SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

VARAŽDIN

Tomislav Tomiek

APLIKACIJA ZA PREPORUKE (ENGL. RECOMMANDER SYSTEM) – KNJIGA POLUSTRUKTURIRANE BAZE PODATAKA - MONGODB + WEB SUČELJE

SEMINAR

TEORIJA BAZA PODATAKA

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Tomiek

Matični broj: 45999/17-R

Studij: Informacijsko i programsko inženjerstvo

APLIKACIJA ZA PREPORUKE (ENGL. RECOMMANDER SYSTEM) – KNJIGA POLUSTRUKTURIRANE BAZE PODATAKA - MONGODB + WEB SUČELJE

SEMINAR

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, lipanj 2022.

Tomislav Tomiek

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Kreiranje vlastite web aplikacije za preporuke knjige. Aplikacija za preporuke je izrađena od nule, gdje je bilo potrebno kreirati sve potrebne modele podataka, dohvat realnih podataka iz baze podataka (MongoDB koji je cijelo vrijeme podignut na njihovoj stranici), pa prikaz i manipulacija podataka na HTML stranici. Detaljno je objašnjeno kako se na jednostavni način kreira jednostavna web aplikacija uz korištenje jednog od plustrukturirane baze podataka, točnije MongoDB.

Ključne riječi: Web aplikacija, baze podataka, MongoDB, HTML, JavaScript, Express JS, Node.js.

Sadržaj

1. Opis aplikacijske domene

Tema ovoga projektnog rada je izrada web aplikacije za preporuke (engl. recommander system), točnije preporuka knjiga koje su dodane u bazu podataka. Baza podataka koju koristim je MongoDB. MongoDB se nalazi na oblaku (eng. cloud) i uvijek je dostupna. Kreirao sam besplatni račun na njihovoj stranici MongoDB – atlas, gdje sam morao kreirati i popuniti postavke novog Clustera. Cluster je zapravo sama Mongova baza podataka.

Admin aplikacije za preporuke može dodavati, ažurirati i brisati imena kategorija knjiga, koje će se koristiti kod dodavanja knjiga. Osim dodavanja kategorija admin može dodavati knjige koje će se prikazivati običnim registriranim korisnicima. Kod dodavanja knjiga admin popunjava opće informacije o knjizi. Knjige koje su se dodale, se mogu na jednostavni način ažurirati i brisati, te nakon unosa knjige admin može unijeti i pripadajuće autore knjige. Registrirani korisnik može pregledavati dodane knjige u bazu i ocjenjivati knjige. Ako korisnik pregledava određenu knjigu, prikazati će mu se preporuke ostalih knjiga na temelju ocjena ostalih korisnika i kategorije otvorene knjige.

Neregistrirani korisnik može se prijaviti ili registrirati se u aplikaciju.

Polustrukturirane baze podataka [1] kao što je MongoDB se najčešće koriste kod izrade raznih web stranice, pošto je navedena baza dosta fleksibilna i nije ograničena sa shemom. Prednosti polustrukturiranih baza je jednostavnost korištenja same baze.

Za izradu same web stranice koristio sam programski jezik JavaScript, kreiranje lokalnog server koristio sam Node.js [2], za prikaz HTML koda i API (eng. Application Programming Interface) koristio sam Express JS kao pozadinski web poslužitelj. Aplikacija se nalazi na mojemu GitHub računu, preko sljedećeg linka možete vidjeti web aplikaicju: `https://github.com/Tom1sl4v97/web_knjige`

2. Teorijski uvod

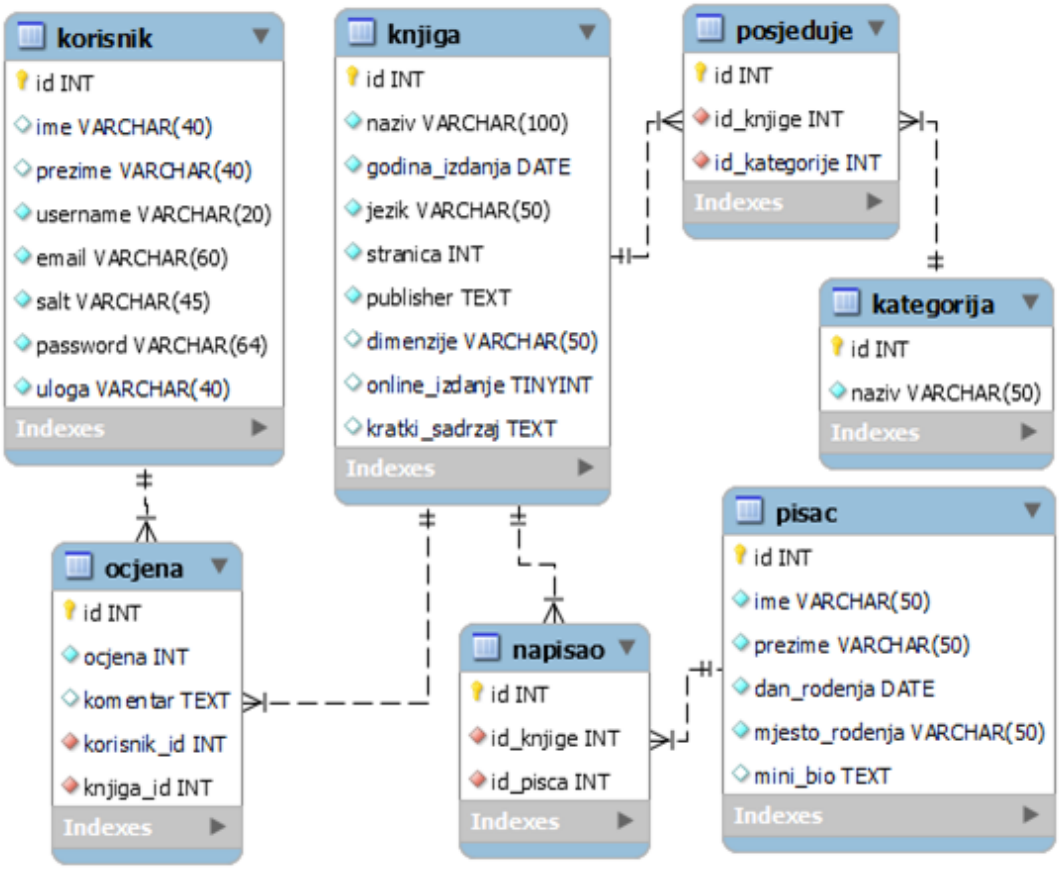
Polustrukturirane baze podataka se ne moraju predefinirati i nemaju određenu strukturu ili shemu. Definiranje strukture ili sheme je opcionalno, onda se može predefinirati. Podaci koji se unose u bazu podataka mogu i ne moraju posjedovati isti tip podataka, podaci se mogu unositi kako to korisnik želi. Polustrukturirane baze podataka redoslijed unosa podataka također nije bitan. Polustrukturirane baze koriste oznake i elemente koji se koriste za grupiranje podataka, gdje se opisuje kako će se koji podatak zapisati. Prema autoru Bogoviću [3] govori kako entiteti koji se nalaze u istoj grupi mogu ili ne moraju imati iste atribute i svojstva, te njihova veličina i tip podataka također može biti različiti.

Prednosti polustrukturiranih baza je njihova fleksibilnost i pohrana raznovrsnih podatak u bazu, gdje se grupirani podaci mogu imati različite atribute i tipove. Podaci su prenosivi, koji nisu predefinirani sa određenom shemom. Polustrukturirana baza podatak je nova tehnologija, koja još nije dosta raširena u svijetu, te njihova primjena može biti kompleksnija, što se tiče korištenja, jer posjeduje dinamičnu strukturu. Jedan od nedostatka je to što je pohrana takvih podatak u odnosu na druge baze podataka je novčano skuplja.

MongoDB je program baze podataka otvorenog koda koji se nalazi na više platformi (eng. cross-platform), orijentiran na dokumente. Koristi takozvanu NoSQL bazu podataka za pristup podacima. MongoDB koristi dokumenta slične JSON-u (eng. JavaScript Object Notation) s izborima shemama. MongoDB podržava pretraživanja lista, raspona i regularnih izraza. Upiti koji se proslijede mogu vratiti liste dokumenata, određene dijelove liste i ostale podatke.

