



Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Inženjerstvo informacionih sistema
Veb orijentisane tehnologije i sistemi

Dokumentacija za MEAN 2 web aplikaciju

Sadržaj

1. Uvod	2
1.1 Opis aplikacije	2
1.2 Tehnologije	3
1.3 Način korišćenja	3
1.4 Osnovna struktura aplikacije	7
2. Backend	10
2.1 Modeli	14
2.2 Rute	16
2.2.1 Autentifikacija	17
3. Frontend	28
3.1 Modeli	34
3.2 Servisi	36
3.2.1 HTTP Zahtevi	37
3.3 Komponente	51
4. Ostalo	80
4.1 Linkovi	80
4.2 Sajt	82
4.3 Bazapodataka	87

1. Uvod

1.1 Opis aplikacije

U pitanju je web aplikacija za trgovanje knjigama.

Registrovani korisnici imaju mogućnost postavljanja knjiga za prodaju, kupovine knjiga od strane drugih korisnika, učestvovanje na forumu, ocenjivanja i komentarisanja kako knjiga, tako i iskustva sa određenim korisnicima u trgovini.

Posetnici koji nisu registrovani imaju mogućnost pregleda dostupnih knjiga, objava na forumu i objava od strane drugih korisnika.

Administrator ima sve mogućnosti kao i registrovani korisnik, s tim da može još da: briše i menja sve knjige, vidi listu svih registrovanih korisnika sajta, briše i menja sve objave na forumu.

1.2 Tehnologije

Za izradu aplikacije su korišćene sledeće tehnologije:

Angular, HTML; CSS i Bootstrap za Frontend i

NodeJS, ExpressJS, Moongose i MongoDB za Backend.

Za izradu aplikacije je korišćeno razvojno okruženje Visual Studio Code (<https://code.visualstudio.com/>).

Aplikacija je hostovana na besplatni servis Heroku (<https://www.heroku.com/>), dok se baza podataka nalazi na platformi Mlab (<https://mlab.com/>).

1.3 Način korišćenja

Za lokalno pokretanje aplikacije neophodno je imati instaliran NodeJS(<https://nodejs.org/en/>), Node Package Manager (NPM) i Mongodb bazu podataka (<https://www.mongodb.com/download-center?jmp=nav#community>).

Prvo je neophodno klonirati sledeći repozitorijum:
<https://github.com/Vukan-Markovic/Book-trading-club>.

Nakon toga je potrebno izvršiti sledeće komande (u command prompt-u, u folderu gde se nalazi projekat):

1. npm install - za instaliranje zavisnosti
2. npm run-script build - za pokretanje frontend dela
3. npm start - za pokretanje backend dela

Aplikacija se takođe nalazi na sajtu: <https://bookclubtrading.herokuapp.com/>.
Kompletan kod je dostupan na sledećem linku:
<https://github.com/Vukan-Markovic/Book-trading-club>.

1.4 Osnovna struktura aplikacije

Osnovna struktura aplikacije je kreirana upotrebom ExpressJS generatora (<https://expressjs.com/en/starter/generator.html>).

Za generisanje strukture aplikacije ExpressJS generatorom potrebno je izvršiti sledeće komande:

1. npm install -g express-generator - instaliranje express js framework-a
2. express mojaPrvaAplikacija - generisanje aplikacije sa Jade templating-om:
3. npm install - instaliranje zavisnosti.

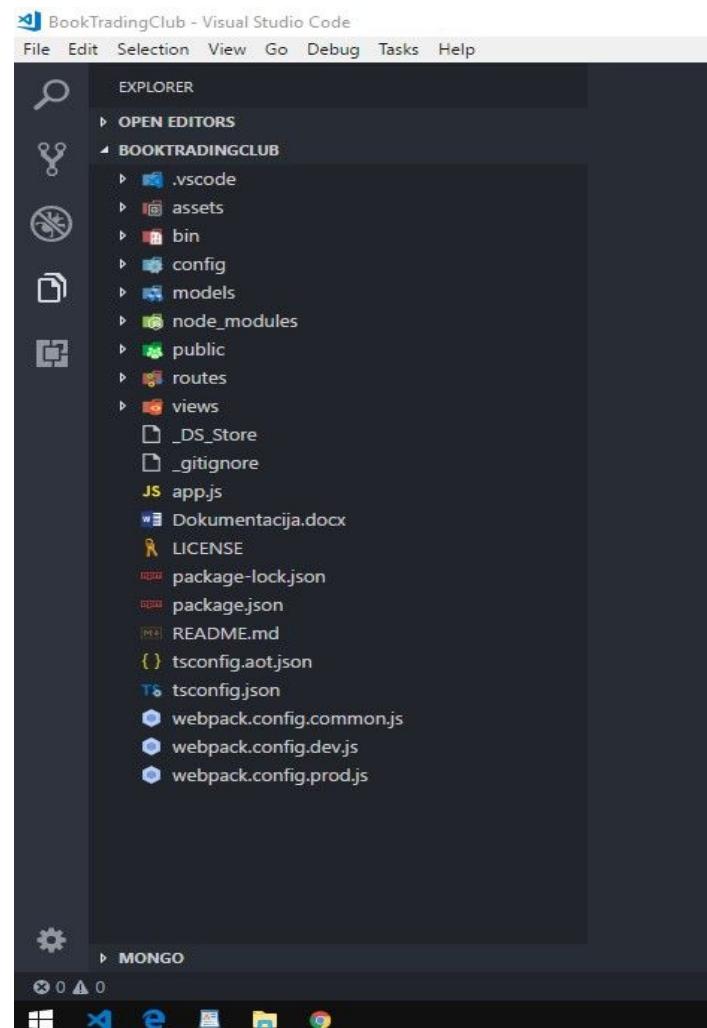
Glavne celine aplikacije:

-Backend:

- Osnovna Express i Serverska konfiguracija
- Modeli
- Rute i kontroleri vezani za njih – odgovaraju HTTP zahtevima
- Autentifikacija

-Frontend:

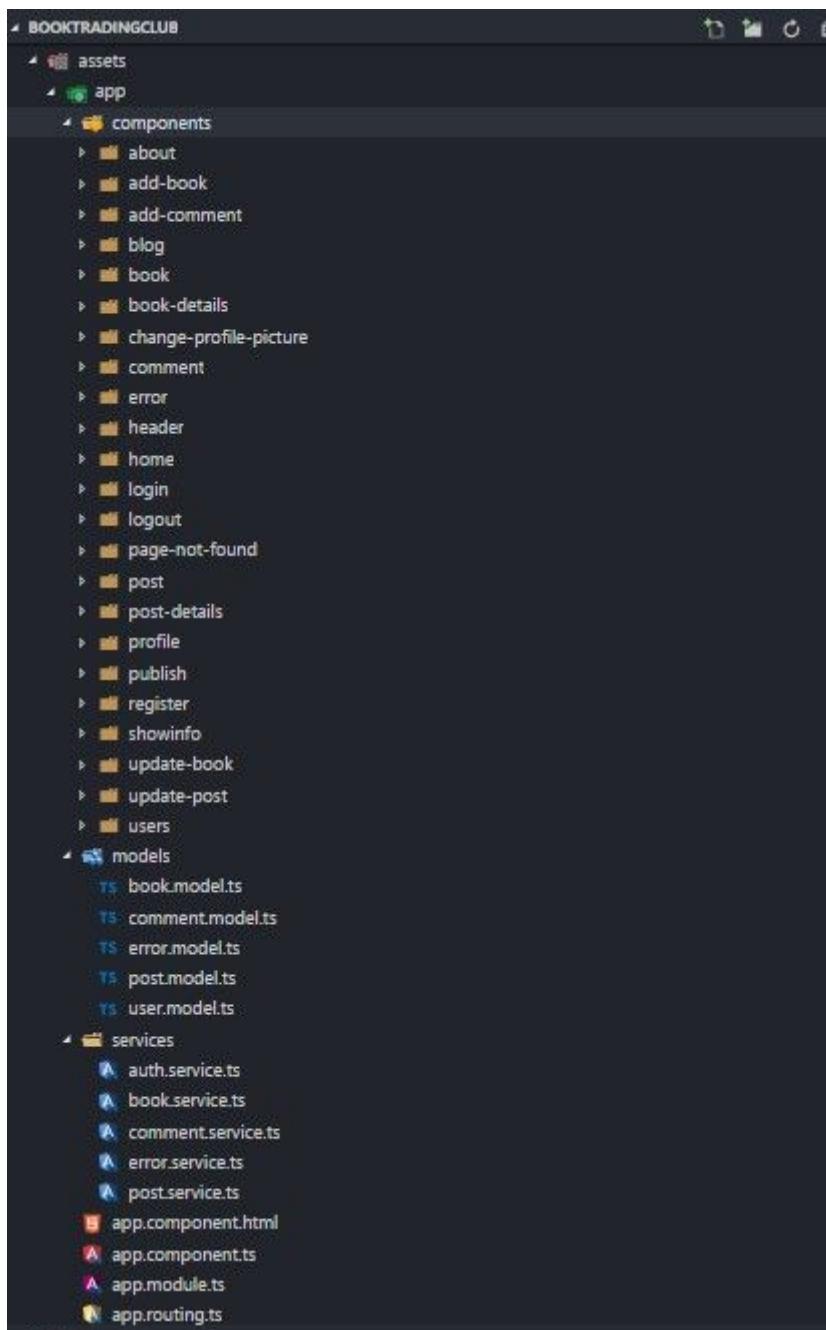
- Modeli
- Komponente
- Servisi
- Rute
- Stilizacija
- Autentifikacija



Slika 1. Glavna struktura aplikacije

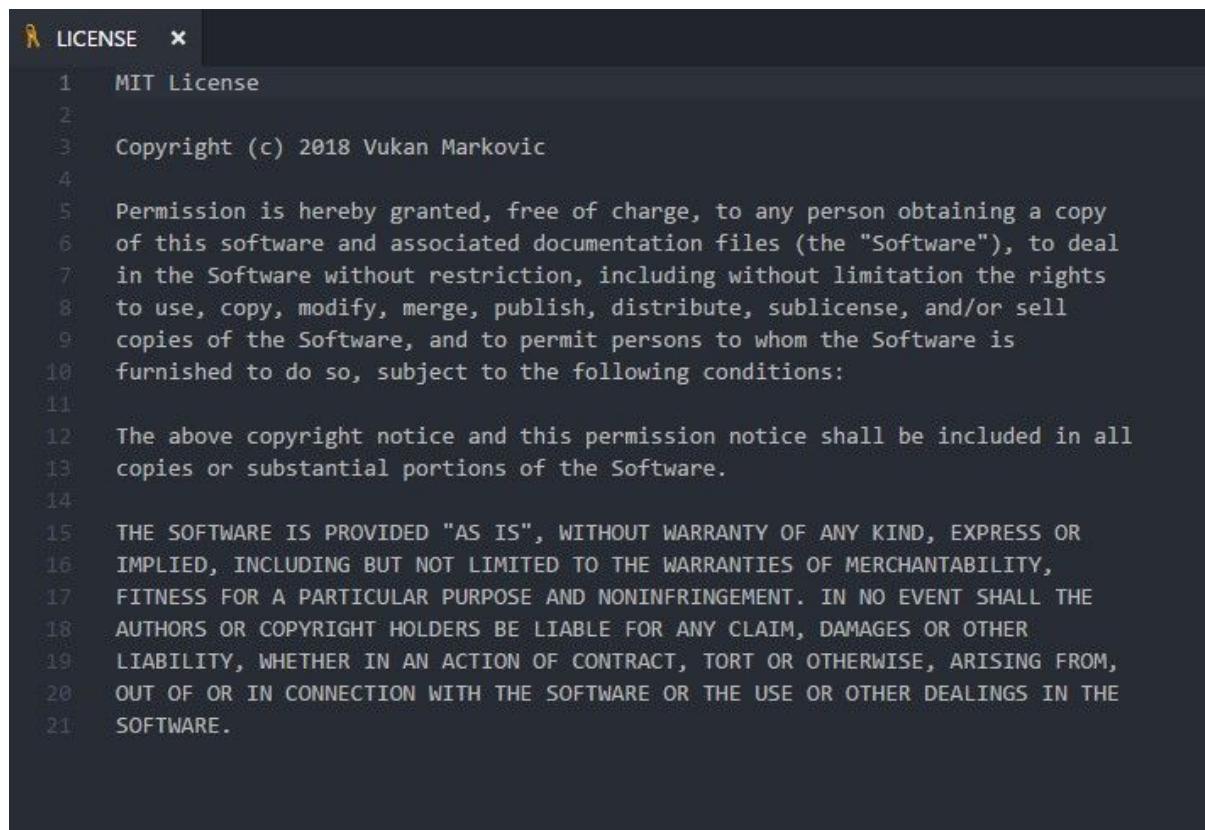


Slika 2. Backend struktura aplikacije



Slika 3. Frontend struktura aplikacije

Aplikacija je razvijana pod MIT licencom.



```
LICENSE ×
1 MIT License
2
3 Copyright (c) 2018 Vukan Markovic
4
5 Permission is hereby granted, free of charge, to any person obtaining a copy
6 of this software and associated documentation files (the "Software"), to deal
7 in the Software without restriction, including without limitation the rights
8 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9 copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in all
13 copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
19 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
20 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
21 SOFTWARE.
```

Slika 4. LICENSE fajl

package.json fajl je prisutan je u root direktorijumu u svakoj Node aplikaciji ili modulu i koristi se za definisanje svojstva paketa.

```
package.json ×
1  {
2    "name": "book-trading-club",
3    "version": "1.0.0",
4    "description": "Web MEAN application for books trading with authentication.",
5    "keywords": [
6      "book",
7      "online shop",
8      "blog",
9      "register",
10     "login",
11     "comment",
12     "grade",
13     "buy",
14     "profile"
15   ],
16   "author": "Vukan Marković",
17   "license": "MIT",
18   "private": true,
19   "main": "app.js",
20   "repository": {
21     "Book-trading-club": "https://github.com/Vukan-Markovic/Book-trading-club.git"
22   },
23   "homepage": "https://bookclubtrading.herokuapp.com/",
24   "engines": {
25     "node": "9.10.1"
26   },
27   "scripts": {
28     "start": "node ./bin/www",
29     "build": "del-cli public/js/app && webpack --config webpack.config.dev.js --progress --profile --watch",
30     "build:prod": "del-cli public/js/app && ngc -p tsconfig.aot.json && ngc -p tsconfig.aot.json && webpack --config webpack.config.pr
31 },
```

```
 30 package.json x
 31
 32   "dependencies": {
 33     "@angular/animations": "^4.0.0",
 34     "@angular/common": "^4.0.0",
 35     "@angular/compiler": "^4.0.0",
 36     "@angular/compiler-cli": "^4.0.0",
 37     "@angular/core": "^4.0.0",
 38     "@angular/forms": "^4.0.0",
 39     "@angular/http": "^4.0.0",
 40     "@angular/platform-browser": "^4.0.0",
 41     "@angular/platform-browser-dynamic": "^4.0.0",
 42     "@angular/platform-server": "^4.0.0",
 43     "@angular/router": "^4.0.0",
 44     "@angular/upgrade": "^4.0.0",
 45     "angular2-flash-messages": "^2.0.5",
 46     "bcryptjs": "^2.3.0",
 47     "body-parser": "~1.15.2",
 48     "bootstrap": "^3.3.7",
 49     "cookie-parser": "~1.4.3",
 50     "core-js": "^2.4.1",
 51     "debug": "~2.2.0",
 52     "email-validator": "^2.0.4",
 53     "express": "~4.14.0",
 54     "hbs": "~3.1.0",
 55     "jsonwebtoken": "^5.7.0",
 56     "mongoose": "^4.4.12",
 57     "mongoose-unique-validator": "^1.0.2",
 58     "morgan": "~1.6.1",
 59     "nodemailer": "^4.6.5",
 60     "npm": "^6.1.0",
 61     "password-validator": "^4.1.1",
 62     "reflect-metadata": "^0.1.3",
 63     "rxjs": "^5.2.0",
 64     "serve-favicon": "~2.3.0",
 65     "zone.js": "^0.8.5"
 66   },
 67   "devDependencies": {
 68     "@types/core-js": "0.9.36",
 69     "@types/node": "^6.0.45",
 70     "angular-router-loader": "^0.5.0",
 71     "angular2-template-loader": "^0.5.0",
 72     "awesome-typescript-loader": "^3.1.2",
 73     "del-cli": "^0.2.0",
 74     "html-loader": "^0.4.4",
 75     "raw-loader": "^0.5.1",
 76     "ts-loader": "^2.0.3",
 77     "typescript": "^2.1.4",
 78     "webpack": "^2.2.1",
 79     "webpack-merge": "^4.1.0"
 80   }
 81 }
```

Slika 6. Zavisnosti u package.json fajlu

2. Backend

U [www.js](#) fajlu se vrši kreiranje servera i kreiranje i podešavanje porta. Ovaj kod je automatski generisan od strane ExpressJS generatora.



```
JS www  ✘
1  #!/usr/bin/env node
2
3  /**
4   * Module dependencies.
5   */
6
7  var app = require('../app');
8  var debug = require('debug')('angular2-nodejs:server');
9  var http = require('http');
10
11 /**
12  * Get port from environment and store in Express.
13  */
14
15 var port = normalizePort(process.env.PORT || '3000');
16 app.set('port', port);
17
18 /**
19  * Create HTTP server.
20  */
21
22 var server = http.createServer(app);
23
24 /**
25  * Listen on provided port, on all network interfaces.
26  */
27
28 server.listen(port);
29 server.on('error', onError);
30 server.on('listening', onListening);
31
32 /**
33  * Normalize a port into a number, string, or false.
34  */
35
36 function normalizePort(val) {
37   var port = parseInt(val, 10);
38
39   if (isNaN(port)) {
40     // named pipe
41     return val;
42   }
43
44   if (port >= 0) {
45     // port number
46     return port;
47   }
48
49   return false;
50 }
51
52 /**
53  *
```

```
JS www  X

52 /**
53 * Event listener for HTTP server "error" event.
54 */
55
56 function onError(error) {
57   if (error.syscall !== 'listen') {
58     throw error;
59   }
60
61   var bind = typeof port === 'string'
62     ? 'Pipe ' + port
63     : 'Port ' + port;
64
65   // handle specific listen errors with friendly messages
66   switch (error.code) {
67     case 'EACCES':
68       console.error(bind + ' requires elevated privileges');
69       process.exit(1);
70       break;
71     case 'EADDRINUSE':
72       console.error(bind + ' is already in use');
73       process.exit(1);
74       break;
75     default:
76       throw error;
77   }
78 }
79
80 /**
81 * Event listener for HTTP server "listening" event.
82 */
83
84 function onListening() {
85   var addr = server.address();
86   var bind = typeof addr === 'string'
87     ? 'pipe ' + addr
88     : 'port ' + addr.port;
89   debug('Listening on ' + bind);
90   console.log('Server listening at port: ' + port);
91 }
```

Slika 8. Serverski [www.js](#) fajl

app.js predstavlja glavni serverski fajl aplikacije. U njemu se prvo se importuju svi potrebni moduli a zatim definišu sve rute koje će se koristiti u aplikaciji. Nakon toga se vrši konektovanje na bazu podataka. Ostala serverska podešavanja su automatski generisana od strane ExpressJS generatora.



The screenshot shows a code editor window with the file 'app.js' open. The code is written in JavaScript and follows a standard boilerplate structure for an Express application. It includes imports for various modules like express, path, favicon, morgan, cookieParser, bodyParser, and mongoose. It defines routes for different endpoints (appRoutes, bookRoutes, userRoutes, postRoutes, commentRoutes) and sets up the express application. It also connects to a MongoDB database using mongoose and sets up a view engine (HBS). Finally, it configures static resources and logging.

```
JS app.js ×
1 // importing modules
2 const express = require('express');
3 const path = require('path');
4 const favicon = require('serve-favicon');
5 const logger = require('morgan');
6 const cookieParser = require('cookie-parser');
7 const bodyParser = require('body-parser');
8 const mongoose = require('mongoose');
9
10 // importing routes
11 const appRoutes = require('./routes/app');
12 const bookRoutes = require('./routes/book');
13 const userRoutes = require('./routes/user');
14 const postRoutes = require('./routes/post');
15 const commentRoutes = require('./routes/comment');
16
17 // define express application
18 const app = express();
19
20 // importing database url
21 const database = require('./config/database.js');
22
23 // set up mongoose promise to global promise
24 mongoose.Promise = global.Promise;
25
26 //connect to database
27 mongoose.connect(database.url, { useNewUrlParser: true }, (error) => {
28   if (error) console.log('Error on connecting to database');
29   else console.log('Connected to database: ' + database.url);
30 });
31
32 // view engine setup
33 app.set('views', path.join(__dirname, 'views'));
34 app.set('view engine', 'hbs');
35
36 // set up static resources
37 app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
38 app.use(express.static(path.join(__dirname, 'public')));
39
40 // set up logging for development
41 app.use(logger('dev'));
42
```

Slika 9. Serverski app.js fajl

```

43 // set up body parser and cookie parser modules
44 app.use(bodyParser.json());
45 app.use(bodyParser.urlencoded({ extended: false }));
46 app.use(cookieParser());
47
48
49 // set up headers
50 app.use((req, res, next) => {
51   res.setHeader('Access-Control-Allow-Origin', '*');
52   res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
53   res.setHeader('Access-Control-Allow-Methods', 'POST, GET, PATCH, DELETE, OPTIONS');
54   next();
55 });
56
57 // define routes
58 app.use('/book', bookRoutes);
59 app.use('/user', userRoutes);
60 app.use('/post', postRoutes);
61 app.use('/comment', commentRoutes);
62 app.use('/', appRoutes);
63
64 // catch 404 and forward to error handler
65 app.use(function(req, res) {
66   return res.render('index');
67 });
68
69 // export app module
70 module.exports = app;

```

Slika 10. Serverski app.js fajl

U database.js fajlu se nalazi modul sa linkom na Mlab bazu podataka.

```

JS database.js ✘
1 // define and export module for mlab database url
2 module.exports = {
3   url: 'mongodb://Vukan:srbija1@ds159400.mlab.com:59400/booktradingclub'
4   // local database url: 'mongodb://localhost:27017/BookTradingClub'
5 };

