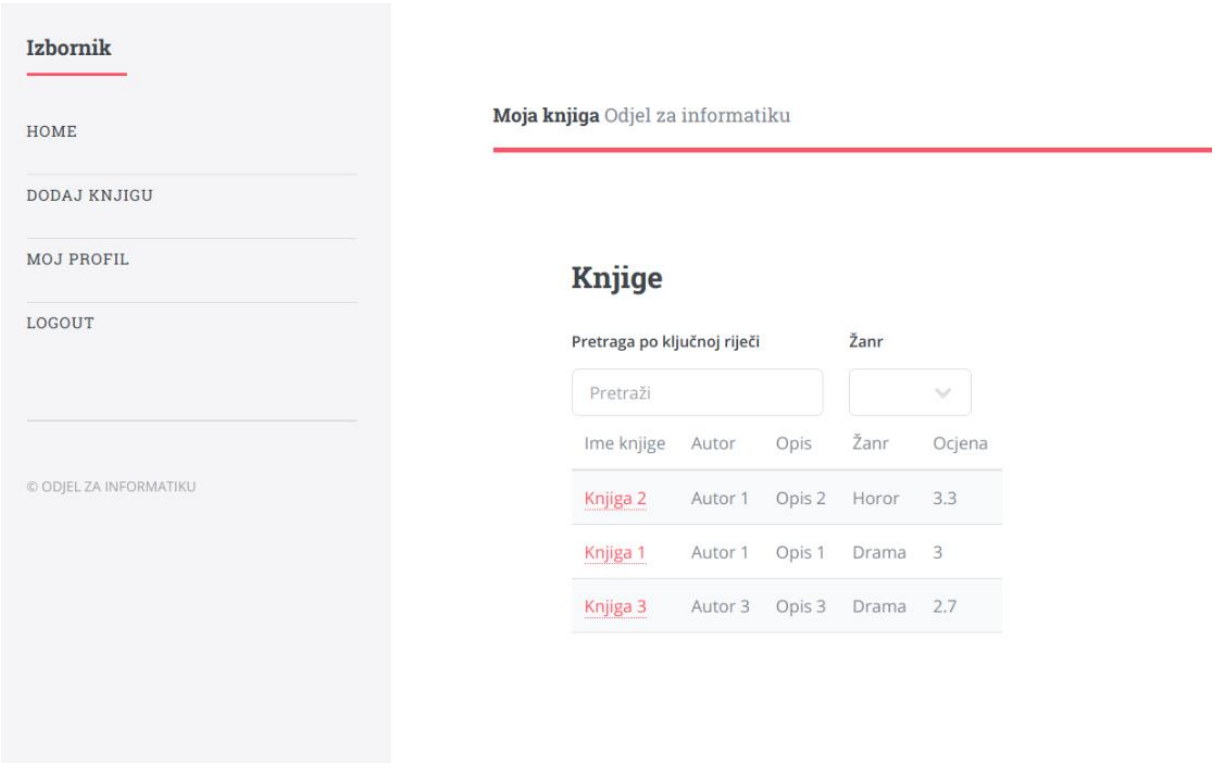
```
<div class="rate-process" *ngIf="!isRated">
    <button type="button" (click)="toggleRateOption(book.rating)"
        class="button small">Ocijeni</button>
    <div class="rate-options" *ngIf="isRateOpen">
      <!-- <label>Rate</label> -->
      <select #rate="ngModel" [(ngModel)]="bookRating" name="rate"
        style="width: 50px; margin: 5px">
        <option>1</option>
        <option>2</option>
        <option>3</option>
        <option>4</option>
        <option>5</option>
      </select>
      <button type="button" (click)="rateBook(bookRating)"
        class="button primary small">Potvrdi</button>
    </div>
</div>
<div *ngIf="isRated">{{book.ratingAverage | number: '1.0-1'}}</div>
```

Slika 99 HTML forma za ocjenjivanje knjige

U „book-rating.component.html” je definiran HTML kod za komponentu ocjenjivanja. Korisnik odabirom ocjene i klikom na „Potvrdi” gumb poziva metodu „rateBook”. Ukoliko je knjiga ocijenjena od strane prijavljenog korisnika on to više neće moći učiniti kako bi se spriječilo beskonačno ocjenjivanje.