3. Model baze podataka

ERA model praktične aplikacije za preporuke knjiga se nalazi na slici 1. ERA model prikazuje tablice i njihovo međusobno povezivanje standardnih bazi podataka, no pošto koristim MongoDB, koji ne zahtjeva predodređeno kreiranje tablica i njihovih tipova, nije bilo potrebno kreirati tablice na bazi podataka. Podaci se pohranjuju na temelju modela koje sam kreirao u programskom jeziku JS. Koristio sam besplatni paket mongosse koji služi za kreiranje shema. Sheme koristim kako bi podatke koje želim pohraniti u bazu podataka ili dobiti iz baze podataka pretvorio u ispravne oblike dokumenata koje MongoDB poznaje i koristi kod pohrane podataka.



Slika 1: Prikaz ERA modela

ERA model se sastoji od sveukupno sedam tablica, od kojih su četiri jaki entiteti i triju slabih entiteta, koje služe za povezivanje tablica više na više. Tablica korisnik služi za pohranu svih korisnikovih informacija o računu aplikacije za preporuke. Tablica ocjena služi za povezivanje tablica korisnika i knjige sa vezom više na više i ujedno još dodatno sadrži ocjenu i komentar knjige. Tablica knjiga sadrži podatke o unesenoj knjizi. Tablica posjeduje je pomoćna tablica za povezivanje veze više na više. Kategorije sadrže samo naziv kategorije knjige. Tablica pisac sadrži potrebne informacije o piscu knjige i posljednja tablica je tablica napisao koje je ponovno pomoćna tablica za povezivanje veze više na više.

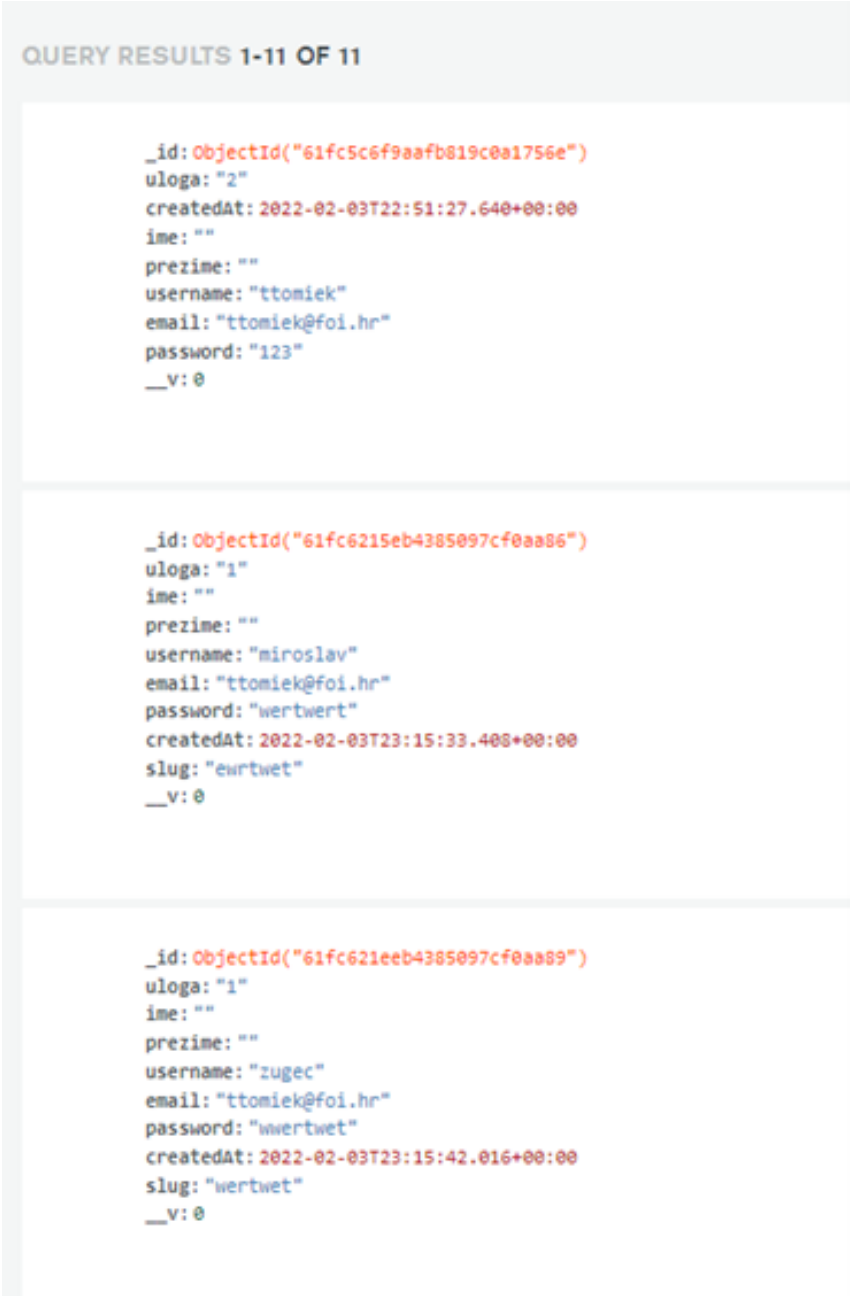
Primjerice model korisnik sadrži novokreiranu mongosse shemu, gdje sam definirao atribute ime, prezime, username, email, password, uloga, vrijeme kreiranja i slug (služi za bolji

prikaz id, koji se definira prije same sheme). Sljedeći dio koda prikazuje js datoteku za model korisnika.

Isječak kôda 1: Primjer kreiranja modela korisnika

```
const mongosse = require('mongoose')
const slugify = require('slugify')
const korisnikSchema = new mongosse.Schema({
    ime: {
        type: String,
    },
        prezime: {
        type: String,
    },
        username: {
        type: String,
        required: true,
    },
        email: {
        type: String,
        required: true,
    },
        password: {
        type: String,
        required: true,
    },
        uloga: {
        type: Number,
        default: 1
    },
        createdAt: {
        type: Date,
        default: Date.now
    },
    slug: {
        type: String,
        require: true,
        unique: true,
    }
})
korisnikSchema.pre('validate', function (next) {
    if (this.username) {
        this.slug = slugify(this.username, { lower: true, strict: true })
    }
    next()
})
module.exports = mongosse.model('Korisnik', korisnikSchema)
```

Osim modela za korisnika, kreirao sam ostale modele prema tablicama iz ERA modela prema istom principu. Slika 2. prikazuje zapisa podataka na MongoDB bazi podataka.



Slika 2: Prikaz zapisanih podataka na MongoDB bazi.

4. Implementacija

Praktični dio se sastoji od izrade web aplikacije sustava za preporuke knjiga. Kod izrade same aplikacije koristio sam Node.js, Express, MongoDB i pakete poput mongoose i session. Aplikaciju sam podijelio na views koji sadrži HTML kodove, routes koji upravlja sa logikom za pohranu i dohvaćanje podataka iz MongoDB baze i models koji sadrži model zapisanih podataka iz baze.

Prvobitna datoteka koja se otvara je server.js koji dohvaća sve ostale potrebne datoteke i povezuje se na MongoDB bazu podataka. Server.js otvara lokalni server, gdje se podiže stranica na port 5000. Web stranicu zatim možete otvoriti u web pregledniku preko sljedećeg linka localhost:5000. Pristup paketu sesija i postavljanje sesije se također izvršava u server.js datoteci, zatim se generira i renderira početna stranica, a to je login stranica za korisnika.

4.1. Login i registracija

Stranica za login i stranica za registraciju su vrlo jednostavne i traže od korisnika opće informacije o korisničkom računu i koriste jednu zajedničku rutu (eng. route), a to je korisnik.js datoteka. Korisnik.js datoteka sadrži logiku za verificiranje korisnika, postavljanje sesije i prikaza podataka na HTML stranici poput pogrešaka kod logina.

Korisnici.js datoteka sadrži dohvaćanje informacija iz proslijeđenim linkom i te ovisno o linku i metodi izvršiti će se određeni dio koda. Primjerice ako proslijedimo samo link bez metode pozvati će se router get i ovisno o drugom dijelu linka koji sadrži naziv radnje izvršiti će se kod gdje je ta j naziv radnje. Primjerice imamo link /korisnici/registracija/ prvi dio linka nas usmjerava na route korisnici dok drugi dio linka nas usmjerava na radnju registracija.

Isječak kôda 2: Primjer usmjeravanja korisnika

```
router.get('/registracija', (req, res) => {
    res.render('registration/registration', { korisnik: new Korisnik() })
})
router.get('/odjava', (req, res) => {
    req.session.username = null
    req.session.uloga = null
    res.render('login/login', { error: '' })
})
```

Navedeni dio koda služe za preusmjeravanje korisnika na stranicu registracija i drugi dio je odjava korisnika i brisanje sesije.

Zatim sljedeći dio koda prikazuje upravljanje podacima koje je korisnik unio na stranici login.