```

Slika 11. database.js fajl

2.1 Modeli

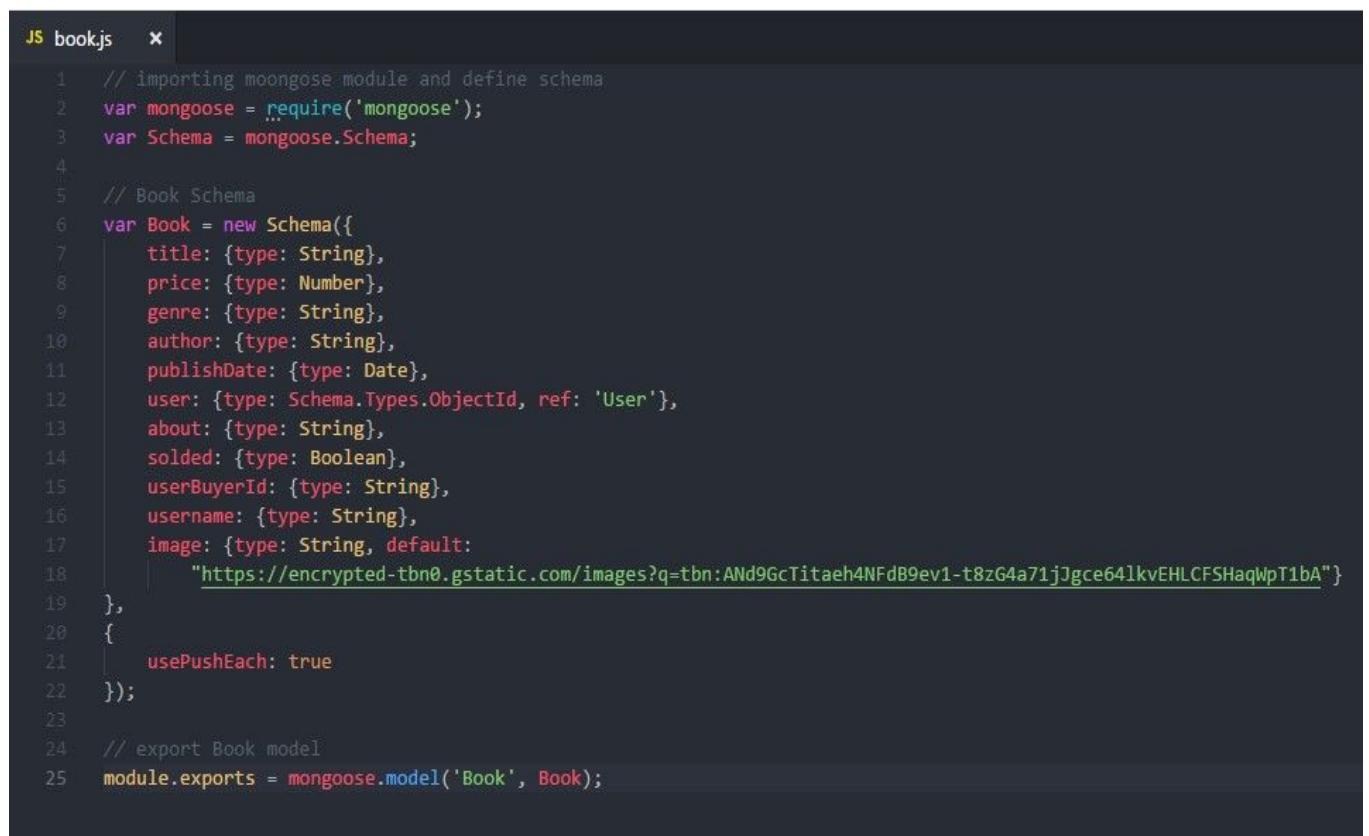
MongoDB rukuje jednostavnim JSON dokumentima. Poslovna logika je smeštena u aplikativni sloj. Mongoose ODM nam omogućuje jednostavnu konverziju između dokumenata i JavaScript objekata

Mongoose nam služi za struktuiranje podataka. Ne može se skladištiti ništa što ne odgovara šemi. Šema sluzi kao deklaracija kako treba da izgledaju podaci.

Modeli su prave klase kreirane u JavaScript-i.

U aplikaciji postoje četiri modela i to:

1. user.js - model koji reprezentuje korisnika
2. post.js - model koji reprezentuje objavu na forumu
3. comment.js - model koji reprezentuje komentar o knjizi
4. book.js - model koji reprezentuje knjigu



```
JS book.js  x
1 // importing mongoose module and define schema
2 var mongoose = require('mongoose');
3 var Schema = mongoose.Schema;
4
5 // Book Schema
6 var Book = new Schema({
7   title: {type: String},
8   price: {type: Number},
9   genre: {type: String},
10  author: {type: String},
11  publishDate: {type: Date},
12  user: {type: Schema.Types.ObjectId, ref: 'User'},
13  about: {type: String},
14  solded: {type: Boolean},
15  userBuyerId: {type: String},
16  username: {type: String},
17  image: {type: String, default:
18    "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTitaeh4NFdB9ev1-t8zG4a71jJgce64lkvEHLCSHaqWpT1bA"
19  },
20  {
21    usePushEach: true
22  });
23
24 // export Book model
25 module.exports = mongoose.model('Book', Book);
```

Slika 12. model (šema) knjige

```
JS comment.js ✘
1 // importing mongoose module and define schema
2 var mongoose = require('mongoose');
3 var Schema = mongoose.Schema;
4
5 // Comment Schema
6 var Comment = new Schema({
7   content: {type: String},
8   publishDate: {type: Date},
9   username: {type: String},
10  book: {type: Schema.Types.ObjectId, ref: 'Book'},
11  grade: {type: Number},
12  tag: {type: String}
13 },
14 {
15   usePushEach: true
16 });
17
18 // export Comment model
19 module.exports = mongoose.model('Comment', Comment);
```

Slika 13. model (šema) komentara o knjizi

```
JS post.js ✘
1 // importing mongoose module and define schema
2 const mongoose = require('mongoose');
3 const Schema = mongoose.Schema;
4
5 // Post Schema
6 var Post = new Schema({
7   title: {type: String, required: true},
8   content: {type: String, required: true},
9   user: {type: Schema.Types.ObjectId, ref: 'User'},
10  username: { type: String }
11 },
12 {
13   usePushEach: true
14 });
15
16 // export Post model
17 module.exports = mongoose.model('Post', Post);|
```

Slika 14. model (šema) objave na forumu

```
JS user.js
1 // importing mongoose module and define schema
2 var mongoose = require('mongoose');
3 var Schema = mongoose.Schema;
4 var mongooseUniqueValidator = require('mongoose-unique-validator');
5
6 // User Schema
7 var User = new Schema({
8     firstName: {type: String, required: true},
9     lastName: {type: String, required: true},
10    password: {type: String, required: true},
11    email: {type: String, required: true, unique: true},
12    contry: {type: String, required: true},
13    city: {type: String, required: true},
14    postalCode: {type: String, required: true},
15    address: {type: String, required: true},
16    phoneNumber: {type: String, required: true},
17    image: {type: String, default: "https://www.shareicon.net/data/2016/09/01/822711_user_512x512.png"}
18 }, {
19     usePushEach: true
20 });
21
22 // plugin mongooseUniqueValidator to User schema
23 User.plugin(mongooseUniqueValidator);
24
25 // export User model
26 module.exports = mongoose.model('User', User);
```

Slika 15. Model (šema) registrovanog korisnika sajta

2.2 Rute

Rutiranje se odnosi na to kako aplikacija odgovra na klijentske zahteve u zavisnosti od endpoint-a, HTTP zahteva (GET, POST,...), URI-a...

Vrši se parsiranje tela zahteva i preuzimanje parametara (parsiranje se vrši ukoliko je content-type application/JSON ili application/x-www-form-urlencoded).

Upiti ka bazi su zahtevni prema resursima. Zato se koriste kroz Callback-ove. Šalje se zahtev, obrađuju se drugi poslovi, i kada se dobije povratna vrednost upita, izvršava se callback funkcija. Upiti su uvek u formatu: function(error, result) {...}.

Express framework omogućuje integriranje middleware-a za odgovor na HTTP zahteve, definisanje ruting tabele radi implementacije raznih akcija u zavisnosti od URL-a i HTTP metoda kao i dinamičko renderovanje HTML strana bazirano na prosleđivanju argumenata templejtu.

Express aplikacije koriste callback funkcije čiji su parametri request i response objekti.

Request objekat predstavlja HTTP zahtev i poseduje propertije za query string, pararametre, body, HTTP hedere... Response objekat predstavlja HTTP odgovor koji Express aplikacija šalje kao odgovor na zahtev.

Pri tome body-parse middleware nam služi za rukovanje JSON, Text, URL enkodovanim i sirovim podacima.

2.2.1 Autentifikacija

Svi zahtevi sa klijenta stižu preko tokena. Korisiti se JWT - JSON web tokens

Svi zahtevi su autentifikovani. Prilikom logovanja token se šalje korisniku sa servera, pri čemu se čuva u LocalStorage na klijentu. U token-u se čuvaju enkriptovane informacije o korisniku. Vreme trajanja tokena je proizvoljno, u aplikaciji je 10800s.

Za svaki zahtev prema bazi, korisnik se verifikuje preko tokena. Ako je token verifikovan, iz njega se izvlače informacije o korisniku i akcije ka bazi vezane su samo za korisnikove podatke.

U aplikaciji postoje četiri rute i to:

1. user.js - ruta za sve zahteve vezane za korisnike (registracija, logovanje, promena profilne slike)
2. post.js - ruta za sve zahteve za objave (dodavanje novih objava, njihovo brisanje i ažuriranje od strane administratora)
3. comment.js - ruta koja reprezentuje sve zahteve vezane za komentare komentar o knjigama (dodavanje novog komentara)
4. book.js - ruta koja reprezentuje sve zahteve vezane za knjige (dodavanje knjige, brisanje, ažuriranje, pregled knjige, kao i brisanje i ažuriranje od strane administratora koji može da briše i ažurira sve knjige).

```
JS comment.js ×
1 // import express module
2 const express = require('express');
3
4 // define express router
5 const router = express.Router();
6
7 // import jsonwebtoken for authentication
8 const jwt = require('jsonwebtoken');
9
10 // importing models
11 const Book = require('../models/book');
12 const Comment = require('../models/comment');
13
14 // GET request for all comments for book
15 router.get('/:id', (req, res) => {
16
17     // find book with provided id in database
18     Book.findById(req.params.id, (err, book) => {
19         if (err) return res.status(500).json({
20             title: 'An error occurred on finding comment',
21             error: err
22         });
23
24         // find and return all comments for that book
25         Comment.find({book: book}, (err, comments) => {
26             if (err) return res.status(500).json({
27                 title: 'An error occurred on finding comment',
28                 error: err
29             });
30             res.status(200).json({message: 'Success on getting book comments', obj: comments});
31         });
32     });
33 });
34
35 // check if user is logged in for all future routes
36 router.use('/', (req, res, next) => {
37     jwt.verify(req.query.token, 'secret', (err) => {
38         if (err) return res.status(401).json({title: 'Not Authenticated', error: err});
39         next();
40     });
41 });
42
```

Slika 16. GET zahtev za sve komentare za određenu knjigu

```
JS comment.js ×
43 // POST request for adding new comment
44 router.post('/:id', (req, res) => {
45
46     // find user with provided id
47     Book.findById(req.params.id, (err, book) => {
48
49         // check for error
50         if (err) return res.status(500).json({title: 'An error occurred', error: err});
51         if (!book) return res.status(500).json({
52             title: 'No book found!',
53             error: {message: 'Book with provided is not found in database'}
54         });
55
56         // create new comment
57         const comment = new Comment({
58             content: req.body.content,
59             publishDate: Date.now(),
60             username: req.body.username,
61             book,
62             grade: req.body.grade,
63             tag: req.body.tag
64         });
65
66         // save new comment
67         comment.save((err, result) => {
68             if (err) return res.status(500).json({title: 'An error occurred', error: err});
69             res.status(201).json({message: 'Saved comment', obj: result});
70         });
71     });
72 });
73
74 // export post-router module
75 module.exports = router;
```

Slika 17. POST zahtev za dodavanje novog komentara o knjizi ili korisniku

```
JS app.js  x
1 // importing express module
2 const express = require('express');
3
4 // define express router
5 const router = express.Router();
6
7 // import book model
8 const Book = require('../models/book');
9
10 // GET request for home page, render index page
11 router.get('/', (req, res) => {
12   res.render('index', { title: 'Book trading club' });
13 });
14
15 // GET request for home page, get and return all books from database
16 router.get('/home', (req, res) => {
17   Book.find({}, ((err, books) => {
18     if (err) return res.status(500).json({title: 'An error occurred', error: err});
19     res.status(200).json({message: 'Success', obj: books});
20   }));
21 });
22
23 // export app router module
24 module.exports = router;
```

Slika 18. Osnovna ruta home stranice, prikazuju se sve knjige svih korisnika

```

JS user.js  x

1 // importing modules
2 const express = require('express');
3 const bcrypt = require('bcryptjs');
4 const jwt = require('jsonwebtoken');
5 const nodemailer = require('nodemailer');
6
7 // define express router
8 const router = express.Router();
9
10 // importing user model
11 const User = require('../models/user');
12
13 // GET request for all users, request can be sent only by admin
14 router.get('/', (req, res) => {
15
16     // find and return all users from database
17     User.find({}, ((err, users) => {
18         if (err) return res.status(500).json({title: 'An error occurred', error: err});
19         res.status(200).json({message: 'Success', obj: users});
20     }));
21 });
22
23 // GET request for one user
24 router.get('/:id', function (req, res) {
25
26     // find and return user with provided id in database
27     User.findById(req.params.id, (err, user) => {
28         if (err) return res.status(500).json({title: 'An error occurred', error: err});
29         res.status(200).json({message: 'Success', obj: user});
30     });
31 });
32
33 // POST request for registration
34 router.post('/register', (req, res) => {
35
36     // create new user object
37     var user = new User({
38         firstName: req.body.firstName,
39         lastName: req.body.lastName,
40         password: bcrypt.hashSync(req.body.password, 10),
41         email: req.body.email,
42         country: req.body.country,
43         city: req.body.city,
44         postalCode: req.body.postalCode,
45         address: req.body.address,
46         phoneNumber: req.body.phoneNumber
47     });
48 });

```

Slika 19. GET zahtevi za sve korisnike i za pojedinačnog korisnika, kao i POST zahtev za registraciju

```
JS user.js ×

49 // save user to database
50 user.save((err, result) => {
51   if (err) return res.status(500).json({title: 'An error occurred', error: err});
52   res.status(201).json({message: 'User created', obj: result});
53
54   // send user welcome email after registration
55   sendEmail(user);
56 });
57 });
58
59 // POST request for login
60 router.post('/login', (req, res) => {
61
62   // Find user to database with provided email and password, checking for errors
63   User.findOne({email: req.body.email}, (err, user) => {
64     if (err) return res.status(500).json({title: 'An error occurred', error: err});
65
66     // no user with provided email address
67     if (!user) return res.status(401).json({
68       title: 'Login failed',
69       error: {message: 'Invalid email and/or password!'}
70     });
71
72     // passwords does not matches
73     if (!bcrypt.compareSync(req.body.password, user.password)) return res.status(401).json({
74       title: 'Login failed',
75       error: {message: 'Invalid email and/or password!'}
76     });
77
78     // login is success, create token for that user
79     const token = jwt.sign({user: user}, 'secret', {expiresIn: 10800});
80     res.status(200).json({message: 'Successfully logged in', token: token, userId: user._id});
81   });
82 });
83
84 // PATCH request for changing user profile picture
```

Slika 20. POST zahtev za logovanje

```
JS user.js  X

184 // PATCH request for changing user profile picture
185 router.patch('/:id', (req, res) => {
186
187     // decode provided user token
188     const decoded = jwt.decode(req.query.token);
189
190     // find user with provided id in database
191     User.findById(req.params.id, (err, user) => {
192         // check for errors
193         if (err) return res.status(500).json({title: 'An error occurred', error: err});
194         if (!user) return res.status(500).json({
195             title: 'No User Found!',
196             error: {message: 'User not found'}
197         });
198         if (user.id !== decoded.user._id) return res.status(401).json({
199             title: 'Not Authenticated',
200             error: {message: 'Users do not match'}
201         });
202         // update user profile image
203         user.image = req.body.image;
204         // save updated user to database
205         user.save((err, result) => {
206             if (err) return res.status(500).json({title: 'An error occurred', error: err});
207             res.status(200).json({message: 'Updated user image', obj: result});
208         });
209     });
210 });
211
212 // send greeting email to user after registration
213 function sendEmail (user) {
214     var transporter = nodemailer.createTransport({
215         service: 'gmail',
216         auth: {
217             user: 'booktradingclub@gmail.com',
218             pass: 'book123!'
219         }
220     });
221     var mailOptions = {
222         from: 'booktradingclub@gmail.com',
223         to: user.email,
224         subject: 'Welcome to book trading club!',
225         html: '<h1>Greeting message</h1>
2  <html lang="en">
3  <head>
4      <base href="/">
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <meta name="keywords" content="Book, Trading, Online shop, Forum, Profile, Authentication">
9      <meta name="description" content="Web application for book trading club">
10     <meta name="author" content="Vukan Markovic">
11     <title>Book Trading Club</title>
12     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" integrity="sha384-WskhaSGFgI" data-bbox="114 384 904 400" data-label="Text">
13     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/dist/css/bootstrap.min.css" data-bbox="114 405 904 421" data-label="Text">
14     <link href="https://fonts.googleapis.com/css?family=EB+Garamond&subset=latin,latin-ext" rel="stylesheet" type="text/css">
15     <link rel="stylesheet" href="/stylesheets/style.css" data-bbox="114 426 904 442" data-label="Text">
16 </head>
17 <body>
18 <my-app><span class="loader"></span></my-app>
19 <script src="/js/app/bundle.js"></script>
20 </body>
21 </html>
```

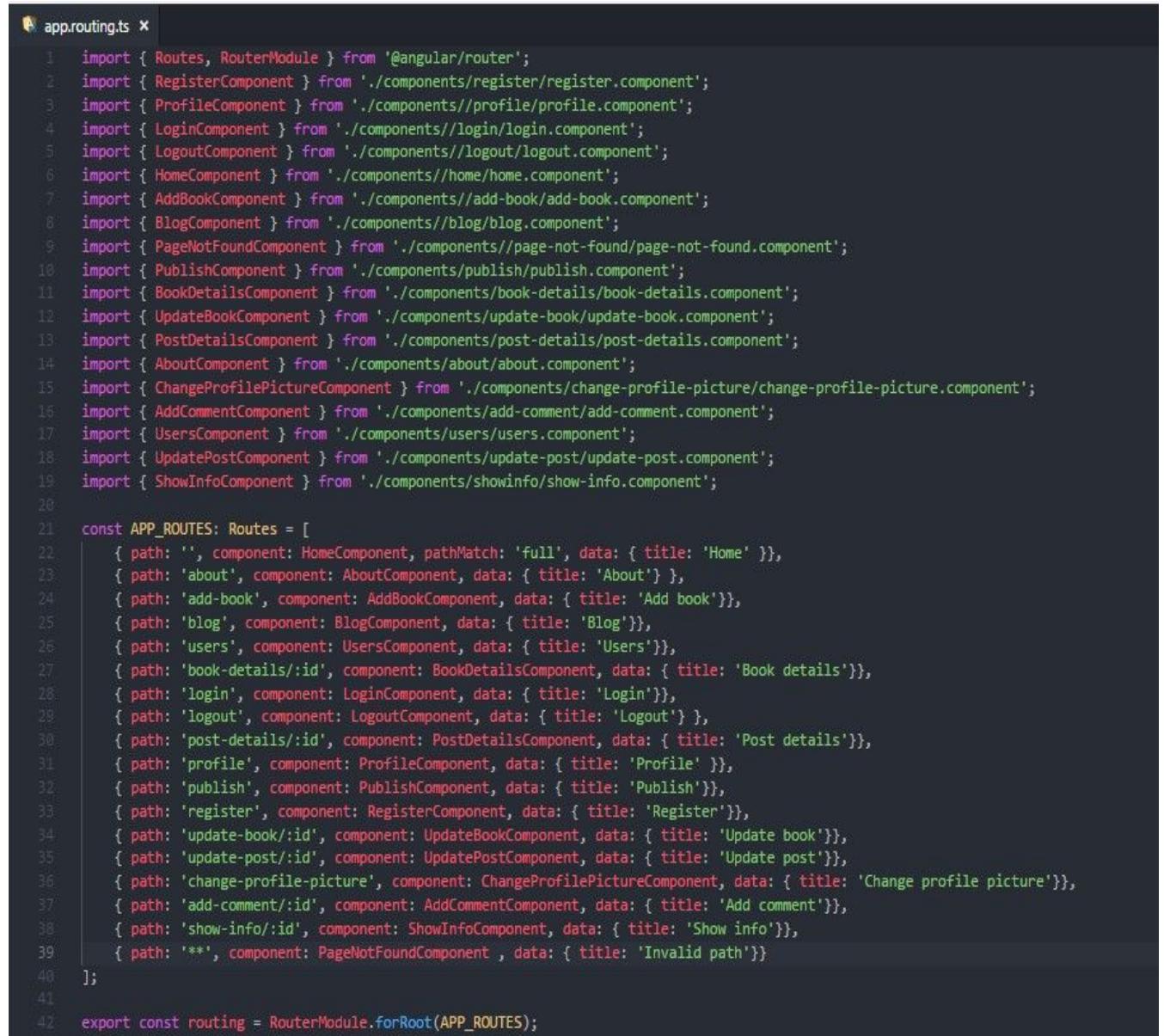
Slika 26. index.hbs html stranica

index.hbs stranica je kreirana putem Jade template-a i omogućava renderovanje templeita.

```
style.css x
1 body {
2     padding: 2em;
3     font-family: 'EB Garamond';
4     background-image: url(http://evonyunites.com/wp-content/uploads/2011/09/old-paper2.jpg);
5 }
6 a {
7     color: #00B7FF;
8     cursor: pointer;
9 }
10 a:hover, a:active {
11     color: #00B7FF;
12     text-decoration: underline;
13 }
14 .danger {
15     color: red;
16 }
17
18 .spacing {
19     margin-bottom: 20px;
20 }
21
22 input.ng-invalid.ng-touched {
23     border: 1px solid red;
24 }
25
26 .loader {
27     border: 16px solid green;
28     border-top: 16px solid greenyellow;
29     border-radius: 50%;
30     width: 40px;
31     height: 40px;
32     animation: spin 2s linear infinite;
33     position: absolute;
34     top: 50%;
35     left: 50%;
36     transform: translate(-50%, -50%);
37 }
38 @keyframes spin {
39     0% {
40         transform: rotate(0deg);
41     }
42     100% {
43         transform: rotate(360deg);
44     }
45 }
46
47 .content {
48     position: absolute;
49     top: 50%;
50     left: 50%;
51     transform: translateX(-50%) translateY(-50%);
52 }
```