11.2.6. „Home“ komponenta

Ova komponenta će imati ulogu za prikaz početne stranice prijavljenog korisnika. Na njoj će se nalaziti tražilica knjiga gdje ćemo ih moći pretražiti po ključnoj riječi i/ili filtrirati po žanru. Lista knjiga je sortirana silazno po prosječnoj ocjeni knjige. Iz popisa rezultata knjiga možemo pristupiti svakoj knjizi preko poveznice koja je ime knjige.



Slika 100 Stranica "Home"

```
1. import { Component, OnInit } from '@angular/core';
2. import { UserService } from '../shared/user.service';
3. import { BookService } from '../shared/book.service';
4. import { Router } from '@angular/router';
5. import { TextFilterPipe } from '../shared/text-filter.pipe';
6. import { StringFilterPipe } from '../shared/string-filter.pipe';
7. import { BookRatingComponent } from '../book-rating/book-rating.component';
8. @Component({
9.   selector: 'app-home-page',
10.  templateUrl: './home-page.component.html',
11.  styleUrls: ['../../assets/css/main.css']
12. })
13. export class HomePageComponent implements OnInit {
14.   libraryData;
15.   libraryData2;
16.   data;
17.   userDataLoaded: boolean;
18.   isRateOpen: boolean;
19.   filteredLibrary = '';
20.   genreFilterValue = 'All';
21.   genreFilterValue2 = 'All';
22.   constructor(private userService: UserService,
23.     private bookService: BookService, private router: Router) { }
24.   ngOnInit() {
25.     this.isRateOpen = false;
26.     this.userService.getUserData().subscribe(
27.       res => {
28.         this.userService.selectedUser = res['user'];
29.         this.data = res['user'];
30.         this.userService.ratedBooks = res['ratedBooks'];
31.         this.userDataLoaded = true;
32.       },
33.       err => {
34.         console.log(err);
```

Slika 101 "home-page.component.ts" skripta 1/2

```
      }
    );
    this.bookService.getLibraryData().subscribe(
      res => {
        this.libraryData = res['books'];
        this.libraryData2 = res['books'];
        this.libraryData2.sort(function(rate1, rate2) {
          if(rate1.ratingAverage > rate2.ratingAverage) {
            return -1;
          } else if(rate1.ratingAverage < rate2.ratingAverage) {
            return 1;
          } else {
            return 0;
          }
        });
      },
      err => {
        console.log(err);
      }
    );
  }
}
```

Slika 102 "home-page.component.ts" skripta 2/2

U mapi „home-page” novokreirane komponente kreirali smo skriptu „home-page.component.ts” čiji je kod iznad. Definirali smo svojstva, konstruktor i metode koje su potrebne kako bismo izvršili potreban prikaz i dohvat podataka za tražilicu knjiga. Dohvaćamo podatke korisnika preko metode „getUserData” gdje dohvaćamo prijavljenog korisnika i ocijene knjiga. Metodom „getLibraryData” dohvaćamo pretražene knjige sortirane po rangu. Metoda sadrži dvije varijable koje sadrže vrijednosti pretrage. „libraryData” nosi vrijednost ključne riječi pretrage dok „libraryData2” vrijednost odabranog žanra po kojemu filtriramo pretragu. Kako bi pretraga bila moguća kreirane su pomoćne skripte za pretragu u „shared” folderu te su uključene na početku skripte. Svaka omogućuje pretragu po jednoj od dvije varijable pretrage. „libraryData” varijabla je tekstualni tip podatka „text” te kreiramo skriptu za pretragu po njoj „text-filter.pipe.ts”.

```
1.  import { Pipe, PipeTransform } from '@angular/core';
2.  @Pipe({
3.    name: 'textFilter'
4.  })
5.  export class TextFilterPipe implements PipeTransform {
6.    transform(values: any[], filterString: string,
7.      propName:string): any {
8.
9.      if(values.length === 0)
10.       return values;
11.       const resultArray = [];
12.     for(const item of values){
13.       if(item[propName].includes(filterString)){
14.         resultArray.push(item);
15.       }
16.     }
17.     return resultArray;
18.   }
19. }
```