Isječak kôda 3: Primjer upravljanja unesenih podataka korisnika

```
router.post('/login', async (req, res) => {
    let korisničkoIme = req.body.username
    const korisnik = await Korisnik.findOne({ username: korisničkoIme })
```

```
    if (!korisnik) {
        res.render('login/login', { error: `Ne postoji username: ${korisničkoIme}`})
    } else {
        if (korisnik.password != req.body.lozinka) {
            res.render('login/login', { error: 'Netočna lozinka' })
        } else {
            const knjige = await Knjiga.find()
            req.session.username = korisničkoIme
            req.session.uloga = korisnik.uloga
            let pristup
            if (korisnik.uloga == 2) {
                pristup = true
            } else {
                pristup = false
            }
            res.render('homepage/homepage', { knjige: knjige, pristup: pristup })
        }
    }
})
```

Najprije dohvatimo podate sa servera preko asinkrone metode, pošto moramo pričekati da se podaci dohvate iz baze podataka. Kreiramo novu varijablu gdje će se pohraniti dohvaćeni korisnik, a korisnika dohvaćamo preko komande await i pridružimo joj pripadajući objekt modela i funkciju za traženje jedne osobe, gdje uključimo korisničko ime koje je korisnik unio. Ako se je pronašao korisnik u bazi podataka provjeravamo da li su identične lozinke, te ako se lozinke poklapaju kreiramo novu sesiju sa pripadajućom ulogom i korisničkim imenom i preusmjeravamo korisnika na početnu stranicu, gdje se prikazuju knjige. Ako se korisnik nije pronašao u bazi podataka onda ispisujemo poruku da je navedeni korisnik nepostojeći i isto tako ako se lozinke ne poklapaju ispisujemo poruku o netočnoj lozinki. Za pohranu i kreiranje novog računa koristim sljedeći dio koda:

Isječak kôda 4: Primjer kreiranja novog korisničkog računa

```
router.post('/', async (req, res, next) => {
    let korisnik = new Korisnik({
        ime: req.body.ime,
        prezime: req.body.prezime,
        username: req.body.username,
        email: req.body.email,
        password: req.body.password,
    })
    try {
        korisnik = await korisnik.save()
        const knjige = await Knjiga.find()
        req.session.username = req.body.username
        req.session.uloga = 1
        res.render('homepage/homepage', { knjige: knjige, pristup: false })
    } catch (e) {
        console.log(e)
        res.render('registration/registration', { korisnik: korisnik })
    }
})
```

Najprije se inicijalizira novi objekt modela korisnika, gdje dodjeljujem podatke koje je korisnik unio na stranici registraciji. Zatim pokušavam zapisati podatke na bazu pomoću metode try catch. Ako se zapis uspije izvršiti u bazu podataka onda se kreira nova sesija sa ulogom registriranog korisnika i korisničkim imenom i preusmjerava se korisnika na početnu stranicu, ako je try catch dohvatio error, to znači da postoji već korisnički račun sa tim korisničkim imenom i prikazujemo korisniku poruku da mora odabrati drugo korisničko ime. Podatke pohranjujemo sa asinkronom komandom await i pridružimo joj ispunjeni objekt korisnika i funkciju save.

4.2. Pocetna stranica

Sama početna stranica je vrlo jednostavna i prikazuje samo par osnovnih informacija o knjigama, no kada se klikne na neku od knjiga otvara nam se malo zahtjevnija stranica gdje se prikazuju sve moguće informacija o knjizi, autoru, komentarima i mogućim preporukama. Prikaz početne stranica se vrši pomoću sljedećeg koda:

Isječak kôda 5: Primjer prikaza pocetne stranice

```
router.get('/prikazi', async (req, res) => {
    const knjige = await Knjiga.find()
    let pristup
    if (req.session.uloga == 2) {
        pristup = true
    } else {
        pristup = false
    }
    res.render('homepage/homepage', { knjige: knjige, pristup: pristup })
})
```

Učitavamo sve pohranjene knjige iz baze i pohranjujemo ih u varijablu knjige, zatim provjeravamo ulogu logiranog korisnika i učitavamo početnu stranicu, gdje joj pridodajemo učitane knjige i boolean vrijednost o ulozi korisnika. Primjerice ako je ulogirani korisnik običan korisnik njemu se pridoda false i neće mu se prikazati izbornik za dodavanje novih knjiga ili ažuriranje postojećih knjiga i kategorija u bazu podataka. Detalji knjiga se generiraju prema sljedećem kodu:

Isječak kôda 6: Primjer generiranja detalja knjiga

```
router.get('/detalji/:id', async (req, res) => {
    const idKnjige = req.params.id
    const ocjene = await Ocjena.find({ idKnjige: idKnjige })
    let usernameKomentara = new Array(ocjene.length)
    for (let i = 0; i < ocjene.length; i++) {
        let korisnik = await Korisnik.findById(ocjene[i].idKorisnika)
        usernameKomentara[i] = korisnik.username
    }
    const napisao = await Napisao.find({ idKnjige: idKnjige })
    let imenaPisaca = []
    for (let i = 0; i < napisao.length; i++) {
        let pisac = await Pisac.findById(napisao[i].idPisca)
        imenaPisaca.push(pisac)
```

```
14      }
15      const posjeduje = await Posjeduje.find({ idKnjige: idKnjige })
16      let imenaKategorija = new Array(posjeduje.length)
17      for (let i = 0; i < posjeduje.length; i++) {
18          let kategorija = await Kategorija.findById(posjeduje[i].idKategorije)
19          imenaKategorija[i] = kategorija.naziv
20      }
21      let preporuke = []
22      for (let i = 0; i < posjeduje.length; i++) {
23          let preporukaPosjedovanja = await Posjeduje.find({ idKategorije: posjeduje[i
            ].idKategorije })
24          for (let j = 0; j < preporukaPosjedovanja.length; j++) {
25              if (preporukaPosjedovanja[j].idKnjige != idKnjige) {
26                  let podaciOcjenama = await Ocjena.find({ idKnjige:
                     preporukaPosjedovanja[j].idKnjige })
27                  let ukupnaOcjena = 0
28                  for (let z = 0; z < podaciOcjenama.length; z++) {
29                      ukupnaOcjena += podaciOcjenama[z].ocjena
30                  }
31
32                  let postojiVec = true
33                  for (let k = 0; k < preporuke.length; k++) {
34                      if (preporuke[k].id == preporukaPosjedovanja[j].idKnjige) {
35                          postojiVec = false
36                      }
37                  }
38                  if ((ukupnaOcjena / podaciOcjenama.length) >= 5) {
39                      if (postojiVec) {
40
41                          let knjiga = await Knjiga.findById(preporukaPosjedovanja[j].
                             idKnjige)
42                          preporuke.push(knjiga)
43                      }
44                  }
45              }
46          }
47      }
48
49      let pristup
50      if (req.session.uloga == 2) {
51          pristup = true
52      } else {
53          pristup = false
54      }
55
56      const knjiga = await Knjiga.findById(idKnjige)
57      res.render('homepage/detalji', {
58          knjiga: knjiga,
59          imenaKategorija: imenaKategorija,
60          pisci: imenaPisaca,
61          ocjene: ocjene,
62          username: usernameKomentara,
63          preporuka: preporuke,
```

```
64          pristup: pristup,
65      })
66  })
```

Najprije dohvatim proslijeđeni id knjige koji mi je potreban za pronalazak posjećene knjige, ocjene knjige i kategorija knjiga. Zatim dohvaćam ocjene i korisnike koju su dali ocjenu iz baze podataka i dodajem ih u novokreiranu listu za prikaz korisničkim imena. Drugi dio je dohvaćanje informacija o piscima knjige, koje pohranjujem u varijablu imenaPisaca. Zatim se pronalaze kategorije koje knjiga posjeduje i pohranjuje ih u varijablu imenaKategorija. Zatim slijedi važan dio ove aplikacije, a to je pronalazak preporuka za ostale knjige. Preporuka se pronalazi na temelju kategorija posjećene knjige i na temelju ocjena koje su ostali korisnici dali posječenoj knjizi. Znači kategorije posječene knjige koje smo ranije spomenuli, kreira se lista ostalih knjiga koje sadrže te iste kategorije, gdje se nadalje provjerava prosječna ocjena posječene knjige, te ako je veća od 5 (polovice bodova, pošto svaki korisnik može ocijeniti knjigu od 1 do 10) onda će se preporučiti knjiga ostalim korisnicima. Primjerice imamo knjigu sa pet recenzije i njegova ukupna ocjena je 31 od ukupnih 50 bodova, te prosječna ocjena iznosi 6,2, što je veće od 5 i navedena knjiga će se preporučiti korisnicima. Zatim posljednje što se mora provjeriti je uloga ulogiranog korisnika i dohvaćam informacije o samoj knjizi, gdje koristim funkciju findById i proslijedim joj id knjige. Tražena knjiga se pohranjuje u varijablu knjige i preusmjeravamo korisnika na stranicu za detalje knjige sa svim navedenim varijablama.