Slika 27. style.css fajl

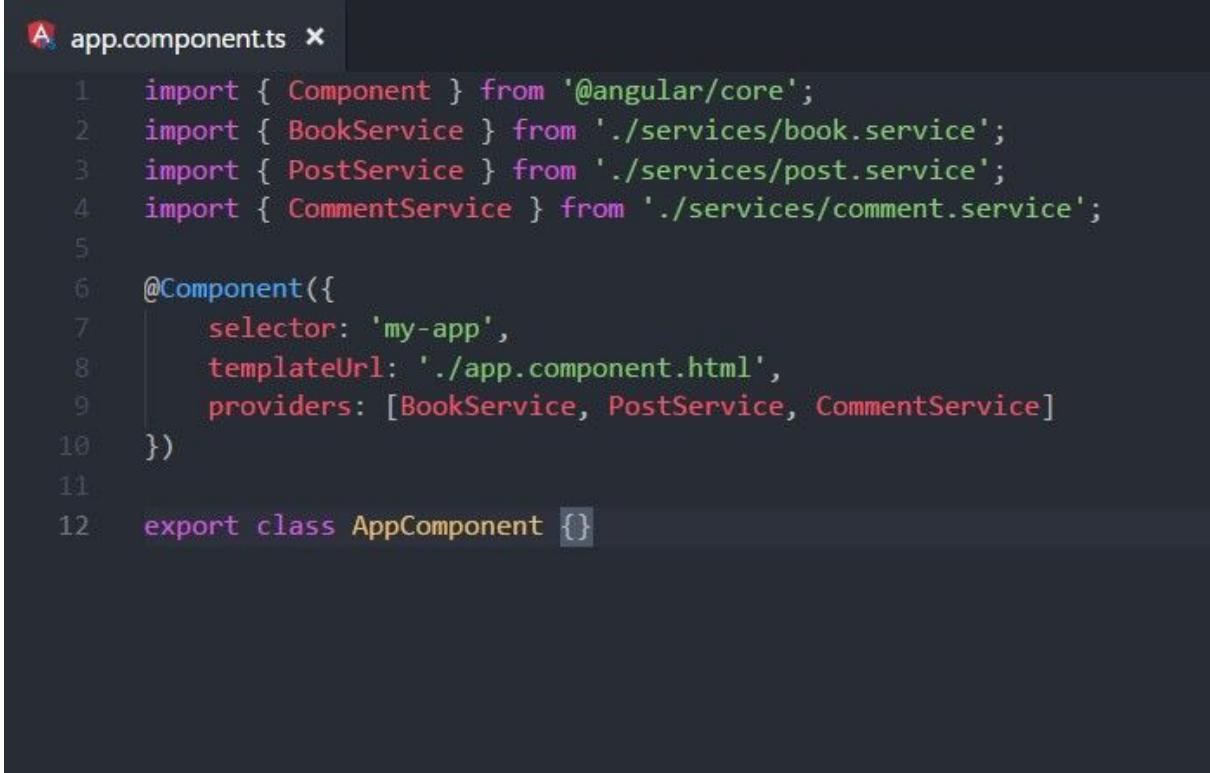
style.css fajl predstavlja glavni stysheet aplikacije, primenjuje se na sve komponente



```
app.routing.ts
1 import { Routes, RouterModule } from '@angular/router';
2 import { RegisterComponent } from './components/register/register.component';
3 import { ProfileComponent } from './components//profile/profile.component';
4 import { LoginComponent } from './components//login/login.component';
5 import { LogoutComponent } from './components//logout/logout.component';
6 import { HomeComponent } from './components//home/home.component';
7 import { AddBookComponent } from './components//add-book/add-book.component';
8 import { BlogComponent } from './components//blog/blog.component';
9 import { PageNotFoundComponent } from './components//page-not-found/page-not-found.component';
10 import { PublishComponent } from './components/publish/publish.component';
11 import { BookDetailsComponent } from './components/book-details/book-details.component';
12 import { UpdateBookComponent } from './components/update-book/update-book.component';
13 import { PostDetailsComponent } from './components/post-details/post-details.component';
14 import { AboutComponent } from './components/about/about.component';
15 import { ChangeProfilePictureComponent } from './components/change-profile-picture/change-profile-picture.component';
16 import { AddCommentComponent } from './components/add-comment/add-comment.component';
17 import { UsersComponent } from './components/users/users.component';
18 import { UpdatePostComponent } from './components/update-post/update-post.component';
19 import { ShowInfoComponent } from './components/showinfo/show-info.component';
20
21 const APP_ROUTES: Routes = [
22   { path: '', component: HomeComponent, pathMatch: 'full', data: { title: 'Home' } },
23   { path: 'about', component: AboutComponent, data: { title: 'About'} },
24   { path: 'add-book', component: AddBookComponent, data: { title: 'Add book'}},
25   { path: 'blog', component: BlogComponent, data: { title: 'Blog'}},
26   { path: 'users', component: UsersComponent, data: { title: 'Users'}},
27   { path: 'book-details/:id', component: BookDetailsComponent, data: { title: 'Book details'}},
28   { path: 'login', component: LoginComponent, data: { title: 'Login'}},
29   { path: 'logout', component: LogoutComponent, data: { title: 'Logout'} },
30   { path: 'post-details/:id', component: PostDetailsComponent, data: { title: 'Post details'}},
31   { path: 'profile', component: ProfileComponent, data: { title: 'Profile' } },
32   { path: 'publish', component: PublishComponent, data: { title: 'Publish'}},
33   { path: 'register', component: RegisterComponent, data: { title: 'Register'}},
34   { path: 'update-book/:id', component: UpdateBookComponent, data: { title: 'Update book'}},
35   { path: 'update-post/:id', component: UpdatePostComponent, data: { title: 'Update post'}},
36   { path: 'change-profile-picture', component: ChangeProfilePictureComponent, data: { title: 'Change profile picture'}},
37   { path: 'add-comment/:id', component: AddCommentComponent, data: { title: 'Add comment'}},
38   { path: 'show-info/:id', component: ShowInfoComponent, data: { title: 'Show info'}},
39   { path: '**', component: PageNotFoundComponent , data: { title: 'Invalid path'}}
40 ];
41
42 export const routing = RouterModule.forRoot(APP_ROUTES);
```

Slika 28. app.routing.ts fajl

U app.routing.ts fajlu je kreiran Angular ruter i definisane su sve moguće putanje (rute) sa odgovarajućim komponentama koje je potrebno učitati.

A screenshot of a code editor window titled "app.component.ts". The code is written in TypeScript and defines an Angular component. It imports four services: BookService, PostService, and CommentService from "./services", and @angular/core. The component is annotated with @Component, specifying a selector of 'my-app', a template URL of './app.component.html', and providers of [BookService, PostService, CommentService]. The class AppComponent is then exported.

```
1 import { Component } from '@angular/core';
2 import { BookService } from './services/book.service';
3 import { PostService } from './services/post.service';
4 import { CommentService } from './services/comment.service';
5
6 @Component({
7   selector: 'my-app',
8   templateUrl: './app.component.html',
9   providers: [BookService, PostService, CommentService]
10 })
11
12 export class AppComponent {}
```

Slika 29. app.component.ts fajl

app.component.ts fajl predstavlja glavni typescript fajl. U njemu definisemo servise BookService, PostService i CommentService kako bi bili dostupni svim ostalim komponentama.

```

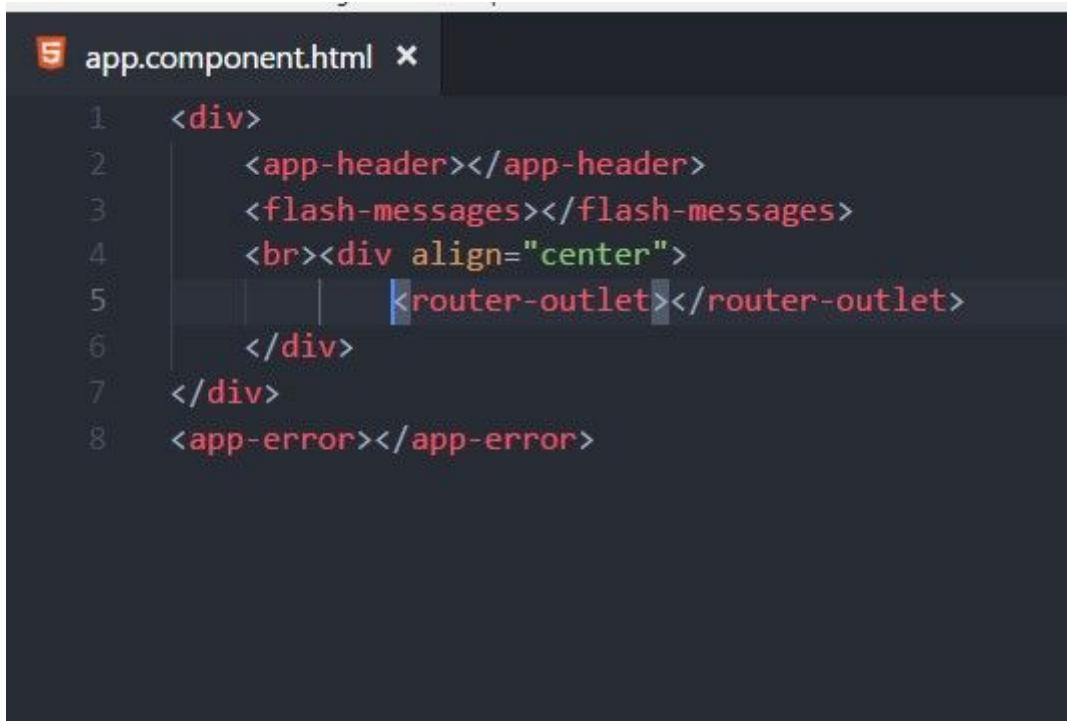

A app.module.ts x
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4 import { HttpClientModule } from '@angular/http';
5 import { FlashMessagesModule } from 'angular2-flash-messages';
6 import { AppComponent } from './app.component';
7 import { HeaderComponent } from './components/header/header.component';
8 import { LoginComponent } from './components/login/login.component';
9 import { RegisterComponent } from './components/register/register.component';
10 import { ProfileComponent } from './components/profile/profile.component';
11 import { AboutComponent } from './components/about/about.component';
12 import { PageNotFoundComponent } from './components/page-not-found/page-not-found.component';
13 import { AddBookComponent } from './components/add-book/add-book.component';
14 import { HomeComponent } from './components/home/home.component';
15 import { ErrorComponent } from './components/error/error.component';
16 import { LogoutComponent } from './components/logout/logout.component';
17 import { BlogComponent } from './components/blog/blog.component';
18 import { PostComponent } from './components/post/post.component';
19 import { PostDetailsComponent } from './components/post-details/post-details.component';
20 import { PublishComponent } from './components/publish/publish.component';
21 import { BookDetailsComponent } from './components/book-details/book-details.component';
22 import { UpdateBookComponent } from './components/update-book/update-book.component';
23 import { BookComponent } from './components/book/book.component';
24 import { ChangeProfilePictureComponent } from './components/change-profile-picture/change-profile-picture.component';
25 import { AddCommentComponent } from './components/add-comment/add-comment.component';
26 import { CommentComponent } from './components/comment/comment.component';
27 import { UsersComponent } from './components/users/users.component';
28 import { UpdatePostComponent } from './components/update-post/update-post.component';
29 import { ShowInfoComponent } from './components/showInfo/showInfo.component';
30 import { AuthService } from './services/auth.service';
31 import { ErrorService } from './services/error.service';
32 import { routing } from './app.routing';

33 @NgModule({
34   declarations: [
35     AppComponent,
36     AboutComponent,
37     AddBookComponent,
38     BlogComponent,
39     BookComponent,
40     BookDetailsComponent,
41     ErrorComponent,
42     HeaderComponent,
43     HomeComponent,
44     LoginComponent,
45     LogoutComponent,
46     PageNotFoundComponent,
47     PostComponent,
48     PostDetailsComponent,
49     ProfileComponent,
50     PublishComponent,
51     RegisterComponent,
52     UpdateBookComponent,
53     ChangeProfilePictureComponent,
54     CommentComponent,
55     AddCommentComponent,
56     UsersComponent,
57     UpdatePostComponent,
58     ShowInfoComponent
59   ],
60   imports: [
61     BrowserModule,
62     FormsModule,
63     ReactiveFormsModule,
64     HttpClientModule,
65     routing,
66     FlashMessagesModule.forRoot()
67   ],
68   providers: [AuthService, ErrorService],
69   bootstrap: [AppComponent]
70 })
71
72 export class AppModule {}


```

Slika 30. app.module.ts fajl

U app.module.ts fajlu definisemo sve komponente aplikacije, module i service AuthService i ErrorService koji će na taj način biti dostupni na nivou cele aplikacije. Definise se root komponenta unutar AppModule unutar bootstrap niza. Ne ucitavaju se sve komponente prilikom pokretanja Angular 2 aplikacije kako se ne bi narušile performanse.



```
5 app.component.html x
1 <div>
2   <app-header></app-header>
3   <flash-messages></flash-messages>
4   <br><div align="center">
5     |   <router-outlet></router-outlet>
6   </div>
7 </div>
8 <app-error></app-error>
```

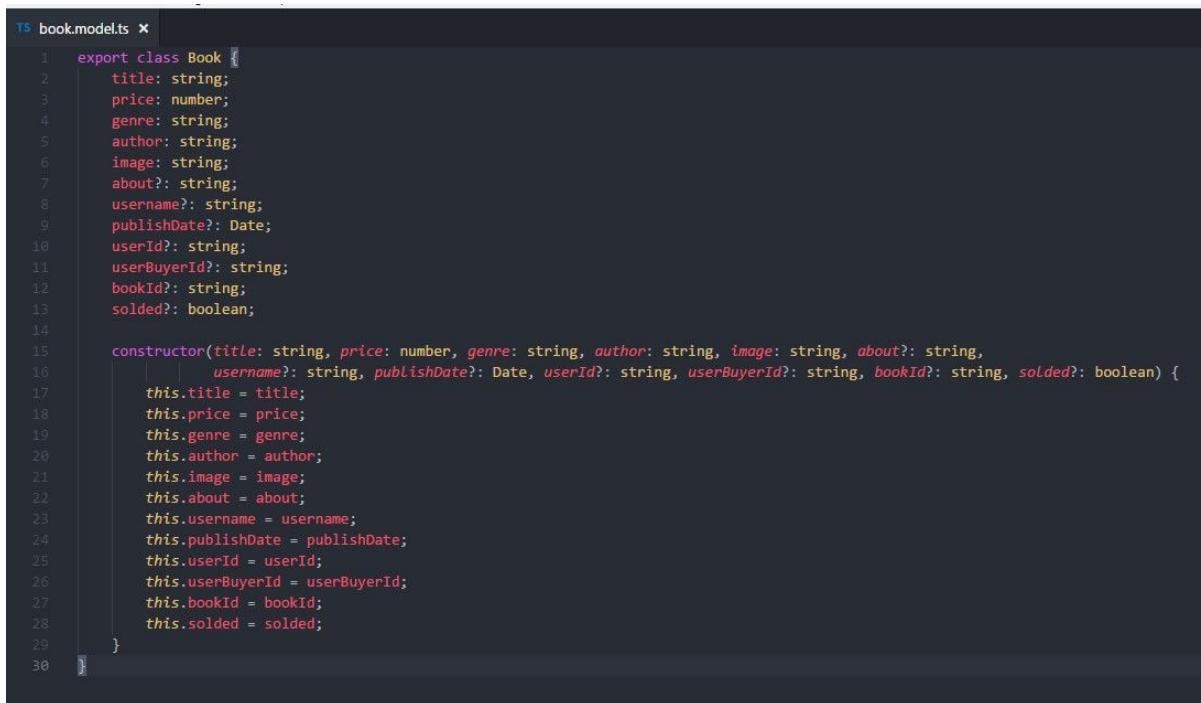
Slika 31. app.component.html fajl

Ovaj fajl će prikazivati renderovane komponente u zavisnosti od rute. Takođe definiše osnovni raspored komponenti.

3.1 Modeli

U aplikaciji postoje 5 modela i to:

1. book.model.ts - model koji reprezentuje knjigu
2. user.model.ts - model koji reprezentuje korisnika
3. post.model.ts - model koji reprezentuje objavu
4. comment.model.ts - model koji reprezentuje komentar
5. error.model.ts - model koji reprezentuje grešku



```
ts book.model.ts ✘
1  export class Book {
2    title: string;
3    price: number;
4    genre: string;
5    author: string;
6    image: string;
7    about?: string;
8    username?: string;
9    publishDate?: Date;
10   userId?: string;
11   userBuyerId?: string;
12   bookId?: string;
13   solded?: boolean;
14
15  constructor(title: string, price: number, genre: string, author: string, image: string, about?: string,
16             username?: string, publishDate?: Date, userId?: string, userBuyerId?: string, bookId?: string, solded?: boolean) {
17    this.title = title;
18    this.price = price;
19    this.genre = genre;
20    this.author = author;
21    this.image = image;
22    this.about = about;
23    this.username = username;
24    this.publishDate = publishDate;
25    this.userId = userId;
26    this.userBuyerId = userBuyerId;
27    this.bookId = bookId;
28    this.solded = solded;
29  }
30 }
```

Slika 32. model knjige

```
TS comment.model.ts ✘
1  export class Comment {
2    content: string;
3    username: string;
4    grade: number;
5    tag: string;
6    publishDate?: Date;
7    bookId?: string;
8    commentId?: string;
9
10   constructor(content: string, username: string, grade: number, tag: string, publishDate?: Date, bookId?: string,
11               commentId?: string) {
12     this.content = content;
13     this.username = username;
14     this.grade = grade;
15     this.tag = tag;
16     this.publishDate = publishDate;
17     this.bookId = bookId;
18     this.commentId = commentId;
19   }
20 }
```

Slika 33. model komentara

```
TS error.model.ts ✘
1  export class Error {
2    constructor(public title: string, public message: string) {}
3 }
```

Slika 34. model greške

```
TS post.model.ts ✘
1  export class Post {
2      title: string;
3      content: string;
4      username: string;
5      userId?: string;
6      postId?: string;
7
8      constructor(title: string, content: string, username: string, postId?: string, userId?: string) {
9          this.title = title;
10         this.content = content;
11         this.username = username;
12         this.userId = userId;
13         this.postId = postId;
14     }
15 }
```

Slika 35. model objave

```
TS user.model.ts ✘
1  export class User {
2      constructor(public email: string, public password: string, public firstName?: string,
3                  public lastName?: string, public country?: string, public city?: string,
4                  public postalCode?: string, public address?: string, public phoneNumber?: string,
5                  public image?: string) {}
6  }
```

Slika 36. model korisnika

3.2 Servisi

Servisi služe za centralizaciju funkcionalnosti. Smanjuje se duplicitanje koda kroz centralizaciju, prosleđuju se samo instance servisa specijalizovanog za određenu logiku unutar aplikacije. Servisi se često koriste za rukovanje podacima – centralizujemo logiku za rukovanje podacima kroz servis.

3.2.1 HTTP Zahtevi

Ne renderuje se serverski nova stranica, podaci se šalju koriz XMLHttpRequest objekat-AJAX zahtevi. Koriste se Observables i Observable patern. Observable služi za asinhronne zahteve, osluškuje različite izvore. Javlja Observer-u kada dođe do promene od interesa. Angular koristi observables u pozadini za zahteve - http.post kreira observable. subscribe na http observable - rukujemo podacima poslanim od Observable-a što je slično kao callbacks ili promises.

Observable HTTP zahtevi – RxJS Svi zahtevi ka backend-u, šalju token kao query parametar.

U aplikaciji postoje pet servisa i to:

1. auth.service.ts - servis za rukovanje autentifikacijom korisnika
2. post.service.ts - servis za rukovanje objavama
3. comment.service.ts - servis za rukovanje komentarima
4. book.service.ts - servis za rukovanje knjigama
5. error.service.ts - zaseban servis i komponenta za rukovanje greskama na nivou aplikacije

Svaki servis poseduje odgovarajuće operacije koje su već navedene prilikom definisanja ruta za korisnika, knjigu, objavu i komentar.

```
auth.service.ts ✘

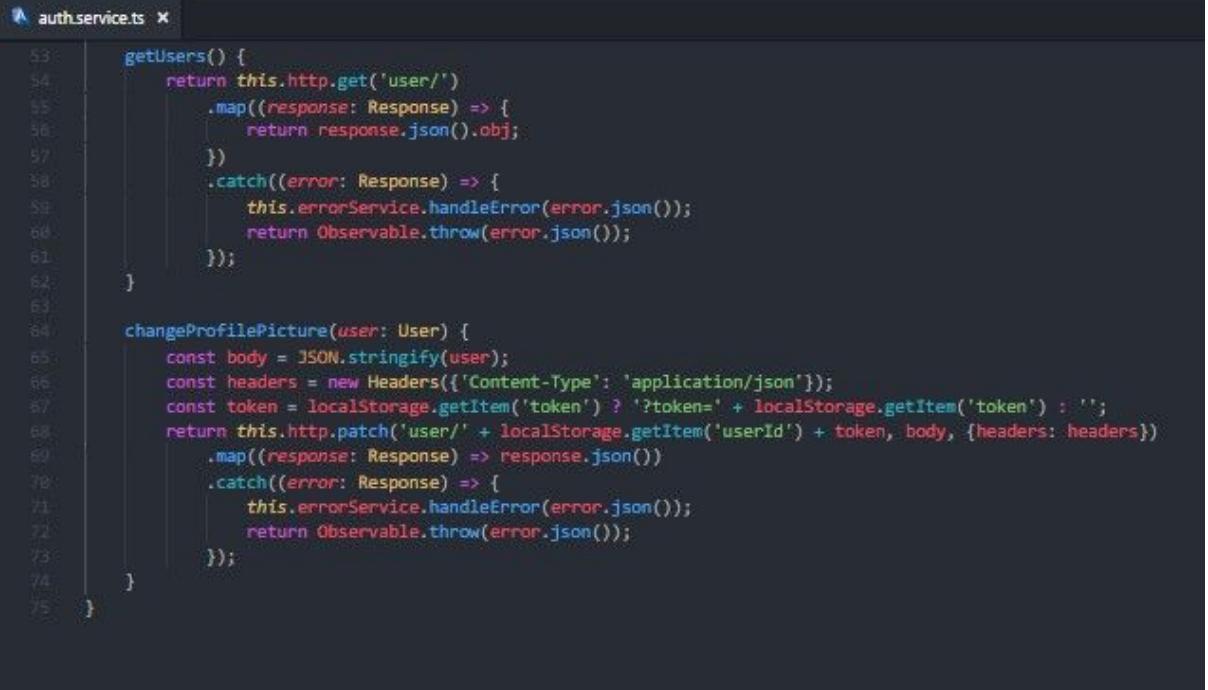
1 import { Injectable } from '@angular/core';
2 import { Http, Headers, Response } from '@angular/http';
3 import { Observable } from 'rxjs';
4 import { User } from '../models/user.model';
5 import { ErrorService } from '../services/error.service';
6 import 'rxjs/Rx';
7
8 @Injectable()
9 export class AuthService {
10   user: User;
11
12   constructor(private http: Http, private errorService: ErrorService) {}
13
14   register(user: User) {
15     const body = JSON.stringify(user);
16     const headers = new Headers({ 'Content-Type': 'application/json' });
17     return this.http.post('user/register', body, {headers: headers})
18       .map((response: Response) => response.json())
19       .catch((error: Response) => {
20         this.errorService.handleError(error.json());
21         return Observable.throw(error.json());
22       });
23   }
24
25   login(user: User) {
26     const body = JSON.stringify(user);
27     const headers = new Headers({ 'Content-Type': 'application/json' });
28     return this.http.post('user/login', body, {headers: headers})
29       .map((response: Response) => response.json())
30       .catch((error: Response) => {
31         this.errorService.handleError(error.json());
32         return Observable.throw(error.json());
33       });
34   }
35
36   logout() {
37     localStorage.clear();
38   }
39
40   isLoggedIn() { return localStorage.getItem('token') !== null; }
41
42   getUser(userID: string) {
43     return this.http.get('user/' + userID)
44       .map((response: Response) => {
45         return response.json().obj;
46       })
47       .catch((error: Response) => {
48         this.errorService.handleError(error.json());
49         return Observable.throw(error.json());
50       });
51   }
52 }
```

Slika 37. servis korisnika

Metoda `logout()` vrši odjavljivanje korisnika brisanje njegovih informacija(tokena i id) iz `localStorage-a`.