Slika 103 "text-filter.pipe.ts" skripta

Naziv „pipe-a” je „textFilter”. Klasa „TextFilterPipe” produžuje klasu „PipeTransform”. Metoda „transform” prima objekt bilo kojeg tipa. „values” je polje koje se prosljeđuje funkciji te mu iz klase „UserDataComponent” dodjeljena varijabla „filteredLibrary” kao polje. Imamo i filter unos koji je je korisnik upisao i svojstvo „name”. Svojstvo objekta je polje. Naziv knjige filtriramo gdje je ime knjige jednako unesenom filter imenu knjige. „Pipe” transformira polje za pretragu i transformira podatke za prikaz.

Skripta „string-filter.pipe.ts” radi na sličan način samo što je ovoga puta obrađujemo tekstualni tip podatka „string”.

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'stringFilter'
})
export class StringFilterPipe implements PipeTransform {
  transform(values: any, filterString: string, propName: string): any {
    if(values.length === 0 || !filterString)
      return values;
    const resultArray = [];

    for(const item of values){
      if(filterString === 'All'){
        resultArray.push(item);
      }
      else if(item[propName] === filterString && filterString !== ''){
        resultArray.push(item);
      }

      }

    return resultArray;
  }
}
```

Slika 104 "string-filter.pipe.ts" skripta

Proces je isti samo pretražujemo knjige po žanru i gdje je vrijednost odabranog žanra jednaka žanru knjige koji joj je dodijeljen.

Ono što je preostalo je kreirati HTML dio „home-page” komponente.

```
<div class="flex-between container">
  <div class="flex-column">
    <div *ngIf="libraryData">
      <h2>Knjige</h2>
      <div class="row gtr-uniform">
        <div class="col-xs-5">
          <label>Pretraga po ključnoj riječi</label>
          <input type="text" [(ngModel)]="filteredLibrary" placeholder="search">
        </div>
        <div class="col-xs-7">
          <label>Žanr</label>
          <select #genre="ngModel" [(ngModel)]="genreFilterValue" name="genre">
            <option>Drama</option>
            <option>Horor</option>
            <option>Poezija</option>
            <option>Roman</option>
          </select>
        </div>
      </div>
      <div class="table-wrapper">
        <table>
          <thead>
            <tr>
              <td>Ime knjige</td>
              <td>Autor</td>
              <td>Opis</td>
              <td>Žanr</td>
              <td>Ocjena</td>
            </tr>
          </thead>
          <tbody>
            <tr *ngFor="let book of libraryData2 |
            textFilter:filteredLibrary:'bookName' |
            stringFilter: genreFilterValue:'genre'">
```

Slika 105 "home-page.component.html" skripta 1/2