4.2.1. Algoritam preporuke korisnika

Nakon što korisnik ocijeni barem jednu ili više knjiga, otvara mu se nova mogućnost, a to je osobna preporuka ne ocijenjenih knjiga. Preporuka je jedinstvena za svakog korisnika, pošto svaki korisnik posjeduje vlastito mišljenje za svaku pojedinu knjigu i na temelju ocijenjenih ocjena koje je korisnik dao određenoj knjizi, generira se jedinstvena preporuka za pojedinog korisnika. Sami algoritam temelji se na YouTube videu za preporuke poznatih web mjesta kao što su Netflix i Amazon [4]. Algoritam se provodi kroz pet koraka, prvi korak je dohvaćanje popis svih ocjena koje je prijavljeni korisnik dao pročitanim knjigama, drugi korak je dohvaćanje cijelog popisa kategorija koje su dodijeljene, gdje se pomoću for petlje prolazi kroz sve kategorije i iz prvog koraka se dodjeljuju ocjene svakoj ocijenjenoj kategorije, gdje može doći do dupliciranih zapisa pojedine kategorije, no to će se kod idućeg koraka sumirati konačna ocjena u jednu kategoriju. Konačna ocjena svake kategorije se računa prema sljedećoj formuli: suma ocjene / 10 * suma ocjene, gdje je 10 (od 1 do 10) maksimalna ocjena koju korisnik može dati pojedinoj knjizi. Kod drugog koraka još dodatno zabilježim maksimalnu moguću ocjenu pojedine kategorije i njezin ID kategorije kako bi se kod idućih koraka mogla pronaći zadana kategorija. Treći korak traži ne ocijenjene knjige (pretpostavlja se da korisnik nije pročitao ne ocijenjenu knjigu) i sumira se njezina konačna postignuta ocjena na temelju iz drugog koraka, gdje se gledaju knjige koje posjeduju ocijenjene kategorije. Pomoću sume dobivamo faktor ocjene pojedine knjige, maksimalni faktor knjige i postotak knjige, koji predstavlja faktor preporuke knjige za prijavljenog korisnika. Treći korak još dodatno filtrira i izbacuje knjige koje sadrži faktor ocjene nula, zato što zadana knjige posjeduje kategorije koje korisnik nije ocijenio i ne zna se da li će se te kategorije sviđati korisnika ili ne. Četvrti korak sortira kreiranu listu iz

trećeg koraka po redu prema postotku preporuke knjige. Na samome vrhu liste preporučanih knjiga nalaze se knjige koje posjeduju najveći postotak preporuke knjige (u principu trebale bi se sviđati korisnika, pošto je korisnik pozitivno ocijenio te kategorije), dok na dnu liste nalaze se knjige koje korisnik bi trebao izbjegavati, pošto je korisnik te kategorije uglavnom negativno ocijenio. Posljednji peti korak je dohvaćanje naziva kategorija preporučanih knjiga iz baze podataka. Sljedeći dio koda prikazuje navedeni algoritam:

Isječak kôda 7: Primjer algoritma za osobne preporuke knjiga

```
router.get('/preporuke', async (req, res) => {
    if (req.session.username) {
        const logiraniKorisnik = req.session.username
        const korisnik = await Korisnik.findOne({ username: logiraniKorisnik })
        const popisOcjenaKorisnika = await Ocjena.find({ idKorisnika: korisnik.id })
        var listaOcjeneKategorija = []
        for (var i in popisOcjenaKorisnika) {
            var kategorijeKnjige = await Posjeduje.find({ idKnjige:
             popisOcjenaKorisnika[i].idKnjige })
            for (j in kategorijeKnjige) {
                var elementKnjige = {}
                elementKnjige.idKategorije = kategorijeKnjige[j].idKategorije
                elementKnjige.ocjena = popisOcjenaKorisnika[i].ocjena
                listaOcjeneKategorija.push(elementKnjige)
            }
        }
        const popisKategorija = await Kategorija.find()
        var popisKategorijeOcjena = []
        for (var i in popisKategorija) {
            idTrenutneKategorije = popisKategorija[i].id

            var ocjeneKaterogije = listaOcjeneKategorija.filter(i => i.idKategorije
             == idTrenutneKategorije).length
            var ukupnaOcjena = listaOcjeneKategorija.filter(i => i.idKategorije ==
             idTrenutneKategorije).reduce((a, b) => a + b.ocjena, 0)
            var konacnaOcjena = ((ukupnaOcjena / 10).toFixed(2) * ukupnaOcjena).
             toFixed(2)

            var noviElement = popisKategorija[i].toObject()
            noviElement.idKategorije = idTrenutneKategorije
            noviElement.konacnaOcjena = konacnaOcjena
            noviElement.maksimalnaOcjena = ocjeneKaterogije * 10 * ocjeneKaterogije
            popisKategorijeOcjena.push(noviElement)
        }
        var popisKnjigaPreporuke = []
        const popisSvihKnjiga = await Knjiga.find()
        for (var i in popisSvihKnjiga) {
            var idKnjige = popisSvihKnjiga[i].id
            var ocjenioKnjigu = popisOcjenaKorisnika.filter(i => i.idKnjige ==
             idKnjige)
            if (ocjenioKnjigu.length === 0) {
                var kategorijeKnjige = await Posjeduje.find({ idKnjige:
                 popisSvihKnjiga[i].id })
                var faktorOcjeneKnjige = 0
```

```
                var maksimalniFaktorOcjene = 0
                for (var j in kategorijeKnjige) {
                    var kategorijaKnjige = popisKategorijeOcjena.find(i => i.
                     idKategorije == kategorijeKnjige[j].idKategorije)
                    faktorOcjeneKnjige += parseFloat(kategorijaKnjige.konacnaOcjena)
                    maksimalniFaktorOcjene += parseFloat(kategorijaKnjige.
                     maksimalnaOcjena)
                }
                if (parseFloat(faktorOcjeneKnjige) > 0) {
                    var elementKnjige = popisSvihKnjiga[i].toObject()
                    elementKnjige.konacnaOcjena = faktorOcjeneKnjige
                    elementKnjige.maksimalnaOcjena = maksimalniFaktorOcjene
                    elementKnjige.postotak = (faktorOcjeneKnjige /
                     maksimalniFaktorOcjene * 100).toFixed(2)
                    popisKnjigaPreporuke.push(elementKnjige)
                }
            }

        }
        popisKnjigaPreporuke.sort(sortiranjeListe)
        for (var i in popisKnjigaPreporuke) {
            var popisKategorijaPreporuke = await Posjeduje.find({ idKnjige:
             popisKnjigaPreporuke[i]._id })
            var stringKategorije = ""
            for (var j in popisKategorijaPreporuke) {
                var podaciKateogirje = await Kategorija.findById(
                 popisKategorijaPreporuke[j].idKategorije)
                if (j == (popisKategorijaPreporuke.length - 1)) {
                    stringKategorije += podaciKateogirje.naziv
                } else {
                    stringKategorije += podaciKateogirje.naziv + ", "
                }
            }
            popisKnjigaPreporuke[i].nazivKategorije = stringKategorije
        }
        res.render('homepage/osobnePreporuke', { preporucaneKnjige: Array.from(
         popisKnjigaPreporuke) })
    } else {
        res.render('login/login', { error: '' })
    }
})
function sortiranjeListe(a, b) {
    if (a.postotak > b.postotak) {
        return -1
    }
    if (a.postotak < b.postotak) {
        return 1;
    }
    return 0;
}
```

4.3. Kategorije

Na stranicu kategorije može pristupiti samo admin i na toj stranici može dodavati nove kategorije knjiga, ažurirati postojeće kategorije ili brisati postojeće kategorije. Brisanje kategorije se vrši tako da se kreira nova forma gdje se prosljeđuje metode post i delete sa pripadajućem idjem kategorije. Funkcije za brisanje jednostavna funkcija gdje se izvrši komanda findByIdAndDelete i dodijeli joj se id kategorije. Funkcije za dodavanje i editiranje su vrlo slične. Kod dodavanja nove kategorije se inicijalizira novi prazni objekt kategorije i poziva se funkcija spremi kategoriju gdje se učitavaju uneseno ime kategorije i pohranjuje se na bazu. Dok funkcija za editiranje proslijedi joj se id kategorije i učitava se sa baze podaci o toj kategoriji.