Metoda `isLoggedIn()` vrši proveru da li je korisnik trenutno ulogovan proveravajući da li je token različit od null.

Metoda register() vrši registrovanje korisnika, metoda login prijavljivanje, dok metoda getUser vraća specifičnog korisnika.

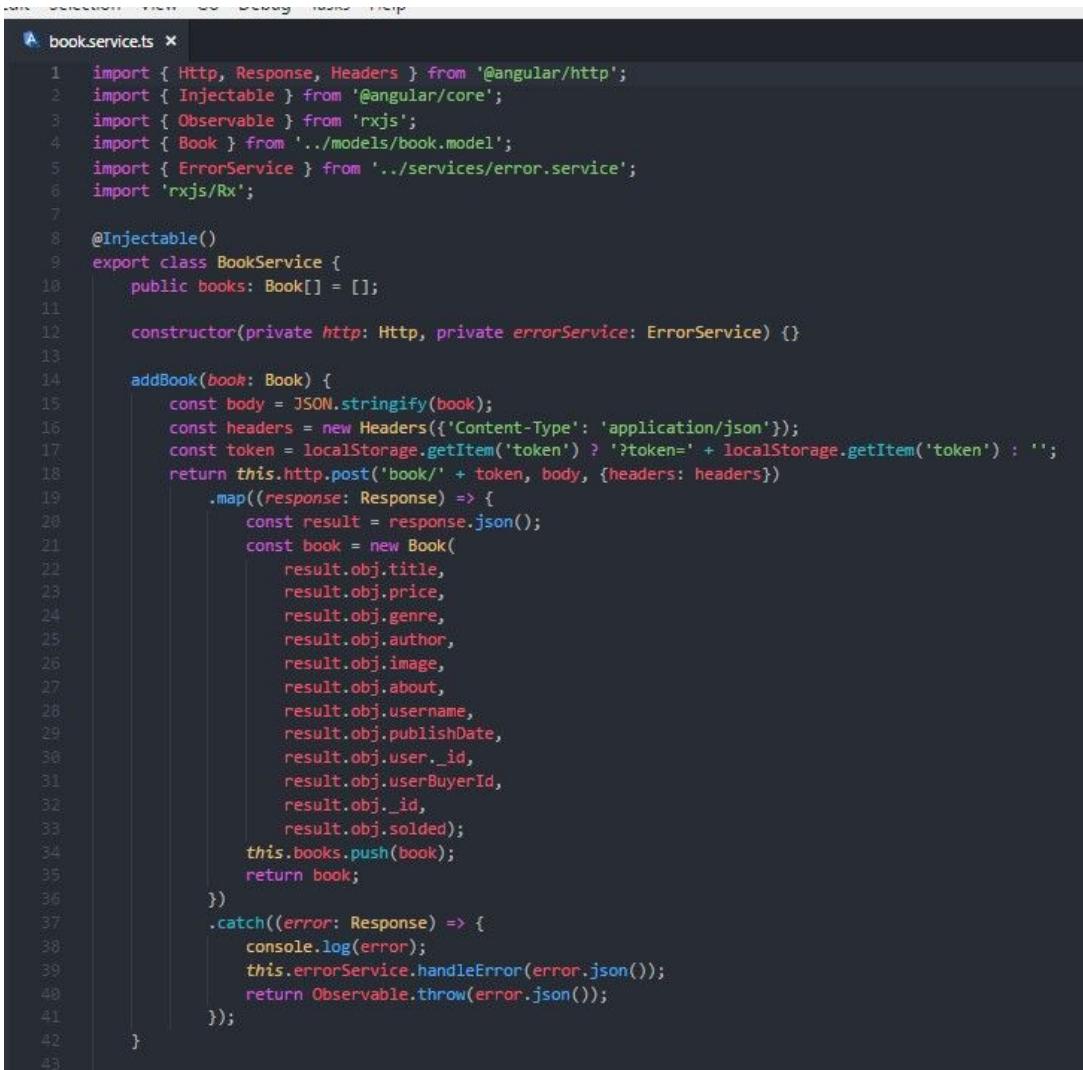


```
53  getUsers(): Observable<User> {
54    return this.http.get('user/')
55      .map((response: Response) => {
56        return response.json().obj;
57      })
58      .catch((error: Response) => {
59        this.errorService.handleError(error.json());
60        return Observable.throw(error.json());
61      });
62  }
63
64  changeProfilePicture(user: User): Observable<User> {
65    const body = JSON.stringify(user);
66    const headers = new Headers({'Content-Type': 'application/json'});
67    const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
68    return this.http.patch('user/' + localStorage.getItem('userId') + token, body, {headers: headers})
69      .map((response: Response) => response.json())
70      .catch((error: Response) => {
71        this.errorService.handleError(error.json());
72        return Observable.throw(error.json());
73      });
74  }
75}
```

Slika 38. servis korisnika

Metoda getUsers() služi za prikaz svih korisnika administratoru.

Metoda changeProfilePicture() služi za promenu profilne slike.



The screenshot shows a code editor window with the file 'book.service.ts' open. The code is written in TypeScript and defines a service for managing books. It imports necessary modules from Angular and includes logic for adding a book to the service's internal array and handling errors.

```
1 import { Http, Response, Headers } from '@angular/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { Book } from '../models/book.model';
5 import { ErrorService } from '../services/error.service';
6 import 'rxjs/Rx';
7
8 @Injectable()
9 export class BookService {
10   public books: Book[] = [];
11
12   constructor(private http: Http, private errorService: ErrorService) {}
13
14   addBook(book: Book) {
15     const body = JSON.stringify(book);
16     const headers = new Headers({ 'Content-Type': 'application/json' });
17     const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
18     return this.http.post('book/' + token, body, {headers: headers})
19       .map((response: Response) => {
20         const result = response.json();
21         const book = new Book(
22           result.obj.title,
23           result.obj.price,
24           result.obj.genre,
25           result.obj.author,
26           result.obj.image,
27           result.obj.about,
28           result.obj.username,
29           result.obj.publishDate,
30           result.obj.user._id,
31           result.obj.userBuyerId,
32           result.obj._id,
33           result.obj.solded);
34         this.books.push(book);
35         return book;
36       })
37       .catch((error: Response) => {
38         console.log(error);
39         this.errorService.handleError(error.json());
40         return Observable.throw(error.json());
41       });
42   }
43 }
```

Slika 39. servis knjiga

Metoda addBook() služi za dodavanje nove knjige.

The screenshot shows a code editor window with the file 'book.service.ts' open. The code is written in TypeScript and defines a service for managing books. The 'getBooks()' method sends a GET request to the '/book' endpoint, maps the response to an array of transformed books, and handles errors by catching them and throwing the error's JSON representation.

```
44     getBooks() {
45         return this.http.get('book/')
46             .map((response: Response) => {
47                 const books = response.json().obj;
48                 let transformedBooks: Book[] = [];
49                 for (let book of books) {
50                     transformedBooks.push(new Book(
51                         book.title,
52                         book.price,
53                         book.genre,
54                         book.author,
55                         book.image,
56                         book.about,
57                         book.username,
58                         book.publishDate,
59                         book.user._id,
60                         book.userBuyerId,
61                         book._id,
62                         book.solded)
63                 );
64             }
65             this.books = transformedBooks;
66             return transformedBooks;
67         })
68         .catch((error: Response) => {
69             this.errorService.handleError(error.json());
70             return Observable.throw(error.json());
71         });
72     }
73 }
```

Slika 40. servis knjiga

Metoda `getBooks()` služi za prikaz svih korisnikovih knjiga.

```
book.service.ts x

74  getBooksHome() {
75    return this.http.get('home/')
76      .map((response: Response) => {
77        const books = response.json().obj;
78        let transformedBooks: Book[] = [];
79        for (let book of books) {
80          transformedBooks.push(new Book(
81            book.title,
82            book.price,
83            book.genre,
84            book.author,
85            book.image,
86            book.about,
87            book.username,
88            book.publishDate,
89            book.user._id,
90            book.userBuyerId,
91            book._id,
92            book.solded));
93        }
94        this.books = transformedBooks;
95        return transformedBooks;
96      })
97      .catch((error: Response) => {
98        this.errorService.handleError(error.json());
99        return Observable.throw(error.json());
100      });
101  }
102 }
```

Slika 41. servis knjiga

Metoda getBooksHome() služi za prikaz svih knjiga svih korinika na početno strani.

```
183     getBook(bookId: string) {
184         return this.http.get('book/' + bookId)
185             .map((response: Response) => {
186                 const book = response.json().obj;
187                 let transformedBook: Book = new Book(
188                     book.title,
189                     book.price,
190                     book.genre,
191                     book.author,
192                     book.image,
193                     book.about,
194                     book.username,
195                     book.publishDate,
196                     book.user._id,
197                     book.userBuyerId,
198                     book._id,
199                     book.solded);
200                 return transformedBook;
201             })
202             .catch((error: Response) => {
203                 this.errorService.handleError(error.json());
204                 return Observable.throw(error.json());
205             });
206     }
207
208     updateBook(book: Book) {
209         const body = JSON.stringify(book);
210         const headers = new Headers({'Content-Type': 'application/json'});
211         const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
212         return this.http.patch('book/' + book.bookId + token, body, {headers: headers})
213             .map((response: Response) => response.json())
214             .catch((error: Response) => {
215                 this.errorService.handleError(error.json());
216                 return Observable.throw(error.json());
217             });
218     }
219
220     updateBookAdmin(book: Book) {
221         const body = JSON.stringify(book);
222         const headers = new Headers({'Content-Type': 'application/json'});
223         const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
224         return this.http.patch('book/admin/' + book.bookId + token, body, {headers: headers})
225             .map((response: Response) => response.json())
226             .catch((error: Response) => {
227                 this.errorService.handleError(error.json());
228                 return Observable.throw(error.json());
229             });
230     }
231 }
```

Slika 42. servis knjiga

Metoda getBook() služi za prikaz određene knjige.

Metoda updateBook() služi za ažuriranje knjige od strane korisnika. Korisnik može da ažurira samo svoje knjige ukoliko je registrovan.

Metoda updateBookAdmin() služi za ažuriranje knjige od strane admina. Admin može da ažurira sve knjige.

```
151     updateBookSold(book: Book) {
152         const body = JSON.stringify(book);
153         const headers = new Headers({ 'Content-Type': 'application/json' });
154         const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
155         return this.http.patch('book/sold/' + book.bookId + token, body, {headers: headers})
156             .map((response: Response) => response.json())
157             .catch((error: Response) => {
158                 this.errorService.handleError(error.json());
159                 return Observable.throw(error.json());
160             });
161     }
162 }
163
164 deleteBook(book: Book) {
165     this.books.splice(this.books.indexOf(book), 1);
166     const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
167     return this.http.delete('book/' + book.bookId + token)
168         .map((response: Response) => response.json())
169         .catch((error: Response) => {
170             this.errorService.handleError(error.json());
171             return Observable.throw(error.json());
172         });
173 }
174
175 deleteBookAdmin(book: Book) {
176     this.books.splice(this.books.indexOf(book), 1);
177     const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
178     return this.http.delete('book/admin/' + book.bookId + token)
179         .map((response: Response) => response.json())
180         .catch((error: Response) => {
181             this.errorService.handleError(error.json());
182             return Observable.throw(error.json());
183         });
184 }
185 }
```

Slika 43. servis knjiga

Metoda updateSold() služi za ažuriranje knjige prilikom njene kupovine od strane nekog korisnika.

Metoda deleteBook() služi za brisanje knjige od strane nekog korisnika. Svaki korisnik može da briše jedino svoje knjige (knjige koje je on postavio).

Metoda deleteBookAdmin() služi za brisanje knjige od strane administratora. Administrator može da briše knjige svih korisnika.

```
comment.service.ts ×

1 import { Http, Response, Headers } from '@angular/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { Comment } from '../models/comment.model';
5 import { ErrorService } from '../services/error.service';
6 import 'rxjs/Rx';
7
8 @Injectable()
9 export class CommentService {
10   private comments: Comment[] = [];
11   comment: Comment;
12
13   constructor(private http: Http, private errorService: ErrorService) {}
14
15   addComment(comment: Comment, bookId: string) {
16     const body = JSON.stringify(comment);
17     const headers = new Headers({ 'Content-Type': 'application/json' });
18     const token = localStorage.getItem('token') ? `token=${localStorage.getItem('token')}` : '';
19     return this.http.post(`comment/${bookId}/${token}`, body, {headers: headers})
20       .map((response: Response) => {
21         const result = response.json();
22         const comment = new Comment(
23           result.obj.content,
24           result.obj.username,
25           result.obj.grade,
26           result.obj.tag,
27           result.obj.publishDate,
28           result.obj.book._id,
29           result.obj._id);
30         this.comments.push(comment);
31         return comment;
32       })
33       .catch((error: Response) => {
34         this.errorService.handleError(error.json());
35         return Observable.throw(error.json());
36       });
37   }
38 }
```

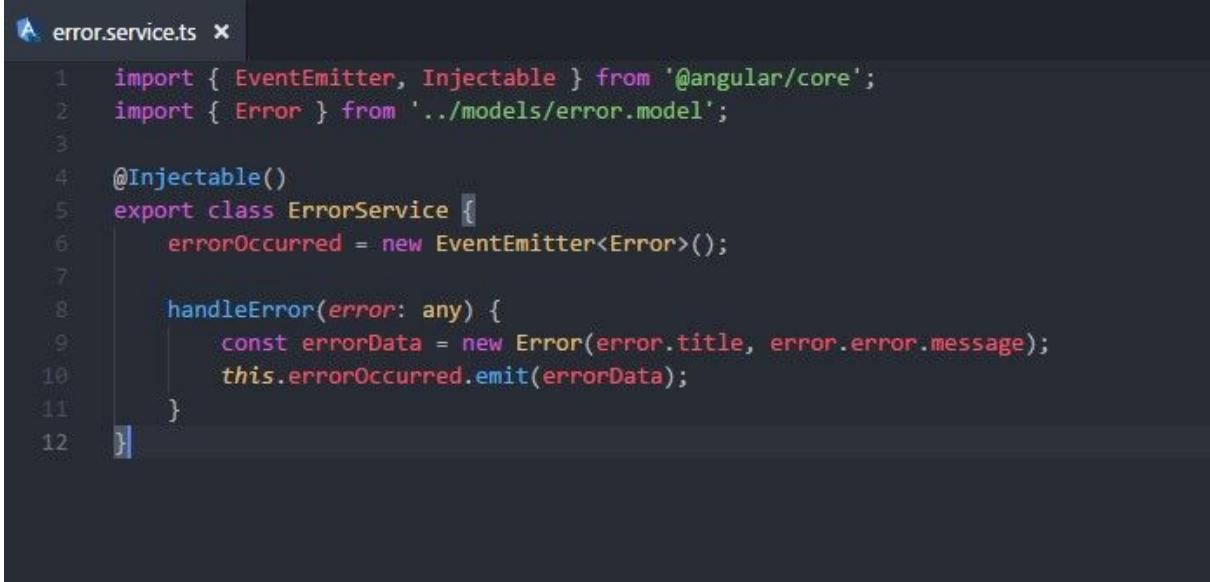
44. servis komentara

Metoda addComment() služi za dodavanje komentara na neku knjigu, o knjizi ili korisniku koji ju je postavio. Samo registrovani korisnici mogu komentarisati.

```
comment.service.ts
39     getComments(bookId: string) {
40         return this.http.get('comment/' + bookId)
41             .map((response: Response) => {
42                 const comments = response.json().obj;
43                 let transformedComments: Comment[] = [];
44                 for (let comment of comments) {
45                     transformedComments.push(new Comment(
46                         comment.content,
47                         comment.username,
48                         comment.grade,
49                         comment.tag,
50                         comment.publishDate,
51                         comment.book._id,
52                         comment._id));
53                 }
54                 this.comments = transformedComments;
55                 return transformedComments;
56             })
57             .catch((error: Response) => {
58                 this.errorService.handleError(error.json());
59                 return Observable.throw(error.json());
60             });
61     }
62 }
```

Slika 45. servis komentara

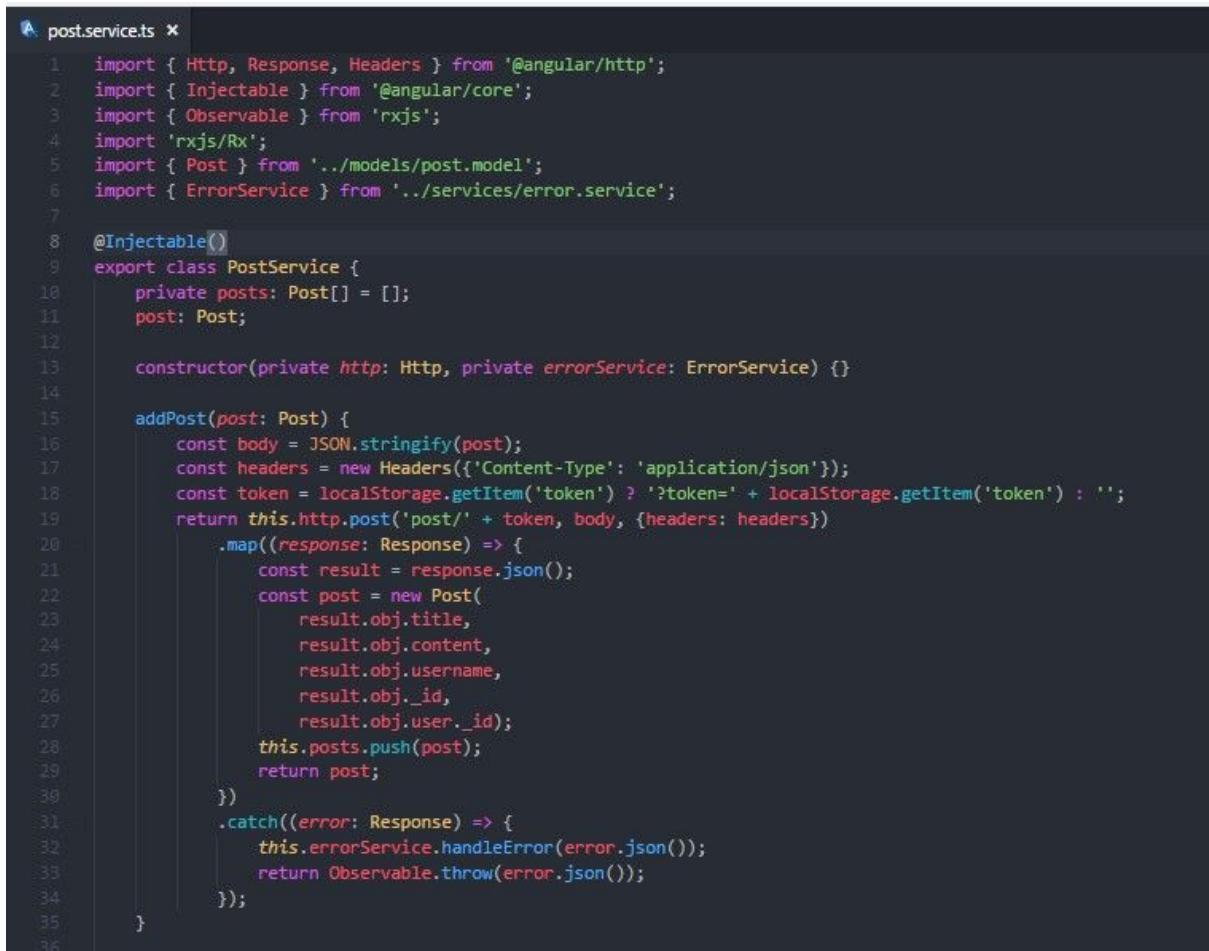
Metoda getComments() služi za prikaz svih komentara neke knjige.