```
35.              <td><a [routerLink]="['/viewbook', book._id]"
36.                routerLinkActive="active">{{book.bookName}}</a></td>
37.              <td>{{book.authorName}}</td>
38.              <td>{{book.description}}</td>
39.              <td>{{book.genre}}</td>
40.              <td>{{book.ratingAverage | number: '1.0-1'}}</td>
41.            </tr>
42.          </tbody>
43.        </table>
44.      </div>
```

Slika 106 "home-page.component.html" skripta 2/2

Kreirali smo „home-page.component.html”. Kreirano je polje za unos ključne riječi za pretragu i padajući izbornik gdje odabiremo žanr po kojemu možemo filtrirati rezultate pretrage. Pretraga se može obaviti po jednom od dva načina ili kombinirano. Kao rezultat dobivamo listu knjiga sortiranu silazno po prosječnoj ocjeni. Upis u polje za unos ključne riječi pretrage ili odabir žanra pozivaju odmah pretragu te se ispod prikazuju dohvaćeni rezultati pretrage.

12. *LAMP* aplikacija

Tehnologiju same LAMP platforme smo prethodno objasnili. Izradi aplikacije u *MEAN* platformi prethodila je izrada iste aplikacije s istim mogućnostima i metodama na *LAMP* platformi. Kako bi se programiranje olakšalo i ubrzalo korišten je programski okvir Phalcon. Bazira se kao i Angular.js na MVC tehnologiji. Izradi aplikacije je prethodilo modeliranje te je izrađena relacijska baza podataka putem MySQL jezika. Izrađene su slične komponente kao i u MEAN aplikaciji te se svaka komponenta sastoji od kontrolera, modela i pregleda te servisa. Svrha ovog rada je pokazati prošireno znanje u izradi *web* aplikacija. Učenje izrade aplikacije na MEAN platformi i sama logika programiranja dosta se mogla iskoristiti i iz znanja programiranja u LAMPU-u. U izradi LAMP aplikacije nije se koristio dodatno JavaScript jezik kako bi se predočila prava razlika u funkcionalnosti ove dvije platforme. Dodavanjem JavaScript-a i dodatnih biblioteka poput Elastic Search-a može se postići gotovo ista brzina i funkcionalnost pretrage mapiranjem podataka te će raditi jednako brzo kao i pretraga u MEAN aplikaciji. To nije učinjeno kako bi se predočila čim oštrija razlika ove dvije platforme. Korištena je PHP7.2 inačica PHP jezika. Kao lokalni server koji je pokretao aplikaciju je korišten Nginx. Aplikacija ima potpuno iste mogućnosti vezano za sve komponente aplikacije koje su iste u obije verzije aplikacije.

13. „AB“ testiranje i usporedba MEAN i LAMP aplikacija

Izrada aplikacija u ove dvije tehnologije u jednom dijelu je slična dok se u nekim dijelovima potpuno razlikuje. Samo programiranje je teže u MEAN platformi kako je riječ o novijoj tehnologiji. Samim time više je traženo jer manji broj programera vlada ovom tehnologijom. JavaScript jezik mnogi programeri ne vole zbog nekih njegovih specifičnosti no svakako je prednost MEAN tehnologije jer je potrebno znati samo jedan programski jezik u odnosu na LAMP tehnologiju. Navesti ćemo prednosti i nedostatke na primjeru izrađene dvije aplikacije pri samom radu.

Tablica 1 Usporedba MEAN - LAMP

MEAN	**LAMP**
teži programski jezik	jednostavniji programski jezik
potrebno znanje samo jednog programskog jezika	potrebno znanje više programskih jezika
struktura koda je složenija	struktura koda je jednostavnija
instalacija je jednostavnija	složenija instalacija
konfiguracija jednostavnija	konfiguracija složenija
kod složeniji i razgranatiji	kod je jednostavniji i manje razgranat
slabije trpi pogreške u kodu	može pretrpiti manje pogreške u kodu (npr. HTML)
teže se dodaju vanjske biblioteke	jednostavnije se dodaju vanjske biblioteke
sigurnost aplikacije kompliciranija	sigurnost aplikacije jednostavnija

Postoje i razlike u radu ovih dviju aplikacija. Tu svakako ima prednost MEAN platforma obzirom da joj se asinkron i „real-time” rad podrazumijeva što joj je ogromna prednost.

Brzina učitavanja stranica:

- prijava-MEAN: 113 ms

- prijava-lamp: 411 ms

Testiranjem utvrđujem kako MEAN platforma ima veću brzinu kod prijave korisnika. Brzina je gotovo četiri puta veća.

Sljedeća prednost je kvalitetnija sigurnost same aplikacije gdje se na svakoj stranici provjerava je li korisnik prijavljen preko „tokena". Također pri svakom unosu u bazu vrši se dodatna provjera je li korisnik prijavljen. Teže je probiti bazu podataka na MEAN platformi obzirom na dodatne sigurnosti. Kod MySQL baze podataka treba biti vrlo pažljiv. Sama validacija prijavljenog korisnika vrši se preko sesijske varijable no pristup bazi je moguć i bez prijave kod registracije i prijave korisnika. Tu postoji prostor i načini upisivanja dodatnog SQL koda koji može ugroziti bazu podataka. Iz tog razloga je svakako bolje koristiti programski okvir poput Phalcona gdje je sigurnost na visokoj razini.

Pretraga u LAMP aplikaciji radi na način da u formu za pretragu koja se sastoji od tekstualnog unosa i/ili odabira žanra te nakon što smo potvrdili pretragu dobivamo rezultate i stranica se učitava ponovno s rezultatima. Kod MEAN platforme pretraga je „real-time" tj. pravovremena te dok mi upisujemo ključni pojam pretrage već nam dolaze rezultati te je gumb „pretraži" u potpunosti nepotreban kod pretraga u ovoj platformi. Pretraga je time dinamičnija te izuzetno brža obzirom da radi na noSql bazi podataka i također samo slanje i potvrda podataka u LAMP tehnologiji su sporiji i od strane korisnika jer zahtjeva više mehaničkih radnji.

Kod komentara imamo sličnu situaciju no ona je još lošija u odnosu na pretragu. Kod komentiranja u MEAN aplikaciji čim komentiramo i potvrdimo komentar on je u istom trenutku vidljiv dok je u LAMP tehnologiji potrebno potvrditi komentar te ponovno učitati stranicu kako bi on bio vidljiv.

Ocjene funkcioniraju na isti način te je sama usporedba ista. Jedina je razlika u tome što je u LAMP-u u relacijskoj bazi jednostavnije dohvatiti neki podatak ili skup podataka te ga obraditi u smislu veza. Jasno da se može to napraviti u MEAN-u no nešto je kompliciranije.

Baza podataka MongoDB radi na puno brži način te je ovdje MEAN svakako u prednosti. LAMP zahtjeva relacijsku bazu podataka koju je teže složiti i strukturirati. Obzirom da se u MongoDB radi o objektima njih možemo strukturirati po želji te im dodijeliti razna svojstva sa svojim svojstvima tj. atributima. U MySQL bazi potrebno je čim više normalizirati bazu i definirati veze preko ključeva. Nemamo mogućnost tablici izravno definirati podtablicu unutar nje nego ćemo strukturu i hijerarhiju definirati preko veza. Također samo korištenje MySQL baze podataka iziskuje puno više znanja i iskustva u odnosu na mongoDB gdje je dovoljno naučiti osnovnu sintaksu i rad s objektima.

Kod uključivanja dodatnih biblioteka prednost ima LAMP gdje preko jedne poveznice i jednostavnog dodavanje biblioteke možemo iz bilo kojeg dijela aplikacije pristupiti biblioteci. Kod MEAN platforme nešto je malo kompliciraniji sam proces gdje je svakoj skripti potrebno uključiti potrebnu biblioteku. Za dodavanje dodatnih JavaScript skripti je poprilično kompliciranije u MEAN-u i potrebno ih je instalirati dok kod LAMP-a je dovoljno dodati npr. na „index.php” tj. glavnoj skripti aplikacije dodati poveznicu.

Css predložak bilo je lakše primijeniti na LAMP platformi gdje se bez problema koriste njegove klase. Kako Angular.js dodaje svoje klase svim HTML elementima one u konačnici znaju utjecati na sam izgled i korištenje CSS-a.

U konačnici smo kvalitetniji proizvod dobili s aplikacijom na MEAN platformi obzirom na sve prednosti koje su navedene. Samo programiranje je bilo kompliciranije i teže obzirom da je znanje bilo višestruko manje u odnosu na znanje LAMP platforme.

14. Zaključak