Isječak kôda 8: Primjer prikaza upravljanja sa kategorijom

```
router.post('/', async (req, res, next) => {
    req.kategorija = new Kategorija()
    next()
}, spremiKategoriju())
router.put('/:id', async (req, res, next) => {
    req.kategorija = await Kategorija.findById(req.params.id)
    next()
}, spremiKategoriju())
function spremiKategoriju() {
    return async (req, res) => {
        let kategorija = req.kategorija
        kategorija.naziv = req.body.naziv
        try {
            kategorija = await kategorija.save()
            const kategorije = await Kategorija.find()
            res.render('kategorija/kategorija', { kategorije: kategorije })
        } catch (e) {
            res.render('kategorija/novaKategorija', { kategorija: kategorija,
            error: ' - kategorija već postoji' })
        }
    }
}
router.delete('/:id', async (req, res) => {
    await Kategorija.findByIdAndDelete(req.params.id)
    const kategorije = await Kategorija.find()
    res.render('kategorija/kategorija', { kategorije: kategorije })
})
```

Osim navedenih funkcija datoteka kategorija sadrži još funkcije za prikaz stranice kategorije (gdje se izlistaju sve postojeće kategorije), dodavanje nove kategorije i editiranje postojeće kategorije.

4.4. Knjige

Datoteka knjige može pristupiti samo admin stranice i datoteka sadrži funkcije za prikaz svih knjiga, prikaz stranice za dodavanje nove knjige, prikaz stranice za editiranje knjige i pri-

kaza stranice detalja o knjigama. Funkcija za brisanje knjiga sadrži još dodatne funkcije poput brisanje veze sa ocjenom, kategorijom i piscem.

Isječak kôda 9: Primjer brisanja knjiga

```
router.delete('/:id', async (req, res) => {
    const idKnjige = req.params.id
    await Knjiga.findByIdAndDelete(idKnjige)
    let posjeduje = await Posjeduje.find({ idKnjige: idKnjige })
    for (let i = 0; i < posjeduje.length; i++) {
        await Posjeduje.findByIdAndDelete(posjeduje[i].id)
    }
    let napisao = await Napisao.find({ idKnjige: idKnjige })
    for (let i = 0; i < napisao.length; i++) {
        await Pisac.findByIdAndDelete(napisao[i].idPisca)
        await Napisao.findByIdAndDelete(napisao[i].id)
    }
    let ocjena = await Ocjena.find({ idKnjige: idKnjige })
    for (let i = 0; i < ocjena.length; i++) {
        await Ocjena.findByIdAndDelete(ocjena[i].id)
    }
    const knjige = await Knjiga.find()
    res.render('knjiga/knjiga', { knjige: knjige })
})
```

Kod editiranja knjiga dodatno moramo još izbrisati sve stare veze sa kategorijama koje knjiga posjeduje, te sljedeći dio koda prikazuje funkcije za dodavanje nove knjige i editiranje postojeće knjige:

Isječak kôda 10: Primjer dodavanja i editiranja knjiga

```
router.put('/:id', async (req, res, next) => {
    req.knjiga = await Knjiga.findById(req.params.id)
    next()
}, spremiKnjigu('edit'))
function spremiKnjigu(path) {
    return async (req, res) => {
        let knjiga = req.knjiga
        const nazivKnjige = req.body.naziv
        knjiga.naziv = nazivKnjige
        const datum = (req.body.godinaIzdanja).split('T')
        knjiga.godinaIzdanja = datum[0]
        knjiga.jezik = req.body.jezik
        knjiga.stranica = req.body.stranica
        knjiga.publisher = req.body.publisher
        knjiga.dimenzije = req.body.dimenzije
        if (req.body.onlineIzdanje == 'on') {
            knjiga.onlineIzdanje = 'checked'
        } else {
            knjiga.onlineIzdanje = ''
        }
        knjiga.kratikiSadrzaj = req.body.kratikiSadrzaj
        let brojKnjiga = req.body.kategorija
        if (typeof brojKnjiga === 'string') {
```

```
            brojKnjiga = [req.body.kategorija]
        }
        try {
            knjiga = await knjiga.save()
            knjiga = await Knjiga.findOne({ naziv: nazivKnjige })
            let posjeduje = await Posjeduje.find({ idKnjige: knjiga.id })
            for (let i = 0; i < posjeduje.length; i++) {
                await Posjeduje.findByIdAndDelete(posjeduje[i].id)
            }
            for (let i = 0; i < brojKnjiga.length; i++) {
                posjeduje = new Posjeduje()
                posjeduje.idKnjige = knjiga.id
                posjeduje.idKategorije = brojKnjiga[i]
                posjeduje = await posjeduje.save()
            }
            const knjige = await Knjiga.find()
            res.render('knjiga/knjiga', { knjige: knjige })
        } catch (e) {
            const kategorije = await Kategorija.find()
            res.render(`knjiga/${path}`, { knjiga: knjiga, kategorije:
            kategorije, error: ' - knjiga već postoji' })
        }
    }
}
```

4.5. Pisci

Datoteka pisci sadrži logiku za prikazivanje stranice dodavanja pisaca za određenu knjigu, za samo dodavanje pisca za knjigu i brisanje pisca za knjigu. Kod dodavanja novog pisca za knjigu popunjavamo isto dvije tablice, a to su tablica pisac i tablica napisao, pošto možemo imati da je knjigu napisalo više pisaca i jedan pisac može napisati više knjiga. Za dodavanje novog pisca imamo sljedeći dio koda:

Isječak kôda 11: Primjer dodavanja novog pisca

```
router.post('/:id', async (req, res, next) => {
    req.knjiga = await Knjiga.findById(req.params.id)
    req.pisac = new Pisac()
    next()
}, spremiPisca())
function spremiPisca() {
    return async (req, res) => {
        let knjiga = req.knjiga
        let pisac = req.pisac
        pisac.ime = req.body.ime
        pisac.prezime = req.body.prezime
        const datum = (req.body.datumRodenja).split('T')
        pisac.datumRodenja = datum[0]
        pisac.mjestoRodenja = req.body.mjestoRodenja
        pisac.miniBio = req.body.miniBio
        try {
```

```
17              pisac = await pisac.save()
18              const idPisca = await Pisac.findOne({ime: req.body.ime, prezime: req.
                 body.prezime})
19              let napisao = new Napisao()
20              napisao.idKnjige = knjiga.id
21              napisao.idPisca = idPisca.id
22              napisao = await napisao.save()
23              const knjige = await Knjiga.find()
24              res.render('knjiga/knjiga', { knjige: knjige })
25          } catch (e) {
26              console.log(e)
27              res.render(`pisac/noviPisac`, { pisac: pisac, knjiga: knjiga, error: ' -
                  pisac već postoji' })
28          }
29      }
30  }
```

Kod za brisanje određenog pisca izgleda ovako:

Isječak kôda 12: Primjer brisanja pisca

```
1  router.delete('/:id', async (req, res) => {
2      await Pisac.findByIdAndDelete(req.params.id)
3      let napisao = await Napisao.find({ idPisca: req.params.id })
4      for (let i = 0; i < napisao.length; i++) {
5          await Napisao.findByIdAndDelete(napisao[i].id)
6      }
7      const knjige = await Knjiga.find()
8      res.render('knjiga/knjiga', { knjige: knjige })
9  })
```

4.6. Ocjene

Datoteka ocjene posjeduje samo dvije funkcije a to su prikaz stranice za dodavanje ocjene knjizi i samu logiku za upravljanje kod unosa ocjene u bazu podataka. Kada želimo pohraniti ocjenu u bazu potrebni su nam sljedeći podaci: sama ocjena, komentar, id knjige i id korisnika. Funkcija za pohranu ocjene izgleda:

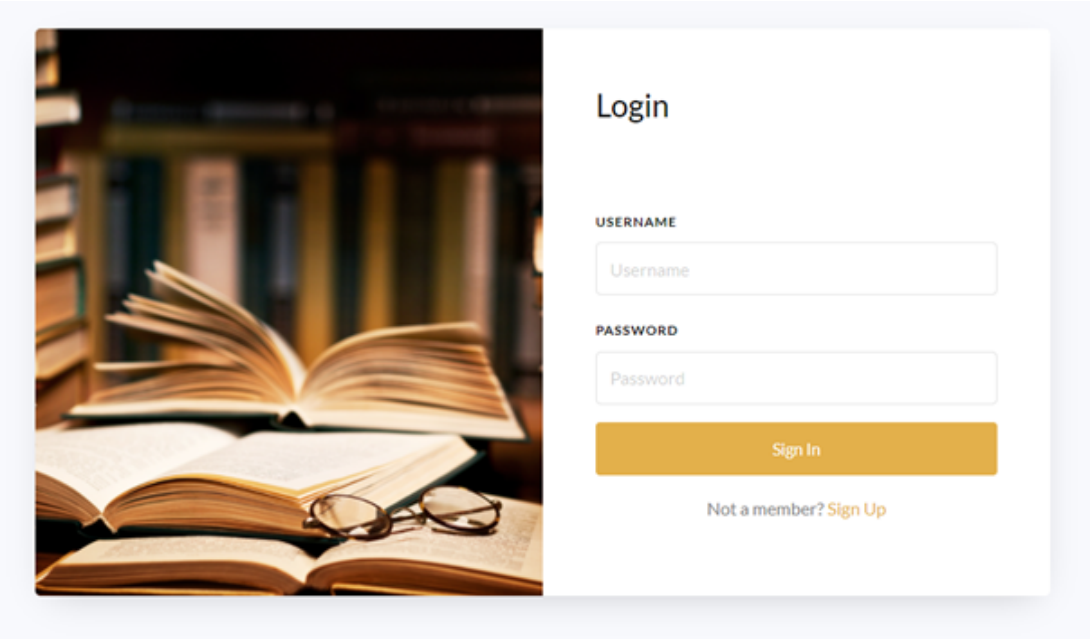
Isječak kôda 13: Primjer pohrane ocjene korisnika

```
1  router.post('/:id', async (req, res, next) => {
2      req.idKnjiga = req.params.id
3      next()
4  }, spremiOcjenu())
5  function spremiOcjenu() {
6      return async (req, res) => {
7          let pristup
8          if (req.session.uloga == 2) {
9              pristup = true
10         } else {
11             pristup = false
12         }
```

```
        const idKnjiga = req.idKnjiga
        const logiraniKorisnik = req.session.username
        const korisnik = await Korisnik.findOne({ username: logiraniKorisnik })
        let ocjena = await Ocjena.findOne({ idKnjige: idKnjiga, idKorisnika:
        korisnik.id })
        if (!ocjena) {
            ocjena = new Ocjena()
        }
        ocjena.ocjena = req.body.ocjena
        ocjena.komentar = req.body.komentar
        ocjena.idKnjige = idKnjiga
        ocjena.idKorisnika = korisnik.id
        try {
            ocjena = await ocjena.save()
            const knjige = await Knjiga.find()
            res.render('homepage/homepage', { knjige: knjige, pristup: pristup })
        } catch (e) {
            console.log(e)
            const knjiga = await Knjiga.findById(idKnjiga)
            res.render('ocjena/ocjena', { knjiga: knjiga, pristup: pristup })
        }
    }
}
```

5. Primjeri koristenja

Prva stranica koja nam se otvori je stranica za login korisnika, gdje se korisnik mora ili prijaviti ili napraviti novi račun kako bi se dalje mogao služiti aplikacijom.



Slika 3: Prikaz stranice login

Nakon što se korisnik uspješno prijavi ili kreira novi račun, otvori nam se naša početna stranica gdje se izlistaju sve knjige koje su pohranjene na bazi podataka. Početna stranica zahtjeva pojedine informacije od baze koje smo joj proslijedili preko routera prije samog učitavanja stranice. Početna stranica dobiva listu objekata i potrebno je prikazati svaki pojedini objekt kojeg je i dobila. Pomoću HTML koda i Express je to i moguće prikazati na stranici, primjerice kod ispistavanja svake pojedine knjige imamo sljedeći dio koda:

Isječak kôda 14: Primjer prikaza HTML koda knjiga

```
"<div class="container px-4 px-lg-5">
    <div class="row gx-4 gx-lg-5 justify-content-center">
        <div class="col-md-10 col-lg-8 col-xl-7">
        <% knjige.forEach(knjiga=> { %>
            <div class="post-preview">
            <a href="/homepage/detalji/<%= knjiga.id %>">
                <h2 class="post-title">
                    <%= knjiga.naziv %>
                </h2>
                <h3 class="post-subtitle">
                    <%= knjiga.kratikiSadrzaj %>
                </h3>
            </a>
            <p class="post-meta">
                Datum publiciranja <%= knjiga.godinaIzdanja %>
            </p>
```

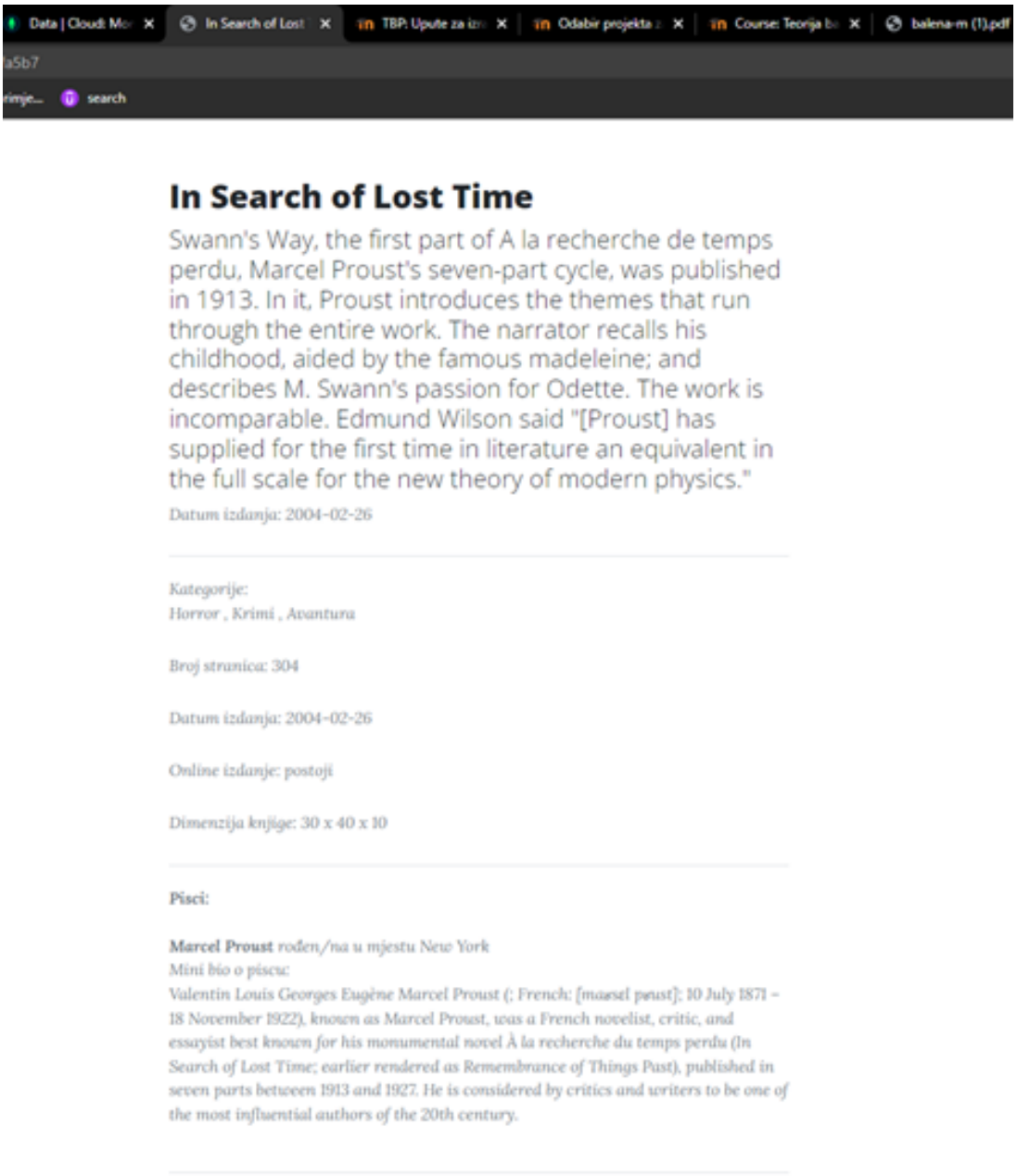
```
17              </div>
18              <!-- Divider-->
19              <hr class="my-4" />
20          <% }) %>
21          </div>
22      </div>
23  </div>"
```