A screenshot of a code editor showing a file named 'error.service.ts'. The code defines a service called 'ErrorService' that emits error events. It imports 'EventEmitter' and 'Injectable' from '@angular/core' and 'Error' from '../models/error.model'. The service has a private property 'errorOccurred' of type 'EventEmitter<Error>' and a public method 'handleError' which creates a new 'Error' object with 'title' and 'message' properties and emits it via 'errorOccurred'.

```
1 import { EventEmitter, Injectable } from '@angular/core';
2 import { Error } from '../models/error.model';
3
4 @Injectable()
5 export class ErrorService {
6   errorOccurred = new EventEmitter<Error>();
7
8   handleError(error: any) {
9     const errorData = new Error(error.title, error.error.message);
10    this.errorOccurred.emit(errorData);
11  }
12}
```

Slika 46. servis za rukovanje greškama

dodavanje novog Slika 47. Slika 47. servis Slika



```
post.service.ts ✘
1 import { Http, Response, Headers } from '@angular/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import 'rxjs/Rx';
5 import { Post } from '../models/post.model';
6 import { ErrorService } from '../services/error.service';
7
8 @Injectable()
9 export class PostService {
10   private posts: Post[] = [];
11   post: Post;
12
13   constructor(private http: Http, private errorService: ErrorService) {}
14
15   addPost(post: Post) {
16     const body = JSON.stringify(post);
17     const headers = new Headers({ 'Content-Type': 'application/json' });
18     const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
19     return this.http.post('post/' + token, body, {headers: headers})
20       .map((response: Response) => {
21         const result = response.json();
22         const post = new Post(
23           result.obj.title,
24           result.obj.content,
25           result.obj.username,
26           result.obj._id,
27           result.obj.user._id);
28         this.posts.push(post);
29         return post;
30       })
31       .catch((error: Response) => {
32         this.errorService.handleError(error.json());
33         return Observable.throw(error.json());
34       });
35   }
36 }
```

Slika 47. servis objava

Metoda addPost() služi za objavljivanje novih postova na blogu. Samo registrovani korisnici mogu objavljivati na blogu.

```
 37     getPosts() {
 38         return this.http.get('post/')
 39             .map((response: Response) => {
 40                 const posts = response.json().obj;
 41                 let transformedPosts: Post[] = [];
 42                 for (let post of posts) {
 43                     transformedPosts.push(new Post(
 44                         post.title,
 45                         post.content,
 46                         post.username,
 47                         post._id,
 48                         post.user._id)
 49                     );
 50                 }
 51                 this.posts = transformedPosts;
 52                 return transformedPosts;
 53             })
 54             .catch((error: Response) => {
 55                 this.errorService.handleError(error.json());
 56                 return Observable.throw(error.json());
 57             });
 58     }
 59
 60    getPost(postId : string){
 61         return this.http.get('post/' + postId)
 62             .map((response: Response) => {
 63                 const post = response.json().obj;
 64                 let transformedPost: Post = new Post(
 65                     post.title,
 66                     post.content,
 67                     post.username,
 68                     post._id,
 69                     post.user._id);
 70                 this.post = transformedPost;
 71                 return transformedPost;
 72             })
 73             .catch((error: Response) => {
 74                 this.errorService.handleError(error.json());
 75                 return Observable.throw(error.json());
 76             });
 77     }
 78
 79     deletePost(post: Post) {
 80         this.posts.splice(this.posts.indexOf(post), 1);
 81         const token = localStorage.getItem('token') ? '?token=' + localStorage.getItem('token') : '';
 82         return this.http.delete('post/' + post.postId + token)
 83             .map((response: Response) => response.json())
 84             .catch((error: Response) => {
 85                 this.errorService.handleError(error.json());
 86                 return Observable.throw(error.json());
 87             });
 88     }
}
```

Slika 48. servis objava

Metoda getPosts() služi za prikaz svih objava na forumu.

Metoda getPost() služi za prikaz jedne određene objave.

Metoda deletePost() služi za brisanje postova. Samo administrator može brisati postove (i to sve).

```
90     updatePost(post: Post) {
91         const body = JSON.stringify(post);
92         const headers = new Headers({'Content-Type': 'application/json'});
93         const token = localStorage.getItem('token') ? `?token=${localStorage.getItem('token')}` : '';
94         return this.http.patch(`post/${post.postId}${token}`, body, {headers: headers})
95             .map((response: Response) => response.json())
96             .catch((error: Response) => {
97                 this.errorService.handleError(error.json());
98                 return Observable.throw(error.json());
99             });
100    }
101 }
```

Slika 49. servis objava

Metoda updatePost() služi za izmenu objava na forumu. Samo administrator sistema može vršiti izmenu objava(i to svih) ukoliko uoči neki neprikladan sadržaj.

3.3 Komponente

Aplikacija se sastoji od ukupno 23 komponente i te komponente su sledeće:

- about - služi za prikaz stranice sa nekim osnovnim informacijama o sajtu
- add-book - reprezentuje formu za dodavanje nove knjige
- add-comment - reprezentuje formu za dodavanje novog komentara
- blog - služi za prikaz stranice za blog
- book - gradivna komponenta koja reprezentuje knjigu
- book-details - služi za prikaz stranice sa detaljima o odabranom knjizi
- change-profile-picture - služi za prikaz stranice koja omogućuje promenu profilne slike
- comment - gradivna komponenta koja reprezentuje komentar
- error - komponenta koja je namenjena za obradu grešaka
- header - služi za prikaz zaglavlja stranice
- home - služi za prikaz naslovne strana sajta
- login - reprezentuje formu za logovanje
- logout - služi za odjavljivanje sa sajta
- page-not-found - komponenta koja se prikazuje u slučaju pogrešno URL-a
- post - gradivna komponenta koja reprezentuje objavu
- post-details - služi za prikaz pojedinačne objave
- profile - služi za prikaz profilne stanice registrovanog korisnika
- publish - služi za objavljivanje novih postova
- register - reprezentuje formu za registraciju
- showinfo - služi za prikaz informacija o kupcu
- update-book - reprezentuje formu za ažuriranje knjige
- update-post - reprezentuje formu za ažuriranje posta
- users - služi za prikaz svih registrovanih korisnika (ova mogućnost je dostupna samo adminu)

U nastavku je dat prikaz svih navedenih komponenti koje se sastoje iz tri dela - htm, css i ts fajl.

```
5 about.component.html x
1  <div id="about" class="content text-center">
2    <div class="row">
3      <div class="col-sm-12">
4        
5        <hr>
6        <p>
7          The application is an example of a MEAN 2 web application with demonstrated CRUD operations and
8          authentication. It was developed for the needs of a student project and the purchase and sale of books
9          is not real.
10         </p>
11       </div>
12     </div>
13     <hr>
14     <div class="row">
15       <div class="col-sm-4">
16         Made by Vukan Markovic.<br>
17         Code released under the MIT License.
18       </div>
19       <div class="col-sm-4">
20         <br>
21         Book Trading Club App&copy;;
22       </div>
23       <div class="col-sm-4">
24         <br>
25         
26         <a href="https://github.com/Vukan-Markovic/Book-trading-club" target="blank" title="Github" style="text-decoration: none">
27           See code here</a>
28       </div>
29     </div>
30   </div>
```

Slika 50. about.component.html

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-about',
5   templateUrl: './about.component.html',
6   styleUrls: ['./about.component.css']
7 })
8
9 export class AboutComponent implements OnInit {
10   constructor() {}
11   ngOnInit() {}
12 }
```

Slika 51. about.component.ts

```
1 <form id="f" (ngSubmit)="onSubmit()" #f="ngForm">
2   <fieldset>
3     <legend>Add a new book</legend>
4     <legend>
5       The legend element represents a caption for the rest of the contents of the
6       legend element's parent fieldset element, if any.
7     </legend>
8     <div class="form-group">
9       <label for="price">Price:</label>
10      <input id="price" name="price" [ngModel]="book?.price" class="form-control" placeholder="Enter the price of the book" type="number" required min="0">
11    </div>
12    <div class="form-group">
13      <label for="genres">Genre:</label>
14      <select id="genre" name="genre" [ngModel]="book?.genre" class="form-control" required>
15        <option selected>Arts & Photography</option>
16        <option>Biographies & Memoirs</option>
17        <option>Business & Investing</option>
18        <option>Children's Books</option>
19        <option>History</option>
20        <option>Literature & Suspense</option>
21        <option>Romance</option>
22        <option>Sci-Fi & Fantasy</option>
23        <option>Teens & Young Adult</option>
24      </select>
25    </div>
26    <div class="form-group">
27      <label for="author">Author:</label>
28      <input name="author" id="author" [ngModel]="book?.author" class="form-control" placeholder="Enter the author of the book" type="text" required>
29    </div>
30    <div class="form-group">
31      <label for="about">Write a couple of words about the book if you like:</label>
32      <textarea id="about" name="about" [ngModel]="book?.about" class="form-control" rows="3" placeholder="Your comment about book" maxlength="80"></textarea>
33    </div>
34    <div class="form-group">
35      <label for="image">Image:</label>
36      <input id="image" name="image" [ngModel]="book?.image" class="form-control" placeholder="Enter the image url of the book cover" type="url">
37      <a href="https://www.google.rs/search?tbm=isch&q={{f.value.title}}+book+cover" target="_blank">Get book cover</a>
38    </div>
39    <button type="button" class="btn btn-lg btn-danger" (click)="onClear()">Clear</button>
40    <button class="btn btn-lg btn-success" type="submit" [disabled]!="form.valid">Add a book</button>
41  </fieldset>
42 </form>
```

Slika 52. add-book.component.html

```

❶ add-book.components.ts
1 import { Component, OnInit, ViewChild } from '@angular/core';
2 import { NgForm } from '@angular/forms';
3 import { BookService } from '../../../../../services/book.service';
4 import { Book } from '../../../../../models/book.model';
5 import { Router } from '@angular/router';
6 import { AuthService } from '../../../../../services/auth.service';
7 import { User } from '../../../../../models/user.model';
8 import { FlashMessagesService } from 'angular2-flash-messages';
9
10 @Component({
11   selector: 'app-add-book',
12   templateUrl: './add-book.component.html',
13   styleUrls: ['./add-book.component.css']
14 })
15
16 export class AddBookComponent implements OnInit {
17   @ViewChild('f') form: NgForm;
18   book: Book;
19   user: any = {};
20
21   constructor(private bookService: BookService, private router: Router,
22     private authService: AuthService, private flashMessage: FlashMessagesService) {}
23
24   ngOnInit() {
25     if (!this.authService.isLoggedIn())
26       return this.router.navigate(['/login']);
27     this.authService.getUser(localStorage.getItem('userId'))
28       .subscribe((user: User) => {
29       this.user = user;
30     });
31   }
32
33   onSubmit() {
34     const book = new Book(this.form.value.title, this.form.value.price, this.form.value.genre, this.form.value.author, this.form.value.image, this.form.value.about, this.user.firstName);
35
36     this.bookService.addBook(book)
37       .subscribe(
38         data => {
39           console.log(data),
40           this.flashMessage.show('New book is added!', { cssClass: 'alert-success', timeout: 3000 });
41         },
42         error => console.error(error));
43     this.form.resetForm();
44     this.router.navigate(['/profile']);
45   }
46
47   onClear() {
48     this.book = null;
49     this.form.resetForm();
50   }
51 }

```

Slika 53. add-book.component.ts

```
⑥ add-comment.component.html ✘
1 ① <form id="f" (ngSubmit)="onSubmit()" #f="ngForm">
2 ②   <fieldset>
3     <legend>Add a new comment</legend>
4     <div class="form-group">
5       <label for="author">This comment is about:</label>
6       <select [ngModel]="comment?.tag" class="form-control" name="tag" id="tag" required>
7         <option selected>Book</option>
8         <option>User</option>
9         </select>
10    </div>
11    <div class="form-group">
12      <label for="content">Content:</label>
13      <input id="content" name="content" [ngModel]="comment?.content" class="form-control" placeholder="Enter the content of the comment" type="text" required
14        max="40">
15    </div>
16    <div class="form-group">
17      <label for="grade">Grade:</label>
18      <input id="grade" name="grade" [ngModel]="comment?.grade" class="form-control" placeholder="Enter the grade" type="number" required min="0" max="10">
19    </div>
20    <button type="button" class="btn btn-lg btn-danger" (click)="onClear()>Clear</button>
21    <button class="btn btn-lg btn-success" type="submit" [disabled]="!form.valid">Add a comment</button>
22  </fieldset>
23 </form>
```

Slika 54. add-comment.component.html

```

  add-comment.components.x
  import { Component, OnInit, ViewChild } from '@angular/core';
  import { NgForm } from '@angular/forms';
  import { CommentService } from '../../../../../services/comment.service';
  import { Router, ActivatedRoute } from '@angular/router';
  import { AuthService } from '../../../../../services/auth.service';
  import { User } from '../../../../../models/user.model';
  import { Comment } from '../../../../../models/comment.model';
  import { FlashMessagesService } from 'angular2-flash-messages';

  @Component({
    selector: 'app-add-comment',
    templateUrl: './add-comment.component.html',
    styleUrls: ['./add-comment.component.css']
  })

  export class AddCommentComponent implements OnInit {
    @ViewChild('f') form : NgForm;
    comment: Comment;
    user: any = {};
    book: any = {};

    constructor(private commentService: CommentService, private router: Router,
      private authService: AuthService, private flashMessage: FlashMessagesService,
      private route: ActivatedRoute) {}

    ngOnInit() {
      if (!this.authService.isLoggedIn())
        return this.router.navigate(['/login']);

      this.authService.getUser(localStorage.getItem('userId'))
        .subscribe((user: User) => {
          this.user = user;
        })
    }

    onSubmit() {
      const comment = new Comment(
        this.form.value.content,
        this.user.firstName,
        this.form.value.grade,
        this.form.value.tag);

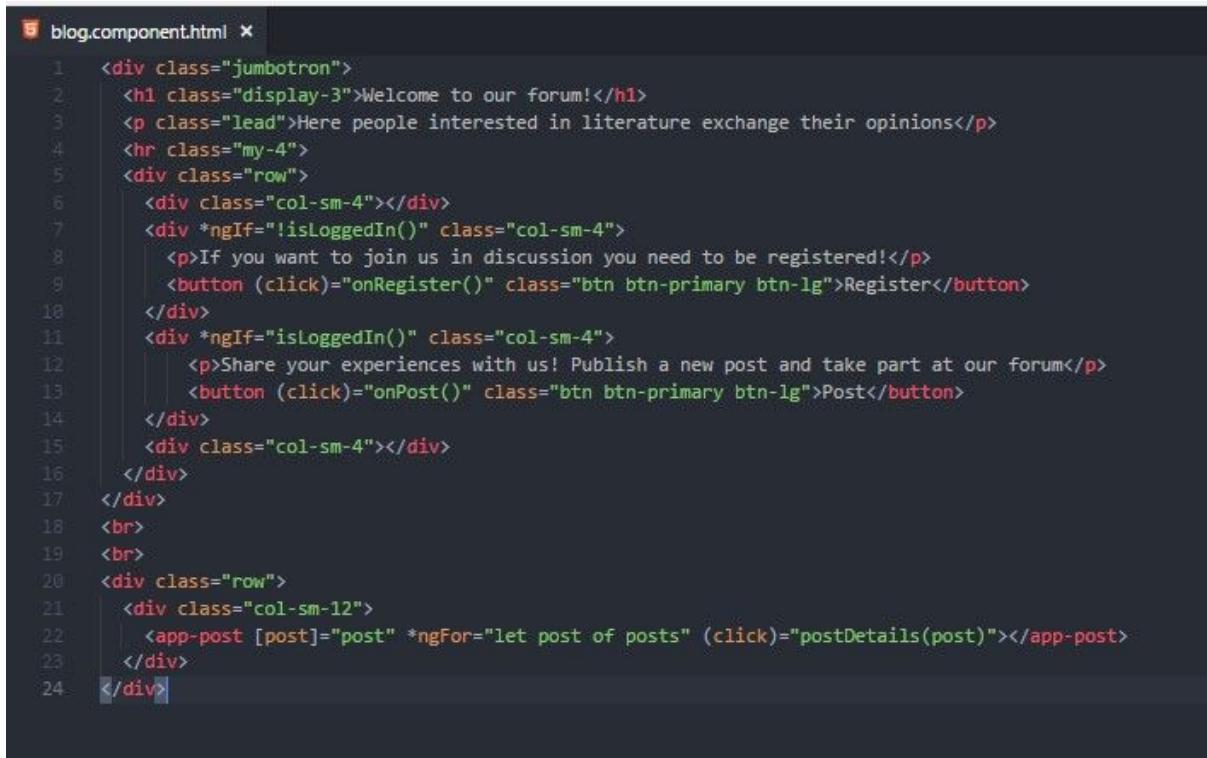
      this.commentService.addComment(comment, this.route.snapshot.params['id'])
        .subscribe(
          data => {
            console.log(data),
            this.flashMessage.show('New comment is added!', { cssClass: 'alert-success', timeout: 3000 });
          },
          error => console.error(error));

      this.form.resetForm();
      this.router.navigate(['/book-details/'], this.route.snapshot.params['id']);
    }

    onClear() {
      this.comment = null;
      this.form.resetForm();
    }
  }

```

Slika 55. add-comment.component.ts



The screenshot shows a code editor window with the file 'blog.component.html' open. The code is written in HTML and includes some Angular-specific directives like *ngIf and (click). The content of the component includes a jumbotron section, a row with two columns for users who are not logged in, another row with a column for logged-in users, and a final row with a single column containing multiple app-post components.

```
1 <div class="jumbotron">
2   <h1 class="display-3">Welcome to our forum!</h1>
3   <p class="lead">Here people interested in literature exchange their opinions</p>
4   <hr class="my-4">
5   <div class="row">
6     <div class="col-sm-4"></div>
7     <div *ngIf="!isLoggedIn()" class="col-sm-4">
8       <p>If you want to join us in discussion you need to be registered!</p>
9       <button (click)="onRegister()" class="btn btn-primary btn-lg">Register</button>
10    </div>
11    <div *ngIf="isLoggedIn()" class="col-sm-4">
12      <p>Share your experiences with us! Publish a new post and take part at our forum</p>
13      <button (click)="onPost()" class="btn btn-primary btn-lg">Post</button>
14    </div>
15    <div class="col-sm-4"></div>
16  </div>
17  <br>
18  <br>
19  <div class="row">
20    <div class="col-sm-12">
21      <app-post [post]="post" *ngFor="let post of posts" (click)="postDetails(post)"></app-post>
22    </div>
23  </div>
24</div>
```

Slika 56.blog.component.html

```
blog.component.ts ✘

1 import { Component, OnInit } from '@angular/core';
2 import { Router } from '@angular/router';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { PostService } from '../../../../../services/post.service';
5 import { Post } from '../../../../../models/post.model';
6
7 @Component({
8   selector: 'app-blog',
9   templateUrl: './blog.component.html'
10 })
11
12 export class BlogComponent implements OnInit {
13   posts: Post [] = [];
14
15   constructor(private router: Router, private authService: AuthService, private postService: PostService) { }
16
17   ngOnInit() {
18     this.postService.getPosts()
19       .subscribe(posts: Post[]) => {
20         this.posts = posts;
21       }
22     );
23   }
24
25   onPost() {
26     this.router.navigate(['/publish']);
27   }
28
29   onRegister() {
30     this.router.navigate(['/register']);
31   }
32
33   postDetails(post: Post) {
34     this.router.navigate(['/post-details', post.postId]);
35   }
36
37   isLoggedIn () {
38     return this.authService.isLoggedIn();
39   }
40 }
```

Slika 58. blog.component.ts

```
book.component.html ✘

1 <div id="book" (click)="bookDetails()" *ngIf="belongsToUser()">
2   <span class="badge badge-success">{{ book.title }}</span>
3   <span class="badge badge-info">{{ book.price | currency: "usd": 3 }}</span>
4   <span class="badge badge-success">{{ book.author }}</span>
5   <span class="badge badge-danger">{{ book.genre }}</span>
6   <span class="badge badge-info">{{ book.publishDate | date:"d/M/y" }}</span>
7   <div class="btn-group">
8     &nbsp;&nbsp;<button class="btn btn-warning" (click)="onUpdate(book)">Update</button>
9     <button class="btn btn-danger" (click)="onDelete()">Delete</button>
10    <button (click)="showInfo()" class="btn btn-info" *ngIf='isBookSolded()'>Solded</button>
11   </div>
12 </div>
```

Slika 59. book.component.html

```
book.component.ts ✘
1 import { Component, OnInit, Input } from '@angular/core';
2 import { Book } from '../../../../../models/book.model';
3 import { BookService } from '../../../../../services/book.service';
4 import { Router } from '@angular/router';
5 import { FlashMessagesService } from 'angular2-flash-messages';
6 import { AuthService } from '../../../../../services/auth.service';
7
8 @Component({
9   selector: 'app-book',
10   templateUrl: './book.component.html',
11   styleUrls: ['./book.component.css'],
12 })
13
14 export class BookComponent implements OnInit {
15   @Input() book: Book;
16
17   constructor(private bookService: BookService, private router: Router, private flashMessage: FlashMessagesService, private authService: AuthService) {}
18
19   ngOnInit() {}
20
21   onUpdate(book: Book) {
22     this.router.navigate(['/update-book', book.bookId]);
23   }
24
25   onDelete() {
26     this.bookService.deleteBook(this.book)
27       .subscribe(
28         result => {
29           console.log(result);
30           this.flashMessage.show('Book is deleted!', { cssClass: 'alert-danger', timeout: 3000 });
31         },
32         error => console.error(error));
33   }
34
35   belongsToUser() {
36     if (this.authService.isLoggedIn())
37       return localStorage.getItem('userId') === this.book.userId;
38     return false;
39   }
40
41   isBookSolved() {
42     return this.book.solved;
43   }
44
45   bookDetails() {
46     this.router.navigate(['/book-details/', this.book.bookId]);
47   }
48
49   showInfo() {
50     this.router.navigate(['/show-info/', this.book.userBuyerId]);
51   }
52 }
```

Slika 60. book.component.ts

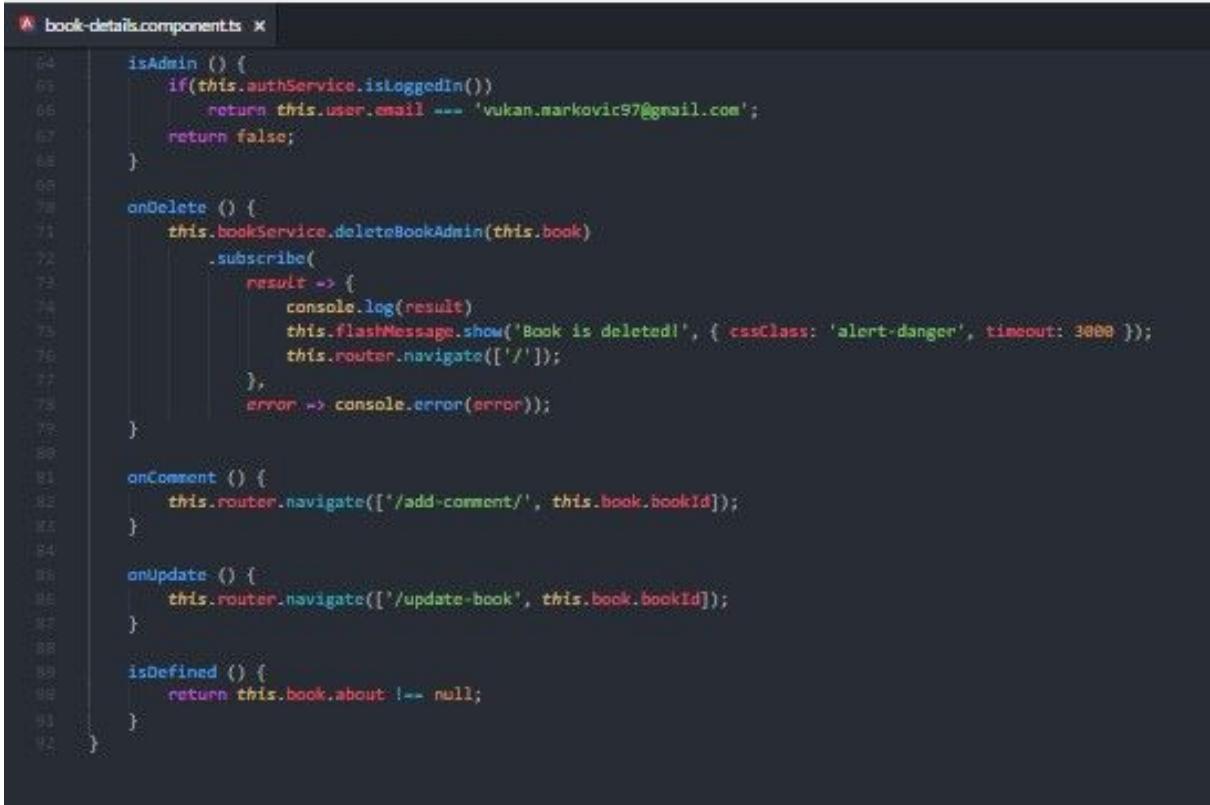
```
book-details.component.html X
1  <article>
2    <div class="card mb-3">
3      <h3 class="card-header"><span id="text">Title:</span> {{ book.title | uppercase }} &reg;</h3>
4      <div class="card-body">
5        <h5 class="card-title"><span id="text">Author:</span> {{ book.author }}</h5>
6        <h6 class="card-subtitle text-muted"><span id="text">Price:</span> {{ book.price | currency: "USD": 3 }} </h6>
7      </div>
8      
10     <p class="card-text">
11       <span id="text">Genre:</span> {{ book.genre }}<br>
12       <span id="text" *ngIf="isDefined()">About:</span> {{ book.about }}<br>
13       <span id="text">User:</span> {{ book.username }}<br>
14     </p>
15   </div>
16   <div class="card-body">
17     <button *ngIf="!belongsToUser()" class="card-link btn btn-lg btn-success" (click)="onBuy()">Buy</button>
18     <button *ngIf="isAdmin()" class="card-link btn btn-lg btn-warning" (click)="onUpdate()">Update</button>
19     <button *ngIf="isAdmin()" class="card-link btn btn-lg btn-danger" (click)="onDelete()">Delete</button>
20     <button *ngIf="!belongsToUser()" class="card-link btn btn-lg btn-danger" (click)="onComment()">Add comment</button>
21   </div>
22   <div class="card-footer text-muted">
23     <span id="text">Published:</span> {{ book.publishDate | date:"d/M/y" }}<br>
24     {{ book.publishDate | date:"shortTime" }}<br>
25   </div>
26 </div>
27 </article>
28 <div class="row">
29   <div class="col-sm-12">
30     <br><h2>Comments:</h2>
31     <app-comment [comment]="comment" *ngFor="let comment of comments"></app-comment>
32   </div>
33 </div>
```

Slika 61. book-details.component.html