MEAN platforma je skup tehnologija pomoću kojih se izrađuju web aplikacije. Kao novija tehnologija koja omogućuje po prvi puta u ovoj skupini tehnologija korištenje samo jednog programskog jezika dočekana je s velikim interesom u IT sektoru. Obzirom da svi noviji internet preglednici podržavaju JavaScript pojavila se ideja da se isti taj jezik koristi i sa serverske strane aplikacije. Uspjeh u ovom poduhvatu prvenstveno je razvio Node.js. Relacijske baze podataka u nekim slučajevima nisu bile adekvatno rješenje obzirom na brzinu izvođenja upita nad njima u slučajevima gdje imamo ogromnu količinu podataka. Kako se pojavila potreba za alternativnim rješenjem jedno od rješenja je bio i MongoDB.

Izrada aplikacija je olakšana programskim okvirima poput Angular.js-a za klijentski dio aplikacije i Express.js za serverski dio. Skup tih tehnologija koji su bazirani na JavaScriptu omogućio je kreiranje aplikacija na brz i jednostavniji način u odnosu na programiranje u Vanilla-i koja je naziv za čisti JavaScript programski jezik. Programskim okvirima JavaScript-a prethodili su jQuery i Ajax koji su olakšavali programiranje u ovom jeziku još dok se koristio u kombinaciji s nekim drugim programskim jezikom koji je komunicirao sa serverom.

Osim jednostavnijeg programiranja JavaScript je i prije stvaranja ove tehnologije omogućio u kombinaciji s drugim programskim jezicima simulirati „real-time“ aplikacije gdje krajnji korisnik nije imao potrebu npr. ponovnog učitavanja Internet stranica kako bi se podatci na njoj ažurirali. Napredovanje JavaScript jezika omogućilo je dodatne mogućnosti i kvalitetniju obradu i dohvat podataka što je u konačnici rezultiralo korištenjem samo tog jezika u čitavim aplikacijama.

Razvoj i mogućnosti ove tehnologije nagnale su me da ju naučim i savladam kako bih u budućnosti mogao koristiti u radu i ovu tehnologiju obzirom da neki pokazatelji govore da će ovaj tip tehnologija biti sve korišteniji u *web* programiranju. Iako se PHP jezik i dalje razvija kao i drugi jezici koji se koriste JavaScript uzima sve više maha u IT industriji te se danas njime pretežno programiraju i mobilne aplikacije. Ukoliko se savlada programiranje u MEAN tehnologiji ne bi trebao biti veliki problem kroz kraći period savladati i programiranje mobilnih aplikacija za koje isto postoji nekoliko programskih okvira.

Sam prelazak s LAMP na MEAN tehnologiju nije previše bolan obzirom da svaki ozbiljniji programer koji koristi LAMP već dobro poznaje sintaksu JavaScript-a. U nekim je trenutcima pomalo teško no postoji dobra dokumentacija počevši od službenih stranica razvojnih grupacija za ove tehnologije do drugih izvora koji su u velikom broju.

15. Prilozi

Uz rad se prilaže „zip“ arhiva „Prilozi“ u kojemu se nalazi MEAN i LAMP aplikacija.

15.1. Instalacija MEAN aplikacije

Potrebno je instalirati MongoDB kako je objašnjeno u poglavlju 5. Zatim kopirati folder “MEAN”. Nakon kopiranja datoteka I mapa potrebno je se pozicionirati u folder “nodejs” te zatim u “MojaKnjiga” te u oba foldera ponoviti istu radnju. Potrebno je na tim rutama otvoriti terminal te upisati naredbu “npm” install“.

15.2. Instalacija LAMP aplikacije

Potrebno je kopirati mapu “LAMP” na računalo. Zatim pokrenuti na ruti te mape komandu u terninalu “composer install” ili “composer update”.

Nakon istalacije paketa potrebno je instalirati Nginx. Kako bismo to učinili upisujemo naredbu u terminal “sudo apt-get install nginx”. Nakon toga potrebno je podesiti dozvole upisivanjem naredbe u terminal “sudo ufw allow 'Nginx HTTP'”. Sljedeći korak biti će konfiguracija Nginx-a. U mapi instaliranog Nginx-a koji se instalira u mapu “etc” pronaći ćemo mapu “sites-available”. U njemu kreiramo tekstualnu datoteku “work” te ćemo u nju upisati kod koji se nalazi u skripti „work.txt“ u mapi „LAMP“. Rute koda je potrebno podesiti ovisno o tome gdje je kod aplikacije kopiran. Nakon kreiranja potrebno je krirati I symbolic link naše Nginx konfiguracije. Upisujemo naredbu u terminal “sudo ln -s [“apsolutna putanja nginx instaliranog foldera”]/sites-available/work” [“apsolutna putanja nginx instaliranog foldera”]/sites-enabled/work”. Zatim u “etc” mapi na računalu potrebno je dodati liniju koda u skripti “hosts” Upisujemo liniju “127.0.0.1 work”. Nakon toga potrebno je ponovno pokrenuti Nginx. Upisujemo naredbu u terminal “sudo service nginx restart”.