In Search of Lost Time

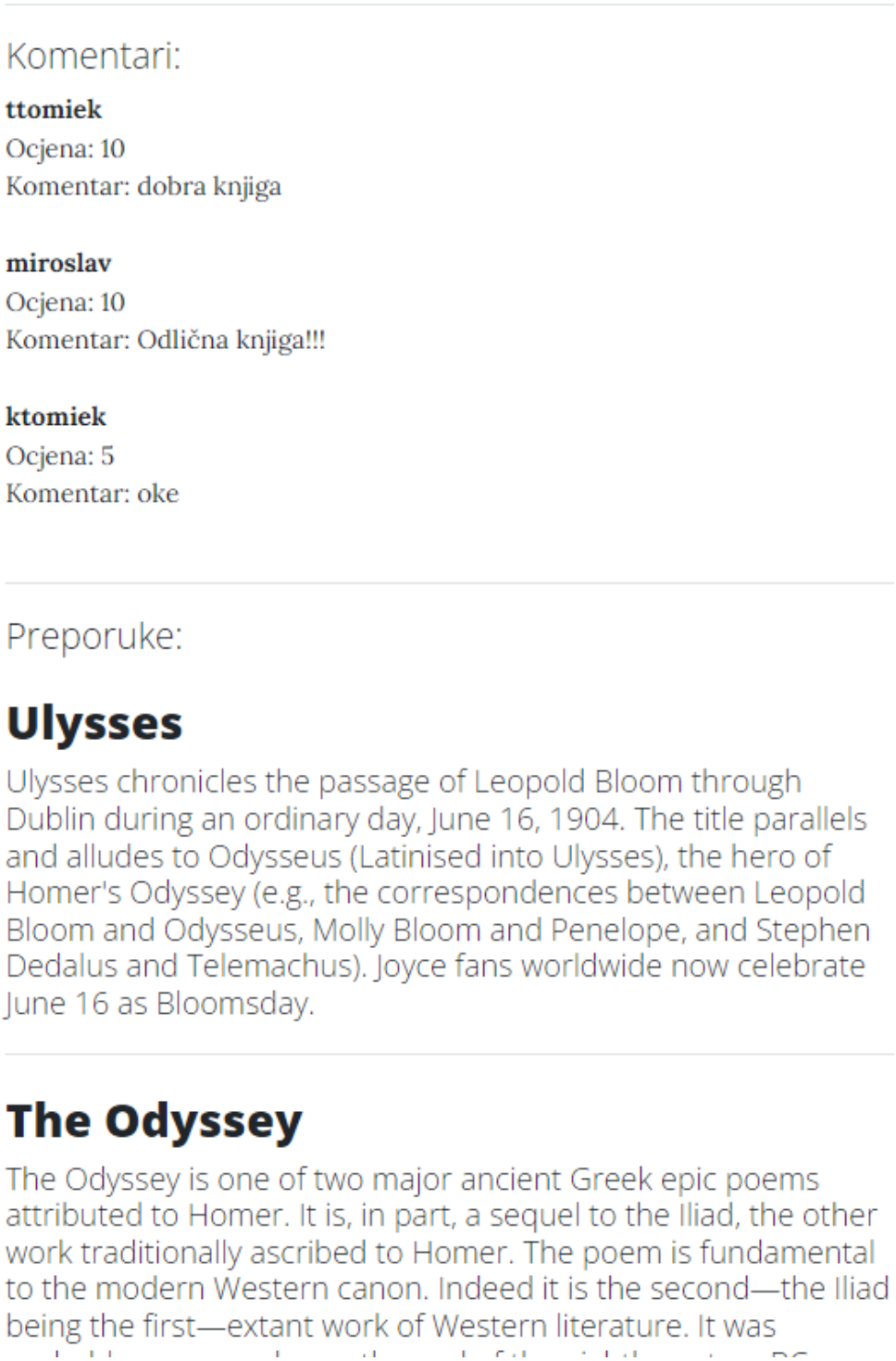
Swann's Way, the first part of A la recherche de temps perdu, Marcel Proust's seven-part cycle, was published in 1913. In it, Proust introduces the themes that run through the entire work. The narrator recalls his childhood, aided by the famous madeleine; and describes M. Swann's passion for Odette. The work is incomparable. Edmund Wilson said "[Proust] has supplied for the first time in literature an equivalent in the full scale for the new theory of modern physics."

Slika 4: Prikaz početne stranice

Kada korisnika pojedina knjiga zainteresira korisnik može jednostavno kliknuti na pojedinu knjigu i otvori mu se nova stranica, gdje je detaljno opisana knjiga. Slike pod rednim brojevima 4 i 5 sadrže prikaz stranice detalja o određenoj knjizi. Slika 5. prikazuje Dodane komentare i ocjene korisnika za otvorenu knjigu, a ispod komentara se nalaze preporuke ostalih knjiga koje se temelji na kategoriji otvorene knjige i ocjenama ostalih korisnika.

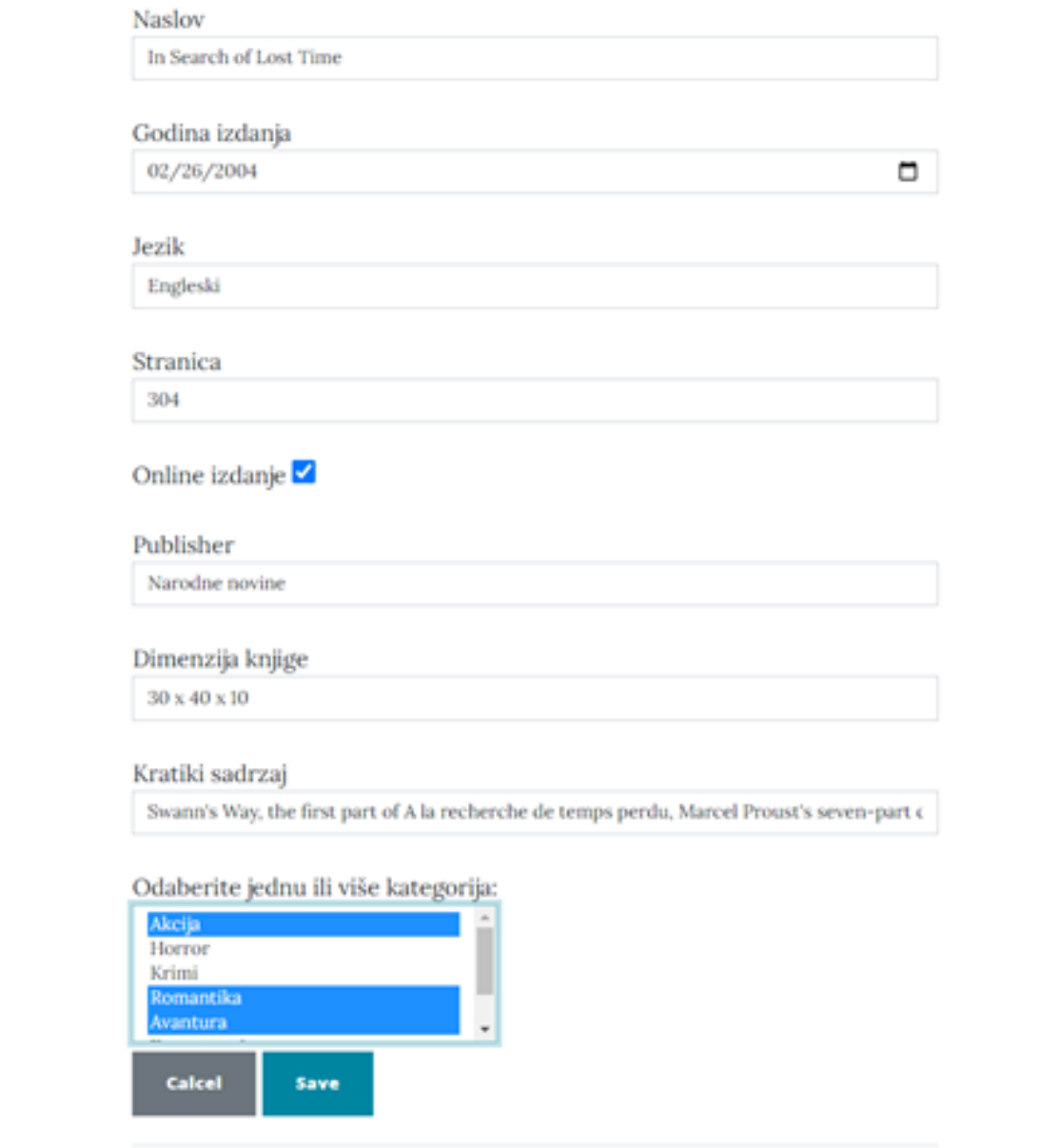


Slika 5: Prikaz stranice detalja o određenoj knjizi.