```

book-details.components.ts | 1
1 import { Component, OnInit } from '@angular/core';
2 import { Book } from '../../models/book.model';
3 import { Comment } from '../../models/comment.model';
4 import { BookService } from '../../services/book.service';
5 import { AuthService } from '../../services/auth.service';
6 import { Router, ActivatedRoute } from '@angular/router';
7 import { FlashMessagesService } from 'angular2-flash-messages';
8 import { CommentService } from '../../services/comment.service';
9 import { User } from '../../models/user.model';
10
11 @Component({
12   selector: 'app-book-details',
13   templateUrl: './book-details.component.html',
14   styleUrls: ['./book-details.component.css'],
15 })
16 export class BookDetailsComponent implements OnInit {
17   book: any = {};
18   user: any = {};
19   comments: Comment[] = [];
20
21   constructor(private bookService: BookService, private authService: AuthService,
22     private router: Router, private flashMessage: FlashMessagesService,
23     private route: ActivatedRoute, private commentService: CommentService) {}
24
25   ngOnInit () {
26     this.bookService.getBook(this.route.snapshot.params['id'])
27       .subscribe(
28         (book: Book) => {
29           this.book = book;
30         }
31       );
32
33     this.commentService.getComments(this.route.snapshot.params['id'])
34       .subscribe((comments: Comment[]) => {
35       this.comments = comments;
36     })
37   );
38
39   if(this.authService.isLoggedIn()) {
40     this.authService.getUser(localStorage.getItem('userId'))
41       .subscribe((user: User) => {
42         this.user = user;
43       })
44   }
45 }
46
47 onBuy () {
48   this.book.userBuyerId = localStorage.getItem('userId');
49
50   this.bookService.updateBookSold(this.book)
51     .subscribe(result => console.log(result), error => console.error(error));
52
53   this.router.navigate(['/']);
54   this.flashMessage.show('Book is bought!', { cssClass: 'alert-success', timeout: 3000 });
55 }
56
57 belongsToUser () {
58   if(this.authService.isLoggedIn())
59     return this.user.firstName === this.book.username;
60   return true;
61 }

```



```
 54 isAdmin () {
  55   if(this.authService.isLoggedIn())
  56     return this.user.email === 'vukan.markovic97@gmail.com';
  57   return false;
  58 }
  59
  60 onDelete () {
  61   this.bookService.deleteBookAdmin(this.book)
  62     .subscribe(
  63       result => {
  64         console.log(result);
  65         this.flashMessage.show('Book is deleted!', { cssClass: 'alert-danger', timeout: 3000 });
  66         this.router.navigate(['/']);
  67       },
  68       error => console.error(error));
  69 }
  70
  71 onComment () {
  72   this.router.navigate(['/add-comment/', this.book.bookId]);
  73 }
  74
  75 onUpdate () {
  76   this.router.navigate(['/update-book', this.book.bookId]);
  77 }
  78
  79 isDefined () {
  80   return this.book.about !== null;
  81 }
  82 }
```

Slika 62. book-details.component.ts



```
 1 <form id="f" (ngSubmit)="onSubmit()" #f="ngForm">
  2   <fieldset>
  3     <legend>Update a profile picture</legend>
  4     <div class="form-group">
  5       <label for="title">Picture:</label>
  6       <input [ngModel]="user?.image" class="form-control" id="image" name="image" placeholder="Enter the url of the image" type="url" required>
  7       <br><br><button class="btn btn-lg btn-warning" type="submit">Change</button>
  8     </div>
  9   </fieldset>
10 </form>
```

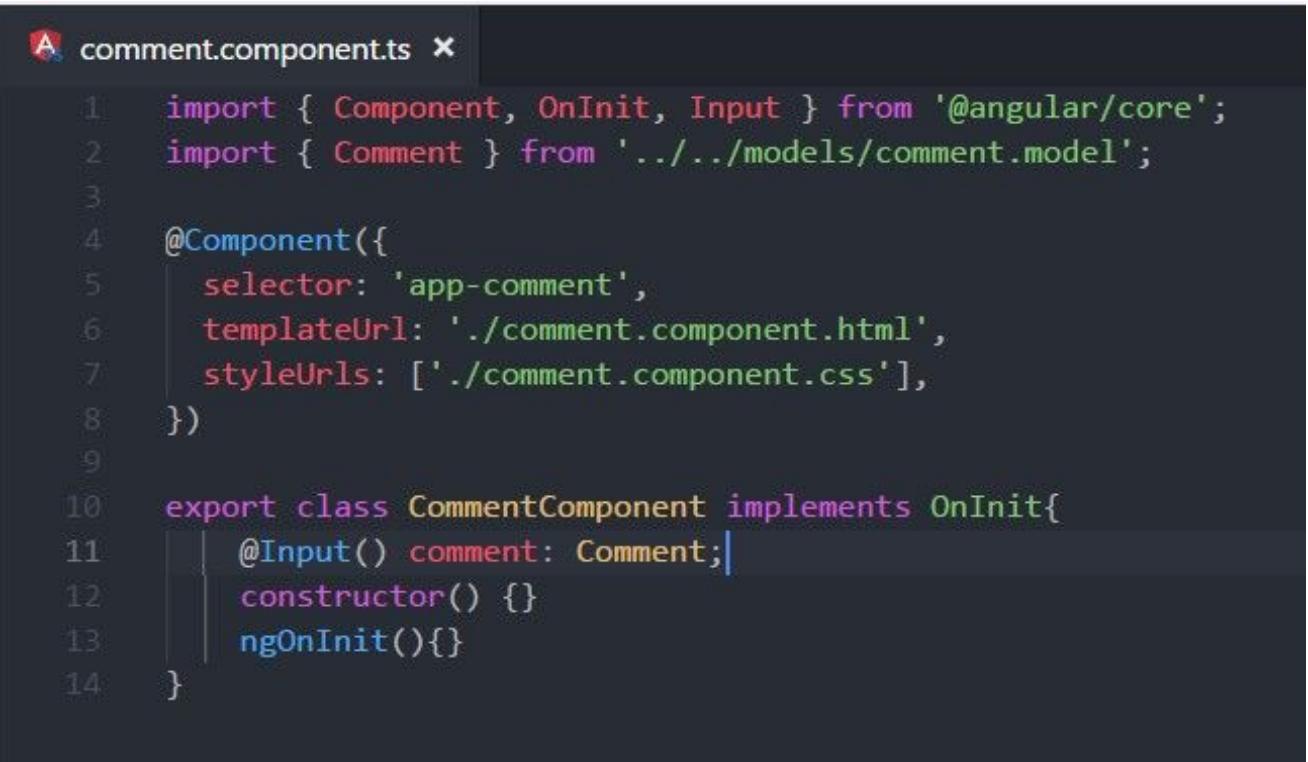
Slika 63. change-profile-picture.component.html

```
A change-profile-picture.component.ts x
1 import { Component, OnInit, ViewChild } from '@angular/core';
2 import { Router } from '@angular/router';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { User } from '../../../../../models/user.model';
5 import { FlashMessagesService } from 'angular2-flash-messages';
6 import { NgForm } from '@angular/forms';
7
8 @Component({
9   selector: 'app-change-profile-picture',
10  templateUrl: './change-profile-picture.component.html',
11  styleUrls: ['./change-profile-picture.component.css']
12})
13
14 export class ChangeProfilePictureComponent implements OnInit {
15  user: any = {};
16  @ViewChild('f') form: NgForm;
17
18  constructor(private router: Router, private authService: AuthService,
19    private flashMessage: FlashMessagesService) {}
20
21  ngOnInit () {
22    if (this.authService.isLoggedIn())
23    {
24      this.authService.getUser(localStorage.getItem('userId'))
25        .subscribe((user: User) => {
26          this.user = user;
27        })
28    );
29  }
30  else this.router.navigate(['/login']);
31
32  onSubmit () {
33    this.user.image = this.form.value.image;
34    this.authService.changeProfilePicture(this.user)
35      .subscribe((user: User) => {
36        this.user = user;
37        this.router.navigate(['/profile']);
38        this.flashMessage.show('Profile picture changed', { cssClass: 'alert-warning', timeout: 3000 });
39      })
40    );
41  }
42}
43}
```

Slika 64.change-profile-picture.component.ts

```
5 comment.component.html x
1 <div id="comment" class="card text-white mb-3 flex-container bg-warning">
2   <div class="card-header flex-item"><span class="badge badge-success">{{ comment.tag }}</span></div>
3   <div class="card-body flex-item">
4     <h4 class="card-title flex-item">{{ comment.content }}</h4>
5     <p class="card-text flex-item">User: {{ comment.username }}</p>
6     <p class="card-text flex-item">Grade: {{ comment.grade }}</p>
7     <p class="card-text flex-item">Published: {{ comment.publishDate | date }}</p>
8   </div>
9 </div>
```

Slika 65. comment.component.html



A screenshot of a code editor showing the file `comment.component.ts`. The code defines a component named `CommentComponent` that implements the `OnInit` interface. It has an input property `comment` of type `Comment`. The component selector is `'app-comment'`, and its template URL is `'./comment.component.html'`.

```
1 import { Component, OnInit, Input } from '@angular/core';
2 import { Comment } from '../../../../../models/comment.model';
3
4 @Component({
5   selector: 'app-comment',
6   templateUrl: './comment.component.html',
7   styleUrls: ['./comment.component.css'],
8 })
9
10 export class CommentComponent implements OnInit{
11   |   @Input() comment: Comment;
12   |   constructor(){}
13   |   ngOnInit(){}
14 }
```

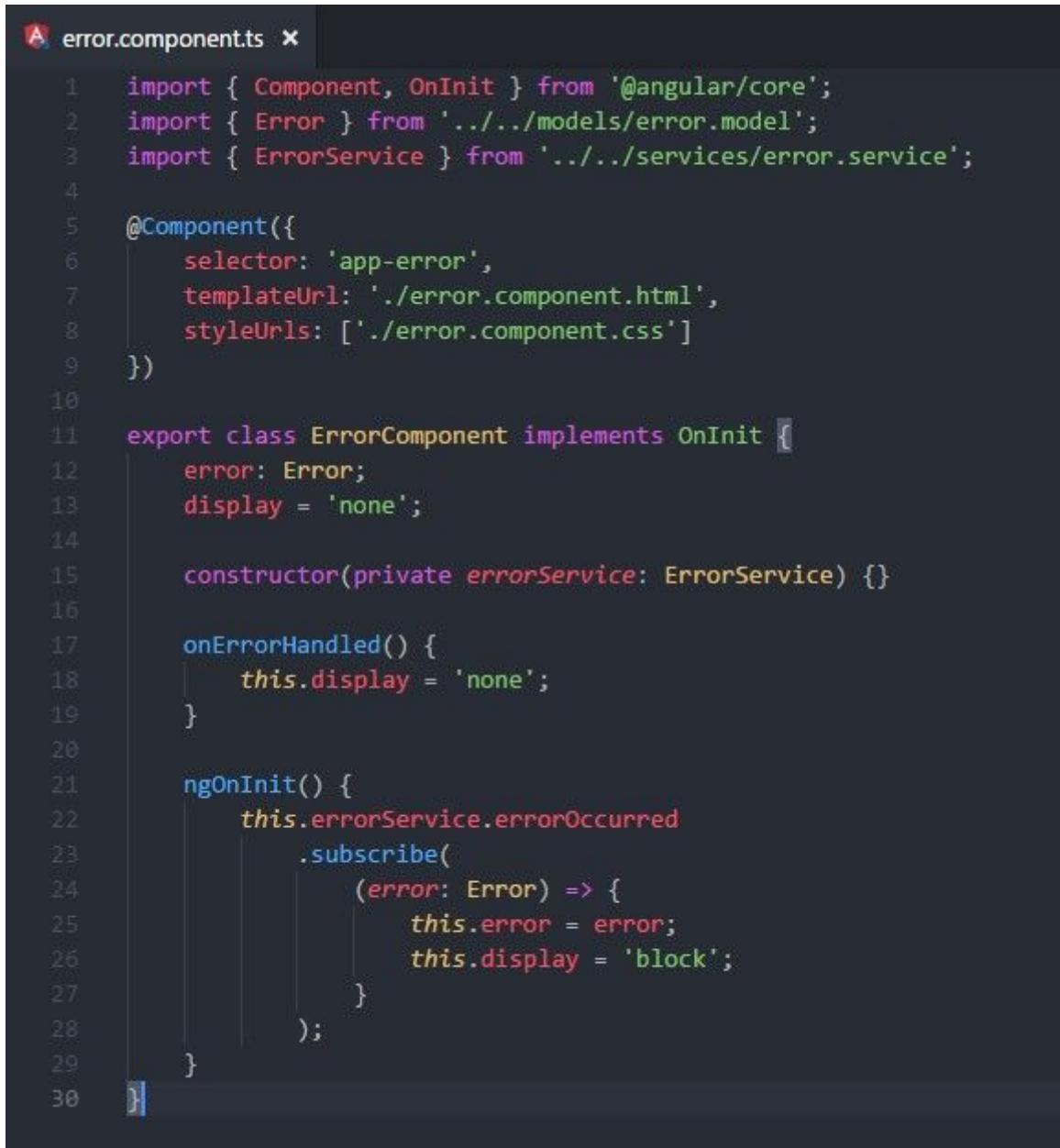
Slika 66. comment.component.ts



A screenshot of a code editor showing the file `error.component.html`. The code defines a modal dialog component. It consists of a backdrop, a modal dialog with a header containing a close button and title, a body with message text, and a footer with a close button.

```
1 <div class="backdrop" [ngStyle]="{'display': display}"></div>
2 <div class="modal" tabindex="-1" role="dialog" [ngStyle]="{'display': display}">
3   <div class="modal-dialog" role="document">
4     <div class="modal-content">
5       <div class="modal-header">
6         <button type="button" class="close" aria-label="Close" (click)="onErrorHandler()">&times;</button>
7         <h4 class="modal-title">{{ error?.title }}</h4>
8       </div>
9       <div class="modal-body">
10         <p>{{ error?.message }}</p>
11       </div>
12       <div class="modal-footer">
13         <button type="button" class="btn btn-default" (click)="onErrorHandler()">Close</button>
14       </div>
15     </div>
16   </div>
17 </div>
```

Slika 67. error.component.html

A screenshot of a code editor showing the file 'error.component.ts'. The code defines a component named 'ErrorComponent' that implements the 'OnInit' interface. It imports 'Component', 'OnInit', 'Error', and 'ErrorService'. The component has a selector 'app-error', a template URL 'error.component.html', and style URLs 'error.component.css'. It contains methods 'onErrorHandler()' and 'ngOnInit()'. The 'ngOnInit()' method uses RxJS's 'subscribe' operator to listen for errors from the 'errorService.errorOccurred' observable and update the component's state.

```
1 import { Component, OnInit } from '@angular/core';
2 import { Error } from '../../models/error.model';
3 import { ErrorService } from '../../services/error.service';
4
5 @Component({
6   selector: 'app-error',
7   templateUrl: './error.component.html',
8   styleUrls: ['./error.component.css']
9 })
10
11 export class ErrorComponent implements OnInit {
12   error: Error;
13   display = 'none';
14
15   constructor(private errorService: ErrorService) {}
16
17   onErrorHandled() {
18     this.display = 'none';
19   }
20
21   ngOnInit() {
22     this.errorService.errorOccurred
23       .subscribe(
24         (error: Error) => {
25           this.error = error;
26           this.display = 'block';
27         }
28       );
29   }
30 }
```

Slika 68. error.component.ts

```
header.component.html ×

1 <header>
2   <nav class="col-md-12 navbar navbar-expand-lg navbar-dark bg-dark nav-default">
3     <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
4       <span class="navbar-toggler-icon"></span></button>
5     <div class="collapse navbar-collapse" id="navbarNav">
6       <ul class="nav nav-tabs mr-auto">
7         <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}"><a class="nav-link" [routerLink]="/">Home</a></li>
8         <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}" *ngIf="!isLoggedIn()"><a class="nav-link" [routerLink]="/profile">Profile</a></li>
9         <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}"><a class="nav-link" [routerLink]="/blog">Blog</a></li>
10        <li *ngIf="isAdmin()" role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}"><a class="nav-link" [routerLink]="/users">Users</a></li>
11      </ul>
12      <ul class="nav nav-tabs">
13        <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}" *ngIf="!isLoggedIn()"><a class="nav-link" [routerLink]="/register">Register</a></li>
14        <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}" *ngIf="!isLoggedIn()"><a class="nav-link" [routerLink]="/login">Login</a></li>
15        <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}" *ngIf="!isLoggedIn()"><a class="nav-link" [routerLink]="/logout">Logout</a></li>
16        <li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]:"{exact:true}"><a class="nav-link" [routerLink]="/about">About</a></li>
17      </ul>
18    </div>
19  </nav>
20</header>
```

Slika 69. header.component.html

```
A header.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from '../../services/auth.service';
3 import { Book } from '../../models/book.model';
4 import { User } from '../../models/user.model';
5
6 @Component({
7   selector: 'app-header',
8   templateUrl: './header.component.html'
9 })
10
11 export class HeaderComponent implements OnInit {
12
13   constructor(private authService: AuthService) {}
14
15   user: any = {};
16   book: Book;
17
18   ngOnInit() {
19     if (this.authService.isLoggedIn()) {
20       this.authService.getUser(localStorage.getItem('userId'))
21         .subscribe((user: User) => {
22           this.user = user;
23         })
24     }
25   }
26
27   isLoggedIn() {
28     return this.authService.isLoggedIn();
29   }
30
31   isAdmin() {
32     if (this.authService.isLoggedIn())
33       return this.user.email === 'vukan.markovic97@gmail.com';
34     return false;
35   }
36
37 }
```

Slika 70. header.component.ts

```
home.component.html ×
1 <div class="jumbotron">
2   <div class="text-center">
3     <h1>Online Book Store</h1>
4     <p>Buy or sell your favorite book</p>
5   </div>
6 </div>
7 <div class="row">
8   <div class="col-md-12">
9     <div *ngFor="let book of books" (click)="bookDetails(book)">
10       <div id="book" *ngIf="!book.solded" class="card text-white mb-3 flex-container bg-info">
11         <div class="card-header flex-item">{{ book.title }}</div>
12         <div class="card-body flex-item">
13           <h4 class="card-title flex-item">{{ book.price | currency : "usd" : 3 }}</h4>
14           <p class="card-text flex-item"><img style="height: 200px; width: 40%; display: block;" src= "{{ book.image }}" alt="Book cover" title="{{ book.title }}"/></p>
15         </div>
16       </div>
17     </div>
18   </div>
19 </div>
20 </div>
```

Slika 71. home.component.html

```
home.component.ts ×
1 import { Component, OnInit } from '@angular/core';
2 import { BookService } from '../../../../../services/book.service';
3 import { Book } from '../../../../../models/book.model';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-home',
8   templateUrl: './home.component.html',
9   styleUrls: ['./home.component.css'],
10 })
11
12 export class HomeComponent implements OnInit {
13   public books: Book[] = [];
14
15   constructor(private bookService: BookService, private router: Router) {}
16
17   ngOnInit () {
18     this.bookService.getBooksHome()
19       .subscribe((books: Book[]) => {
20         this.books = books;
21       })
22     );
23   }
24
25   bookDetails(book: Book) {
26     this.router.navigate(['/book-details/', book.bookId]);
27   }
28 }
```

Slika 72. home.component.ts