Još što nam je preostalo je podesiti MySQL bazu podataka. Potrebno je instalirati MySQL. Potrebno je u terminal redom upisati sljedeće dvije naredbe. “sudo apt-get install mysql-server “

I “mysql_secure_installation”. Nakon toga potrebno je pronaći u glavnom “LAMP” mapi priloga SQL bazu podataka te ju pokrenuti. To ćemo napraviti upisivanjem naredbi u termina “CREATE DATABASE booksb” I “mysql -u [username] -p newdatabase < [database name].sql”. Za ime skripte baze podataka potrebno je upisati cijelu njezinu rutu. Zadnji korak je upisati MySQL podatke za spajanje na bazu podataka. Potrebno je urediti klasu “DbAdapter

“ I dodati joj odgovarajuće podatke. Klasa se nalazi u skripti “services.php” koja je u mapi “config”.

16. Literatura

[1] Deploying LAMP - Linux, Apache, MySQL Perl / PHP / Python - training course, » Well House Consultants «, [Mrežno].
Dostupno: http://www.wellho.net/course/alfull.html.
[Pokušaj pristupa 15 srpanj 2018.].

[2] The Editors of Encyclopaedia Britannica, » Britannica «, [Mrežno]. Dostupno: https://www.britannica.com/technology/SQL.
[Pokušaj pristupa 15 srpanj 2018.].

[3] David Hemmendinger, » Britannica «, [Mrežno]. Dostupno: https://www.britannica.com/technology/computer-programming-language/SGML#ref849870
[Pokušaj pristupa 15 srpanj 2018.].

[4] M. Zekić-Sušac, » JavaScript - Uvod, varijable, naredbe, petlje «, [Mrežno].
Dostupno: http://www.mathos.unios.hr/wp/wp2009-10/P8_Java.pdf.
[Pokušaj pristupa 15 srpanj 2018].

[5] Ilya Kantor, » An Introduction to JavaScript «, [Mrežno]. Dostupno: https://javascript.info/intro. [Pokušaj pristupa 15 srpanj 2018].

[6] Janet Swisher, Florian Scholz, Havvy Ryan, Eric Shepherd, Trevor Hobson, » Back to the Server: Server-Side JavaScript On The Rise «, [Mrežno].
Dostupno: https://javascript.info/intro. [Pokušaj pristupa 15 srpanj 2018].

[7] Mozilla, Langdon Holly, Rick Waldron i drugi » JavaScript «, [Mrežno].
Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[Pokušaj pristupa 17 srpanj 2018].

[8] Graham Cox » JavaScript Decorators: What They Are and When to Use Them «, [Mrežno].
Dostupno: https://www.sitepoint.com/javascript-decorators-what-they-are.
[Pokušaj pristupa 30 srpanj 2018].

[9] Mladen Jurak, Nela Bosner, Boris Muha » Predavanje 3 - Klase i objekti «, [Mrežno]. Dostupno:
https://web.math.pmf.unizg.hr/nastava/rp2/pred3/pred3.html
[Pokušaj pristupa 15 kolovoz 2018].

[10] Zachary Bergmann » JavaScript Pseudoclassical Pattern Classes and Subclasses (ES5 vs ES6)«, [Mrežno]. Dostupno:
https://medium.com/@zbbergma/javascript-pseudoclassical-instantiation-pattern-and-subclassing-es5-vs-es6-7397f3d8470c.
[Pokušaj pristupa 17 kolovoz 2018].

[11] Jeremy M Williams » Back to the Server: Server-Side JavaScript On The Rise «, [Mrežno]. Dostupno: https://medium.com/@jeremyvsjeremy/what-is-the-

mean-stack-9d11ae2cd384.
[Pokušaj pristupa 18 kolovoz 2018].

[12] Amos Q. Haviv, MEAN Web Development,Livery Place: Packt Publishing Ltd., 2014.

[13] Lukas Marx » Angular: Everything you need to get started«, [Mrežno]. Dostupno: https://malcoded.com/posts/angular-beginners-guide .
[Pokušaj pristupa 08 rujan 2018].

[14] HugeServer » How to install MEAN Stack (MongoDB, Express.js, Angular.js, Node.js) on Ubuntu 16.04 «, [Mrežno]. Dostupno: https://www.hugeserver.com/kb/install-mean-stack-ubuntu16/ .
[Pokušaj pristupa 20 srpanj 2018].