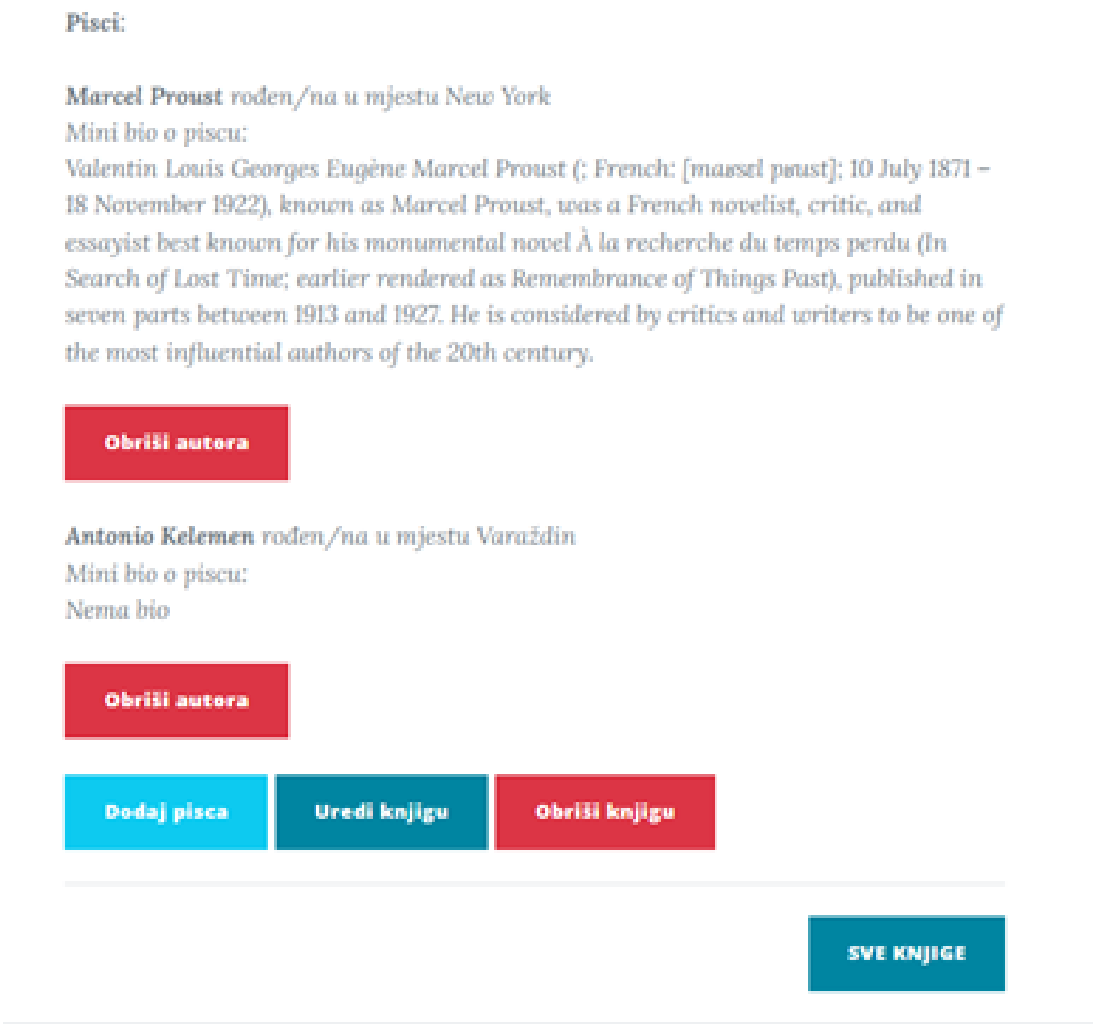
Slika 6: Prikaz stranice detalja o određenoj knjizi, prikaz komentara i preporuke.

Prikaz stranice uređivanja knjiga izgleda identično kao i početna stranica, no njoj može pristupiti samo prijavljeni korisnik sa ulogom admina. Stranica dodatno sadrži dugmiće za dodavanje nove knjige u sustav, brisanje postojeće knjige i editiranje postojećih knjiga. Kod dodavanja nove knjige ili editiranje postojeće knjige stranica izgledaju identično, no razlika je u tome što se kod editiranja ispuni obrazac za knjigu i dugmići posjeduju drugačije route linkove.



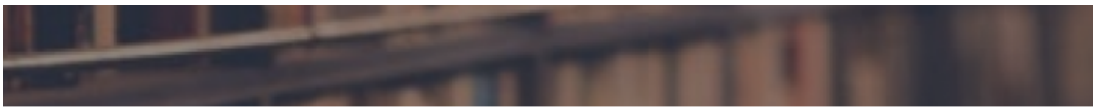
Slika 7: Prikaz obrasca za ispunu knjige.

Kada admin odabere pojedinu knjigu iz popisa otvori mu se stranica detalja o odabranoj knjizi gdje može ponovno uređivati knjigu, brisati knjigu ili dodati autora/autore knjizi. Stranica sadrži sve informacije o knjizi i naknadno nakon informacija o knjizi nalaze se svi autori knjige, gdje admin može dodavati novog autora ili brisati postojećeg autora.



Slika 8: Prikaz admin stranice detalja knjige.

Prikaz osobnih preporuka knjiga se otvara kada korisnik otvori link moje preporuke. Algoritam se provodi svaki puta kada korisnik dođe na navedenu stranicu, kako bi se preuzeli posljednje informacije koje je korisnik zabilježio i ocijenio prethodne knjige. Na stranici su na početku prikazane samo one knjige koje korisnik nije ocijenio (pretpostavlja se da one knjige koje nije ocijenio nije niti pročitao) i koje bi se korisniku mogle sviđati sa dodijeljenim postotkom preporučljivosti. Što je postotak preporučljivosti veći, u principu bi se te knjige trebale više sviđati korisniku, dok suprotno ako je postotak manji korisnik bi trebao izbjegavati navedenu knjigu. Kako korisnik pomiče stranicu na dolje postotak preporučljivosti pada i korisniku se prikazuju knjige koje mu se ne sviđaju, pošto je korisnik kategorije knjige ocijenio uglavnom negativno.



Knjige koje bi Vam preporučili na temelju Vaših ocjenjenih komentara ostalih knjiga:

Pustolovna avantura sa plot twistom

Postotak preporučljivosti: 90.25% (36.1/40)

Kategorija/e: Horror, Avantura

Ljubavno ubojstvo dr. Štefa

Postotak preporučljivosti: 79.40% (39.7/50)

Kategorija/e: Akcija, Horror, Romantika, Znanstveni

In Search of Lost Time

Postotak preporučljivosti: 73.25% (58.6/80)

Kategorija/e: Horror, Krimi, Avantura

Osveta detektiva duha

Postotak preporučljivosti: 73.25% (58.6/80)

Kategorija/e: Horror, Krimi, Avantura

Slika 9: Prikaz stranice osobnih preporuka.

6. Zaključak

Izrada same aplikacije je bila zanimljiva i poučna. Trebao sam možda iskoristiti više fleksibilnost polustrukturiranih bazi podataka poput MongoDB, a ne koristiti pomoćne tablice koje povezuju tablice sa vezama više na više, što mi je strašno otežalo editiranje i brisanje pojedinih tablica. Da sam imao više iskustva sa polustrukturiranim bazama ili iskustvo koje sam stekao nakon izrade projekta, izrada i implementacija takvih aplikacija bi bila duplo lakša. Ne bi se mučio i razmišljao kako da dohvatim sve potrebne id-jeve svih potrebnih tablica, nego bi iskoristio fleksibilnost takvih baza i sve pohranio u jednu tablicu sa raznovrsnim informacijama i podacima. Dohvaćanje i zapisivanje samih podataka je mnogo lakša za provesti, nego kod klasičnih SQL baza podataka, što je veliki plus kod polustrukturiranih bazama.

Popis literature

[1] M. Maleković i M. Schatten, *Teorija i primjena baza podataka*. Varaždin, HR: Redak, 2017.

[2] W. D. Simplified. „How To Build A Markdown Blog Using Node.js, Express, And MongoDB.” (14. 3. 2020.), adresa: https://www.youtube.com/watch?v=1NrHkjlWVhM&ab_channel=WebDevSimplified (pogledano 30. 6. 2022.).

[3] B. Dario. „Razvoj polustrukturirane baze podataka za potrebe internetskog foruma kao web-aplikacije.” (15. 9. 2020.), adresa: https://repozitorij.foi.unizg.hr/islandora/object/foi:6154 (pogledano 30. 6. 2022.).

[4] A. of the Problem. „How Recommender Systems Work (Netflix/Amazon).” (28. 9. 2020.), adresa: https://www.youtube.com/watch?v=n3RKsY2H-NE (pogledano 2. 7. 2022.).

Popis slika

Popis isječaka koda