```
5 login.component.html ✘
1  <div class="col-md-3 col-md-offset-2 content">
2    <form [formGroup]="form" (ngSubmit)="onSubmit()" class="form-signin">
3      <h2 class="form-signin-heading">Login</h2>
4      <div class="form-group">
5        <label for="email">Email</label>
6        <input name="email" type="email" id="email" class="form-control" formControlName="email" required autocomplete="email">
7      </div>
8      <div class="form-group">
9        <label for="password">Password</label>
10       <input type="password" id="password" class="form-control" formControlName="password" required
11         autocomplete="current-password">
12     </div>
13     <button class="btn btn-success btn-lg" type="submit" [disabled]="!form.valid">Submit</button>
14   </form>
15   <br>Not registered. Click <span (click)="register()">here</span> to join us!
16 </div>
```

Slika 73. login.component.html

```

A login.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup, FormControl, Validators } from '@angular/forms';
3 import { Router } from '@angular/router';
4 import { User } from '../models/user.model';
5 import { AuthService } from '../services/auth.service';
6 import { FlashMessagesService } from 'angular2-flash-messages';
7
8 @Component({
9   selector: 'app-login',
10  templateUrl: './login.component.html',
11  styleUrls: ['./login.component.css']
12 })
13
14 export class LoginComponent implements OnInit {
15   form: FormGroup;
16
17  constructor(private authService: AuthService, private router: Router, private flashMessage: FlashMessagesService) {}
18
19  ngOnInit() {
20    this.form = new FormGroup({
21      email: new FormControl(null, [
22        Validators.required,
23        Validators.pattern("[a-zA-Z0-9!#$%&*+/=?^_{}~-]+(?:\.[a-zA-Z0-9!#$%&*+/=?^_{}~-]+)*@(?:[a-zA-Z0-9](?:[a-zA-Z0-9]*[a-zA-Z0-9])?\\.)+[a-zA-Z0-9-]*[a-zA-Z0-9])?"))
24    },
25    password: new FormControl(null, Validators.required)
26  });
27 }
28
29
30  onSubmit() {
31    const user = new User(this.form.value.email, this.form.value.password);
32    this.authService.login(user)
33      .subscribe(
34        data => {
35          localStorage.setItem('token', data.token);
36          localStorage.setItem('userId', data.userId);
37          this.flashMessage.show('Logged in!', { cssClass: 'alert-success', timeout: 3000 });
38          this.router.navigateByUrl('/');
39        },
40        error => {
41          this.flashMessage.show('Invalid email or/and password!', { cssClass: 'alert-danger', timeout: 3000 });
42          console.error(error)
43        }
44      );
45    this.form.reset();
46  }
47
48  register() {
49    this.router.navigate(['/register']);
50  }
51 }

```

Slika 74. login.component.ts

```

5 logout.component.html x
1 <div class="col-md-8 col-md-offset-2 content">
2   <button class="btn btn-lg btn-danger" (click)="onLogout()">Logout</button>
3 </div>

```

Slika 75. logout.component.html

```
A logout.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { Router } from '@angular/router';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { FlashMessagesService } from 'angular2-flash-messages';
5
6 @Component({
7   selector: 'app-logout',
8   templateUrl: './logout.component.html'
9 })
10 export class LogoutComponent implements OnInit {
11
12   constructor(private authService: AuthService, private router: Router, private flashMessage: FlashMessagesService) { }
13
14   ngOnInit() {
15     if (!this.authService.isLoggedIn())
16       this.router.navigate(['/login']);
17   }
18
19   onLogout() {
20     this.authService.logout();
21     this.flashMessage.show('Logged out!', { cssClass: 'alert-danger', timeout: 3000 });
22     this.router.navigate(['']);
23   }
24 }
```

Slika 76. logout.component.ts

```
5 page-not-found.component.html x
1 <div class="content">
2   <div align="center">
3     
4   </div>
5 </div>
```

Slika.77. page-not-found.component.html

```
A page-not-found.component.ts x
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-page-not-found',
5   templateUrl: './page-not-found.component.html'
6 })
7
8 export class PageNotFoundComponent implements OnInit {
9   constructor() { }
10  ngOnInit() {}
11 }
```

Slika 78. page-not-found.component.ts

```
5 post.component.html x
1 <div id="post" class="card text-white mb-3 flex-container bg-success">
2   <div class="card-header flex-item">{{ post.title }}</div>
3   <div class="card-body flex-item">
4     <h4 class="card-title flex-item">By: {{ post.username }}</h4>
5     <p class="card-text flex-item"></p>
6   </div>
7 </div>
```

Slika 79. post.component.html

```
6 post.component.ts x
1 import { Component, OnInit, Input } from '@angular/core';
2 import { Post } from '../../../../../models/post.model';
3
4 @Component({
5   selector: 'app-post',
6   templateUrl: './post.component.html',
7   styleUrls: ['./post.component.css'],
8 })
9
10 export class PostComponent implements OnInit{
11   @Input() post: Post;
12   constructor() {}
13   ngOnInit(){}
14 }
```

Slika 80. post.component.ts



The screenshot shows a code editor window with the title "post-details.component.html". The code is written in HTML and includes some CSS classes and Angular-specific directives like *ngIf.

```
1 <article>
2   <div class="card mb-3 bg-danger">
3     <h3 class="card-header"><span id="text">Title:</span> {{ post.title | uppercase }}</h3>
4     <div class="card-body">
5       <h5 class="card-title"><span id="text">By:</span> {{ post.username }}</h5>
6       <h6 class="card-subtitle text-muted"></h6>
7     </div>
8     <div class="card-body">
9       <p class="card-text"></p>
10    </div>
11    <div class="card-body">
12      {{ post.content }}
13    </div>
14    <div class="card-footer text-muted">
15      <button *ngIf="isAdmin()" class="card-link btn btn-lg btn-success" (click)="onDelete()>Delete</button>
16      <button *ngIf="isAdmin()" class="card-link btn btn-lg btn-success" (click)="onUpdate()>Update</button>
17    </div>
18  </div>
19 </article>
```

Slika 81. post-details.component.html

```
post-details.component.ts X
1 import { Component, OnInit } from '@angular/core';
2 import { Post } from '../../../../../models/post.model';
3 import { PostService } from '../../../../../services/post.service';
4 import { ActivatedRoute, Router } from '@angular/router';
5 import { AuthService } from '../../../../../services/auth.service';
6 import { FlashMessagesService } from 'angular2-flash-messages';
7 import { User } from '../../../../../models/user.model';
8
9 @Component({
10   selector: 'app-post-details',
11   templateUrl: './post-details.component.html',
12   styleUrls: ['./post-details.component.css'],
13 })
14
15 export class PostDetailsComponent implements OnInit {
16   post: any = {};
17   user: any = {};
18
19   constructor(private postService: PostService, private route: ActivatedRoute,
20             private authService: AuthService, private flashMessage: FlashMessagesService,
21             private router: Router) {}
22
23   ngOnInit() {
24     this.postService.getPost(this.route.snapshot.params['id'])
25       .subscribe((post: Post) => {
26         this.post = post;
27       });
28
29     if(this.authService.isLoggedIn()) {
30       this.authService.getUser(localStorage.getItem('userId'))
31         .subscribe((user: User) => {
32           this.user = user;
33         });
34     }
35   }
36
37   isAdmin() {
38     if(this.authService.isLoggedIn())
39       return this.user.email === 'vukan.markovic97@gmail.com';
40     return false;
41   }
42
43   onDelete() {
44     this.postService.deletePost(this.post)
45       .subscribe(
46         result => {
47           console.log(result);
48           this.router.navigate(['/blog']);
49           this.flashMessage.show('Post is deleted!', { cssClass: 'alert-danger', timeout: 3000 });
50         },
51         error => console.error(error));
52   }
53
54   onUpdate() {
55     this.router.navigate(['/update-post', this.post.postId]);
56   }
57 }
```

Slika 82. post-details.component.ts

```
profile.component.html X
1 <div class="row">
2   <div class="col-sm-12">
3     <div class="jumbotron">
4       
5       <br><br><button id="changeProfilePicture" class="btn btn-info btn-sm btn-block" (click)=changeProfilePicture()>Change profile picture</button>
6       <br><br><div><span class="text-info">{{ user.firstName }}</span></div>
7       <div><span class="text-danger">{{ user.lastName }}</span></div>
8       <div><span class="text-success">{{ user.email }}</span></div>
9     </div>
10    <br><button id="btnAddBook" class="btn btn-primary btn-lg btn-block" (click)=addBook()>Add a book for sale</button>
11    <button id="btnAddBook" class="btn btn-default btn-lg btn-block" (click)=buyBook()>Buy book</button>
12  </div>
13 </div>
14 <div class="row">
15   <div class="col-sm-12">
16     <br><h2>Books:</h2>
17     <app-book [book]="book" *ngFor="let book of books"></app-book>
18   </div>
19 </div>
```

Slika 83. profil.component.html

```

profile.component.ts ✘

1 import { Component, OnInit } from '@angular/core';
2 import { Book } from '../../../../../models/book.model';
3 import { BookService } from '../../../../../services/book.service';
4 import { Router } from '@angular/router';
5 import { AuthService } from '../../../../../services/auth.service';
6 import { User } from '../../../../../models/user.model';
7
8 @Component({
9   selector: 'app-profile',
10  templateUrl: './profile.component.html',
11  styleUrls: ['./profile.component.css']
12})
13
14 export class ProfileComponent implements OnInit {
15   books: Book[];
16   user: any = {};
17
18   constructor(private bookService: BookService, private router: Router,
19               private authService: AuthService) {}
20
21   ngOnInit () {
22     if (this.authService.isLoggedIn())
23     {
24       this.authService.getUser(localStorage.getItem('userId'))
25         .subscribe((user: User) => {
26           this.user = user;
27         })
28     };
29
30     this.bookService.getBooks()
31       .subscribe((books: Book[]) => {
32         this.books = books;
33       })
34     );
35   }
36   else this.router.navigate(['/login']);
37 }
38
39 addBook () {
40   this.router.navigate(['/add-book']);
41 }
42
43 buyBook () {
44   this.router.navigate(['']);
45 }
46
47 changeProfilePicture () {
48   this.router.navigate(['/change-profile-picture']);
49 }
50 }

```

Slika 84. profil.component.ts

```

publish.component.html ✘

1 <form (ngSubmit)="onSubmit()" #f="ngForm">
2   <fieldset>
3     <legend>Publish a new post</legend>
4     <div class="form-group">
5       <label for="title">Title of the post:</label>
6       <input type="text" id="title" name="title" class="form-control" [ngModel]="post?.title" rows="14" required>
7     </div>
8     <div class="form-group">
9       <label for="content">Shares with us something interesting</label>
10      <textarea id="content" name="content" class="form-control" [ngModel]="post?.content" rows="14" required></textarea>
11    </div>
12    <p id="warning" class="form-text text-muted">We want just to notify you that this is forum about literature and books and all non-topic posts will be deleted!</p>
13    <button class="btn btn-lg btn-success" type="submit" [disabled]="!form.valid">Post</button>
14    <button type="button" class="btn btn-danger btn-lg" (click)="onClear()">Clear</button>
15  </fieldset>
16 </form>

```

Slika 85. publish.component.html

```
publish.component.ts ✘

1 import { Component, OnInit, ViewChild } from '@angular/core';
2 import { NgForm } from '@angular/forms';
3 import { PostService } from '../../../../../services/post.service';
4 import { Post } from '../../../../../models/post.model';
5 import { AuthService } from '../../../../../services/auth.service';
6 import { Router } from '@angular/router';
7 import { User } from '../../../../../models/user.model';
8
9 @Component({
10   selector: 'app-publish',
11   templateUrl: './publish.component.html'
12 })
13
14 export class PublishComponent implements OnInit {
15   @ViewChild('f') form: NgForm;
16   post: Post;
17   user: any = {};
18
19   constructor(private postService: PostService, private authService: AuthService, private router: Router) {}
20
21   ngOnInit() {
22     if (!this.authService.isLoggedIn())
23       this.router.navigate(['/login']);
24     this.authService.getUser(localStorage.getItem('userId'))
25       .subscribe((user: User) => {
26         this.user = user;
27       })
28     );
29   }
30
31   onSubmit() {
32     const post = new Post(this.form.value.title, this.form.value.content, this.user.firstName);
33     this.postService.addPost(post)
34       .subscribe(data => console.log(data), error => console.error(error));
35     this.form.resetForm();
36     this.router.navigate(['/blog']);
37   }
38
39   onClear() {
40     this.post = null;
41     this.form.resetForm();
42   }
43 }
```

Slika 86. publish.component.ts

```
register.component.html ✘
1 <div class="col-md-4 col-md-offset-2 content">
2   <form [formGroup]="form" (ngSubmit)="onSubmit()">
3     <h2>Register</h2>
4     <div class="form-group">
5       <label for="firstName">First Name</label>
6       <input type="text" id="firstName" class="form-control" formControlName="firstName" autocomplete="name">
7     </div>
8     <div class="form-group">
9       <label for="lastName">Last Name</label>
10      <input type="text" id="lastName" class="form-control" formControlName="lastName" autocomplete="family-name">
11    </div>
12    <div class="form-group">
13      <label for="email">Email</label>
14      <input type="email" id="email" class="form-control" formControlName="email" autocomplete="email">
15    </div>
16    <div class="form-group">
17      <label for="password">Password</label>
18      <input type="password" id="password" class="form-control" formControlName="password" autocomplete="current-password">
19    </div>
20    <div class="form-group">
21      <label for="country">Country</label>
22      <input type="text" id="country" class="form-control" formControlName="country" autocomplete="country">
23    </div>
24    <div class="form-group">
25      <label for="city">City</label>
26      <input type="text" id="city" class="form-control" formControlName="city">
27    </div>
28    <div class="form-group">
29      <label for="postalCode">Postal code</label>
30      <input type="text" id="postalCode" class="form-control" formControlName="postalCode" autocomplete="postal-code">
31    </div>
32    <div class="form-group">
33      <label for="address">Address</label>
34      <input type="text" id="address" class="form-control" formControlName="address" autocomplete="address-level2">
35    </div>
36    <div class="form-group">
37      <label for="phoneNumber">Phone number</label>
38      <input type="tel" id="phoneNumber" class="form-control" formControlName="phoneNumber" autocomplete="tel">
39    </div>
40    <button class="btn btn-lg btn-info" type="submit" [disabled]="!form.valid">Submit</button>
41    <br><br>Already have account. <span (click)="login()">Log in.</span>
42  </form>
43</div>
```

Slika 87. register.component.html

```

  * register.component.ts
  import { Component, OnInit } from '@angular/core';
  import { FormGroup, FormControl, Validators } from '@angular/forms';
  import { Router } from '@angular/router';
  import { AuthService } from './services/auth.service';
  import { User } from './models/user.model';
  import { FlashMessageService } from 'angular2-flash-messages';

  @Component({
    selector: 'app-register',
    templateUrl: './register.component.html',
    styleUrls: ['./register.component.css']
  })

  export class RegisterComponent implements OnInit {
    form: FormGroup;
    re: any;

    constructor(private authService: AuthService, private router: Router, private flashMessage: FlashMessageService) {}

    ngOnInit() {
      this.re = /^(?(([^0-9])|([0-9]))+([0-9]))+([0-9])+(?(([^0-9][1,3])|([0-9][1,3])|([0-9][1,3])|([0-9][1,3])))|(([a-zA-Z]-[0-9])+,[a-zA-Z][2,3]))$/;
      this.form = new FormGroup({
        firstName: new FormControl(null, [
          Validators.required,
          Validators.pattern(`^${this.re}(?=[2,3]$)(?![_.])${1}.*[_.]{2})${a-zA-Z0-9}_+${2}[_.]{2}$`),
        ]),
        lastName: new FormControl(null, [
          Validators.required,
          Validators.pattern(`^${this.re}(?=[4,20]$)(?![_.])${1}.*[_.]{2})${a-zA-Z0-9}_+${2}[_.]{2}$`),
        ]),
        email: new FormControl(null, [
          Validators.required,
          Validators.pattern(this.re)
        ]),
        password: new FormControl(null, Validators.required),
        country: new FormControl(null, Validators.required),
        city: new FormControl(null, Validators.required),
        postalCode: new FormControl(null, Validators.required),
        address: new FormControl(null, Validators.required),
        phoneNuber: new FormControl(null, Validators.required)
      });
    }

    onSubmit() {
      const user = new User(
        this.form.value.email,
        this.form.value.password,
        this.form.value.firstName,
        this.form.value.lastName,
        this.form.value.country,
        this.form.value.city,
        this.form.value.postalCode,
        this.form.value.address,
        this.form.value.phoneNuber
      );

      this.authService.register(user)
        .subscribe(data => {
          console.log(data);
          this.flashMessage.show('Registered!', { cssClass: 'alert-success', timeout: 3000 });
          this.router.navigate(['/login']);
        },
        error => console.error(error));
    }

    this.form.reset();
  }

  login () {
    this.router.navigate(['/login']);
  }
}

```

Slika 88. register.component.ts

```
5 show-info.component.html x
1 <article>
2   <div class="card mb-3 bg-danger">
3     <h3 class="card-header"><span id="text">User: </span> {{ user.firstName }} {{user.lastName}}</h3>
4     <div class="card-body">
5       <h5 class="card-title"></h5>
6       <h6 class="card-subtitle text-muted"></h6>
7     </div>
8     <div class="card-body">
9       <p class="card-text">
10        
11      </p>
12    </div>
13    <div class="card-body">
14      <div class="card-footer text-muted">
15        {{ user.country }}&nbsp;&nbsp;
16        {{ user.city }}&nbsp;&nbsp;
17        {{ user.postalCode }}&nbsp;&nbsp;
18        {{ user.address }}&nbsp;&nbsp;
19        {{ user.phoneNumber }}
20      </div>
21    </div>
22  </article>
```

Slika 89. show-info.component.html

```
6 show-info.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from '../../../../../services/auth.service';
3 import { User } from '../../../../../models/user.model';
4 import { ActivatedRoute } from '@angular/router';
5
6 @Component({
7   selector: 'app-show-info',
8   templateUrl: './show-info.component.html'
9 })
10
11 export class ShowInfoComponent implements OnInit {
12   public user: any = {};
13
14   constructor(private authService: AuthService, private route: ActivatedRoute) {}
15
16   ngOnInit () {
17     if(this.authService.isLoggedIn()) {
18       this.authService.getUser(this.route.snapshot.params['id'])
19         .subscribe((user: User) => {
20           this.user = user;
21         })
22     }
23   }
24 }
25 ]|
```

Slika 90. show-info.component.ts

```

    id="updateForm" (ngSubmit)="onSubmit()" #f="ngForm">

<legend>Update a book</legend>
<div class="form-group">
  <label for="title">Title:</label>
  <input [ngModel]="book?.title" class="form-control" id="title" name="title" placeholder="Enter the title of the book" type="text" required>
</div>
<div class="form-group">
  <label for="price">Price:</label>
  <input [ngModel]="book?.price" class="form-control" id="price" name="price" placeholder="Enter the price of the book" type="number" required>
</div>
<div class="form-group">
  <label for="genres">Genre:</label>
  <select [ngModel]="book?.genre" class="form-control" name="genre" id="genres" required>
    <option>Arts &and; Photography</option>
    <option>Biographies &and; Memoirs</option>
    <option>Business &and; Investing</option>
    <option>Children's Books</option>
    <option>History</option>
    <option>Literature &and; Suspense</option>
    <option>Romance</option>
    <option>Sci-Fi &and; Fantasy</option>
    <option>Teens &and; Young Adult</option>
  </select>
</div>
<div class="form-group">
  <label for="author">Author:</label>
  <input [ngModel]="book?.author" name="author" class="form-control" id="author" placeholder="Enter the author of the book" type="text" required>
</div>
<div class="form-group">
  <label for="about">Write a couple of words about the book if you like:</label>
  <textarea [ngModel]="book?.about" name="about" class="form-control" id="about" rows="3" placeholder="Your comment about book"></textarea>
</div>
<div class="form-group">
  <label for="image">Image:</label>
  <input [ngModel]="book?.image" name="image" class="form-control" id="image" placeholder="Enter the image url of the book cover" type="text">
</div>
<button class="btn btn-lg btn-warning" type="submit">Update a book</button>
</fieldset>

```

Slika 91. update-book.component.html

```
50  onSubmit() {
51      this.book.title = this.form.value.title;
52      this.book.author = this.form.value.author;
53      this.book.price = this.form.value.price;
54      this.book.genre = this.form.value.genre;
55      this.book.about = this.form.value.about;
56
57      if (this.user.email == "vukan.markovic97@gmail.com") {
58          this.bookService.updateBookAdmin(this.book)
59              .subscribe(data => console.log(data), error => console.error(error));
60
61          this.book = null;
62          this.form.resetForm();
63          this.flashMessage.show('Book is updated!', { cssClass: 'alert-warning', timeout: 3000 });
64          this.router.navigate(['/']);
65
66      } else {
67
68          this.bookService.updateBook(this.book)
69              .subscribe(data => console.log(data), error => console.error(error));
70          this.book = null;
71
72          this.form.resetForm();
73          this.flashMessage.show('Book is updated!', { cssClass: 'alert-warning', timeout: 3000 });
74          this.router.navigate(['/profile']);
75      }
76  }
77
78  onClear(form: NgForm) {
79      this.book = null;
80      form.resetForm();
81  }
82}
```

Slika 92. update-book.component.ts

```
1  <form id="updateForm" (ngSubmit)="onSubmit()" #f="ngForm">
2      <fieldset>
3          <legend>Update a post</legend>
4          <div class="form-group">
5              <label for="title">Title:</label>
6              <input [ngModel]="post?.title" id="title" name="title" class="form-control" placeholder="Enter the title of the post"
7                  type="text" required>
8          </div>
9          <div class="form-group">
10             <label for="content">Content:</label>
11             <textarea id="content" name="content" class="form-control" [ngModel]="post?.content" rows="14"></textarea>
12         </div>
13         <button class="btn btn-lg btn-warning" type="submit">Update a post</button>
14     </fieldset>
15 </form>
```

Slika 93. update-post.component.html

```

1  update-post.component.ts
2
3  import { Component, OnInit, ViewChild } from '@angular/core';
4  import { NgForm } from '@angular/forms';
5  import { PostService } from '../../../../../services/post.service';
6  import { Post } from '../../../../../models/post.model';
7  import { Router, ActivatedRoute } from '@angular/router';
8  import { AuthService } from '../../../../../services/auth.service';
9  import { FlashMessagesService } from 'angular2-flash-messages';
10
11 @Component({
12   selector: 'app-update-post',
13   templateUrl: './update-post.component.html',
14   styleUrls: ['./update-post.component.css']
15 })
16
17 export class UpdatePostComponent implements OnInit {
18   post: any = {};
19   @ViewChild('f') form: NgForm;
20
21   constructor(private postService: PostService, private router: Router, private authService: AuthService, private route: ActivatedRoute, private flashMessage: FlashMessagesService) {}
22
23   ngOnInit() {
24     if (this.authService.isLoggedIn()) {
25       this.postService.getPost(this.route.snapshot.params['id'])
26         .subscribe((post: Post) => {
27           this.post = post;
28           this.form.setValue({
29             title: post.title,
30             content: post.content
31           });
32         });
33     } else this.router.navigate(['/login']);
34   }
35
36   onSubmit() {
37     this.post.title = this.form.value.title;
38     this.post.content = this.form.value.content;
39
40     this.postService.updatePost(this.post)
41       .subscribe(data => console.log(data), error => console.error(error));
42
43     this.post = null;
44     this.form.resetForm();
45     this.flashMessage.show('Post is updated!', { cssClass: 'alert-warning', timeout: 3000 });
46     this.router.navigate(['/blog']);
47   }
48
49   onClear(form: NgForm) {
50     this.post = null;
51     form.resetForm();
52   }
53 }

```

Slika 94. update-post.component.ts

```

1  users.component.html
2
3  <div class="row">
4    <div class="col-md-12">
5      <div *ngFor="let user of users">
6        <div *ngIf="!isAdmin(user)" id="user" class="card text-white mb-3 flex-container bg-info">
7          <div class="card-header flex-item">{{ user.firstName }} {{ user.lastName }}</div>
8          <div class="card-body flex-item">
9            <h4 class="card-title flex-item">{{ user.email }}</h4>
10           <p class="card-text flex-item"><img style="height: 160px; width: 40%; display: block;" src= "{{ user.image }}" alt="Book cover" title="{{ user.firstName }}"/></p>
11         </div>
12       </div>
13     </div>
14   </div>

```

Slika 95. users.component.html

```
A users.component.ts x

1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from '../../services/auth.service';
3 import { User } from '../../models/user.model';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-users',
8   templateUrl: './users.component.html',
9   styleUrls: ['./users.component.css'],
10 })
11
12 export class UsersComponent implements OnInit {
13   public users: User[] = [];
14
15   constructor(private authService: AuthService, private router: Router) {}
16
17   ngOnInit () {
18     if (!this.authService.isLoggedIn()) {
19       this.router.navigate(['/login']);
20     }
21     else {
22       this.authService.getUsers()
23         .subscribe((users: User[]) => {
24           this.users = users;
25         })
26     }
27   }
28
29   isAdmin (user: User) {
30     return user.email === "vukan.markovic97@gmail.com";
31   }
32 }
33 }
```

Slika 96. users.component.ts

4. Ostalo

4.1 Linkovi

Tehnologije:

<https://angular.io/>

<https://expressjs.com/>

<https://nodejs.org/en/>

<http://mongoosejs.com/>

Tema: <https://bootswatch.com/sketchy/>

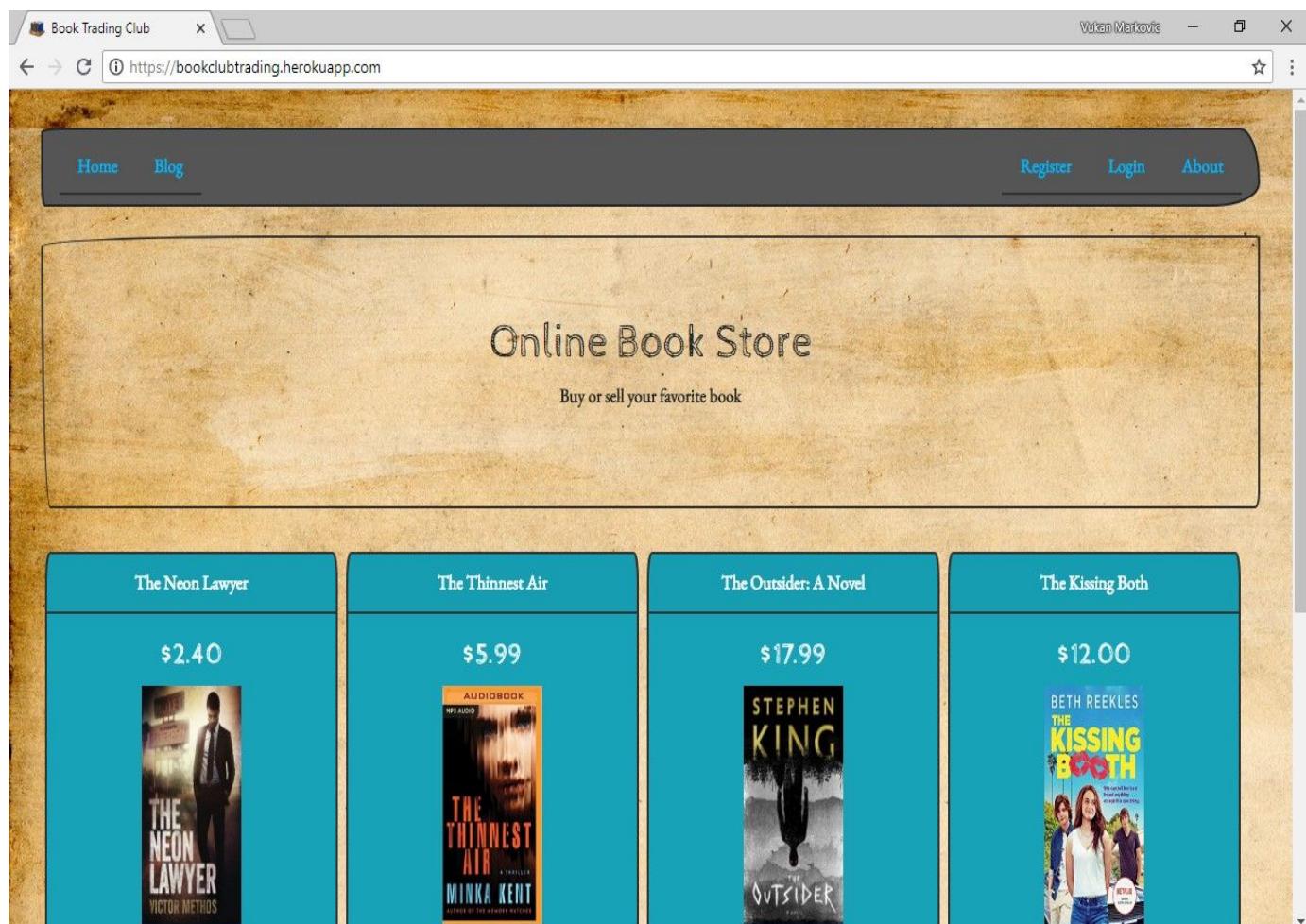
Hosting servis: <https://www.heroku.com/>

Sajt: <https://bookclubtrading.herokuapp.com/>.

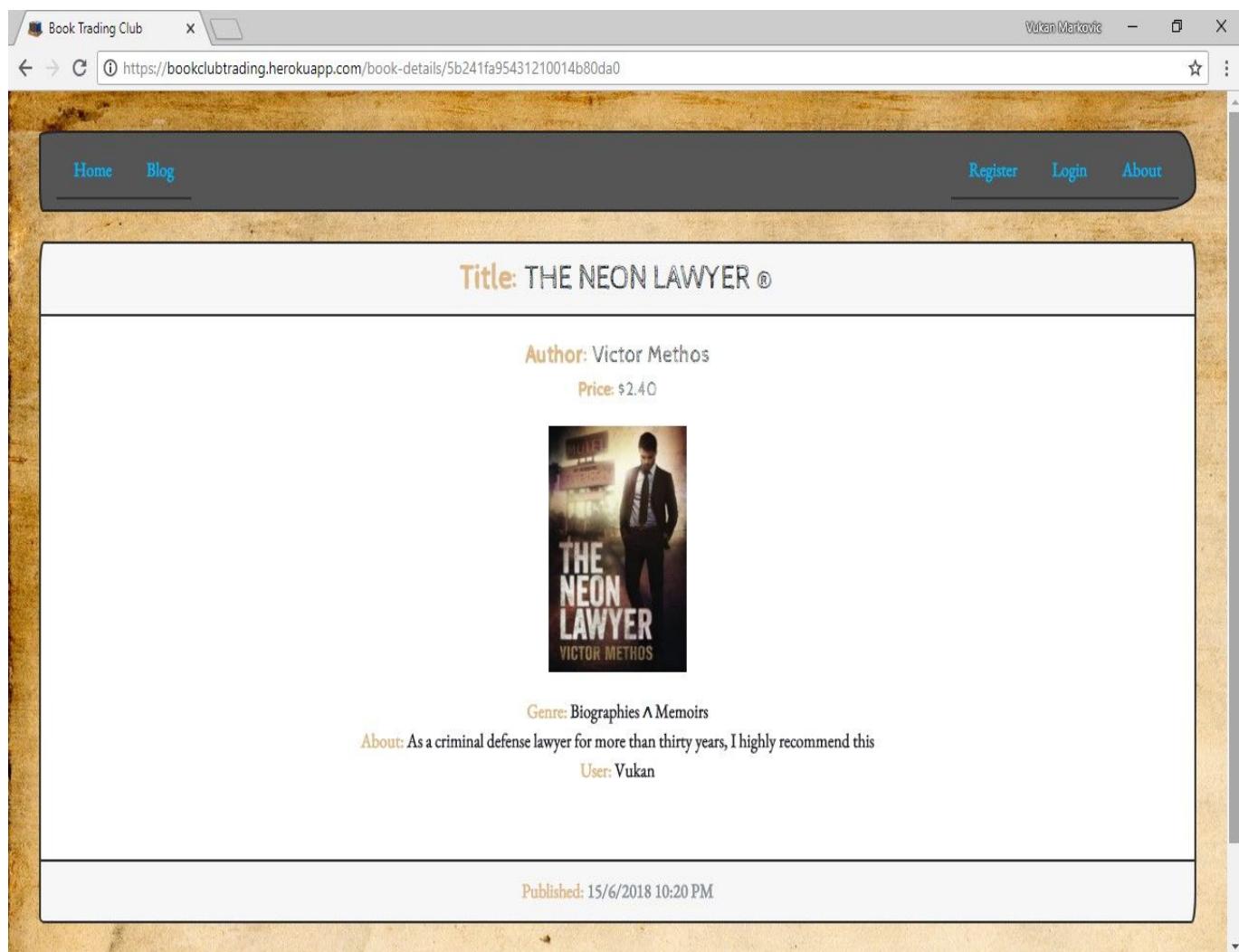
Baza podataka: <https://mlab.com/>

Repozitorijum: <https://github.com/Vukan-Markovic/Book-trading-club>

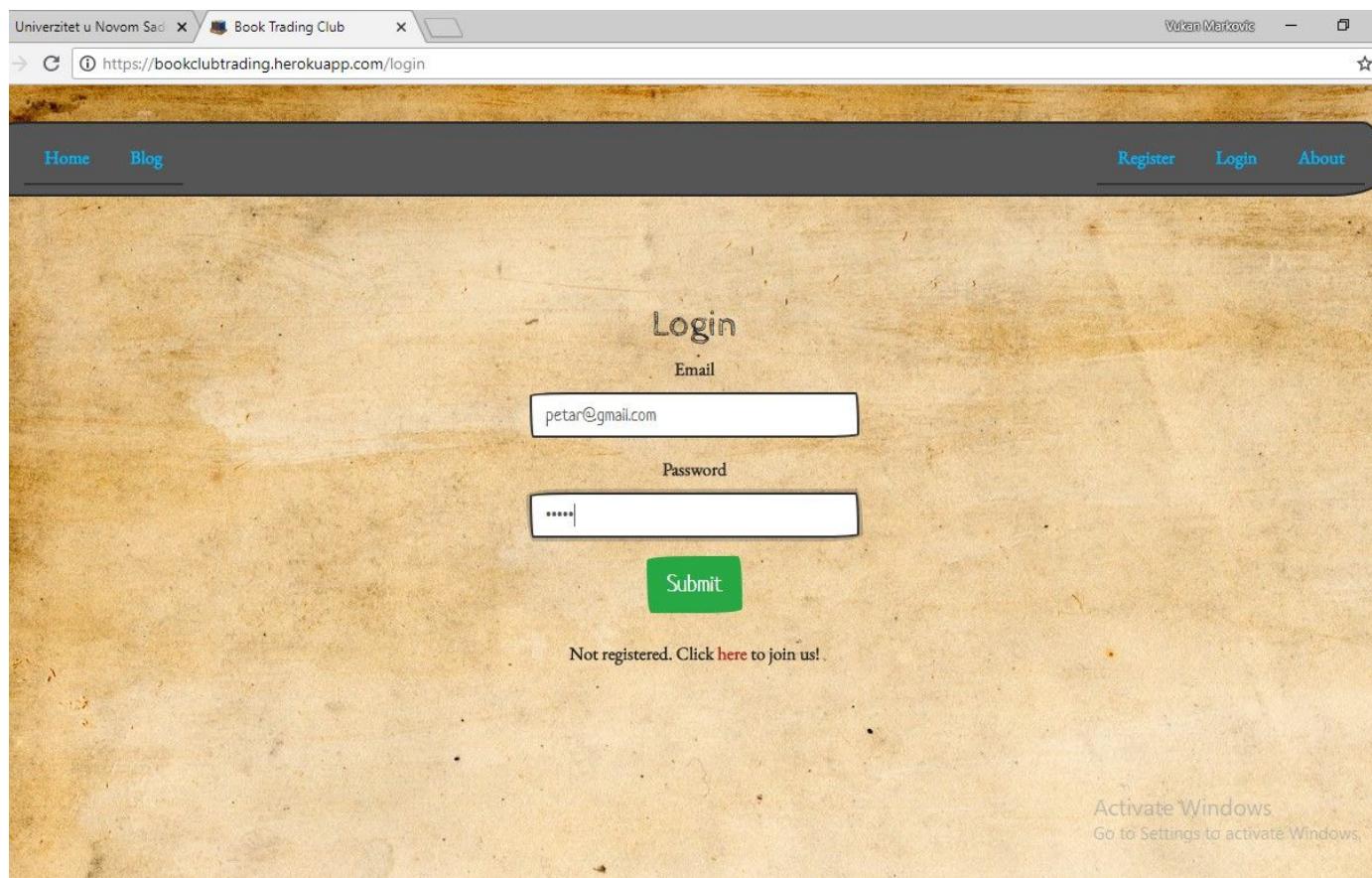
4.2 Sajt



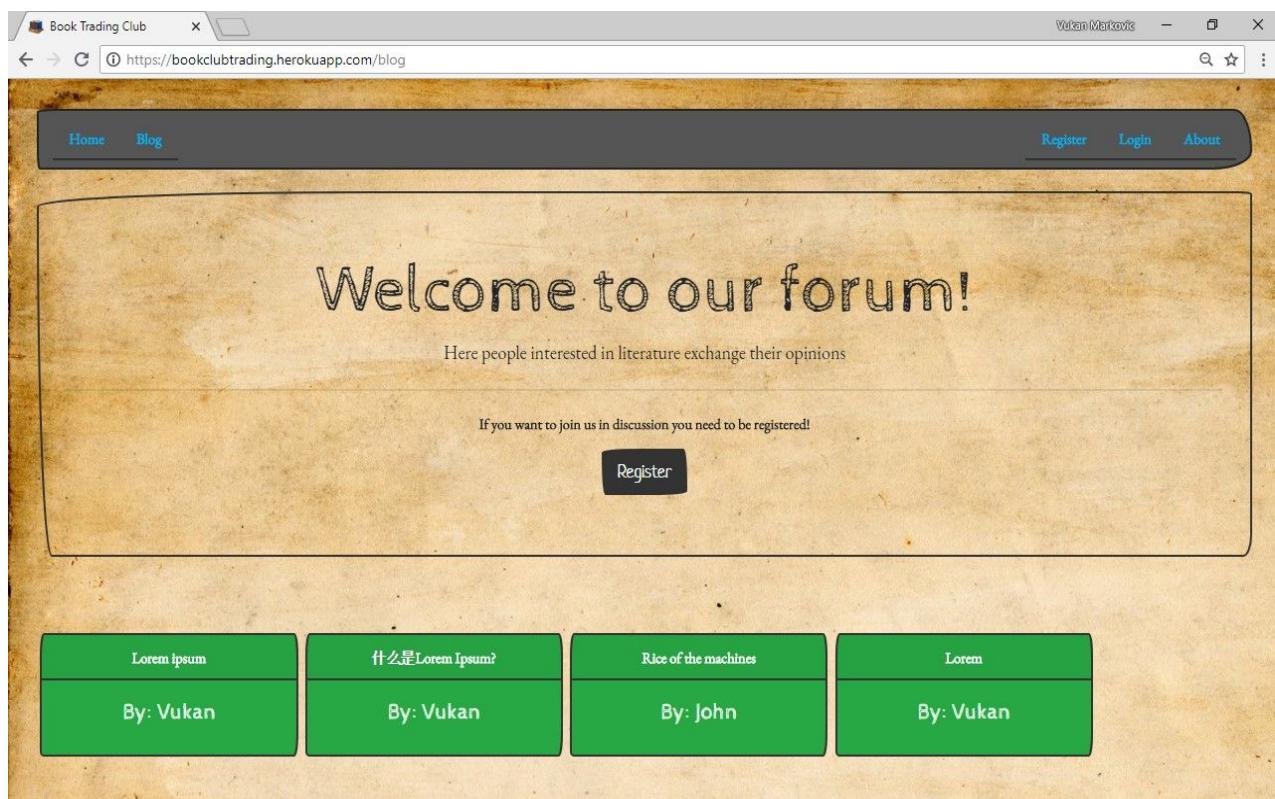
Slika 97. Naslovna strana sajta iz aspekta neregistrovanog korisnika



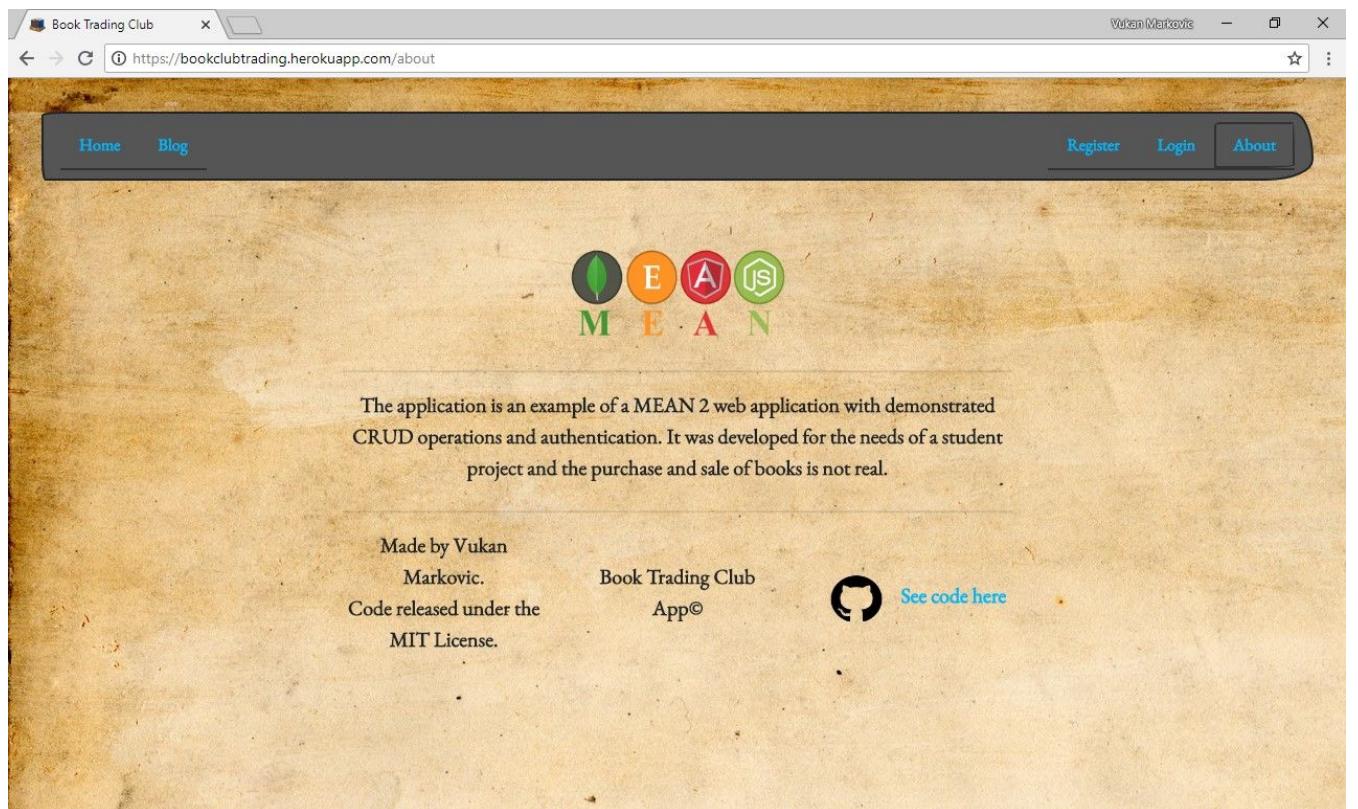
Slika 98. Neregistrovan korisnik može samo da pregleda dostupne knjige bez ostalih



Slika 99. Login strana vidljiva samo ako korisnik nije već ulogovan



Slika 100. Blog, neregistrovan korisnik može samo da pregleda objave



Slika 101. About stranica sa kratkim opisom sajta

A screenshot of a web browser window titled "Univerzitet u Novom Sadu" and "Book Trading Club". The URL in the address bar is "https://bookclubtrading.herokuapp.com/register". The page has a light brown, aged paper background. It is a registration form with fields for "First Name" (Petar), "Last Name" (Petrovic), "Email" (petar@gmail.com), "Password" (*****), "Country" (Serbia), "City" (Sombor), "Postal code" (25000), and "Address" (empty). There is also a watermark at the bottom right that says "Activate Windows Go to Settings to activate Windows".

A screenshot of a registration form on a brown paper background. The form consists of two input fields: 'Address' containing 'Futoska' and 'Phone number' containing '0652242442'. Below the inputs is a blue 'Submit' button. At the bottom of the form, there is a link 'Already have account. Log in.'

Address.
Futoska

Phone number
0652242442

Submit

Already have account. [Log in.](#)

Slika 101. Register stranica vidljiva samo korisnicima koji nisu registrovani (ulogovani)

A screenshot of a login page for 'Book Trading Club' on a brown paper background. The page features a dark header bar with 'Home' (highlighted), 'Blog', 'Register', 'Login', and 'About' links. The main content area is titled 'Login'. It contains two input fields: 'Email' with the value 'vukan.markovic97@gmail.com' and 'Password' with a masked value. A green 'Submit' button is located below the password field. At the bottom of the page, there is a link 'Not registered. Click [here](#) to join us!'

Book Trading Club

Vukan Markovic

Home Blog Register Login About

Login

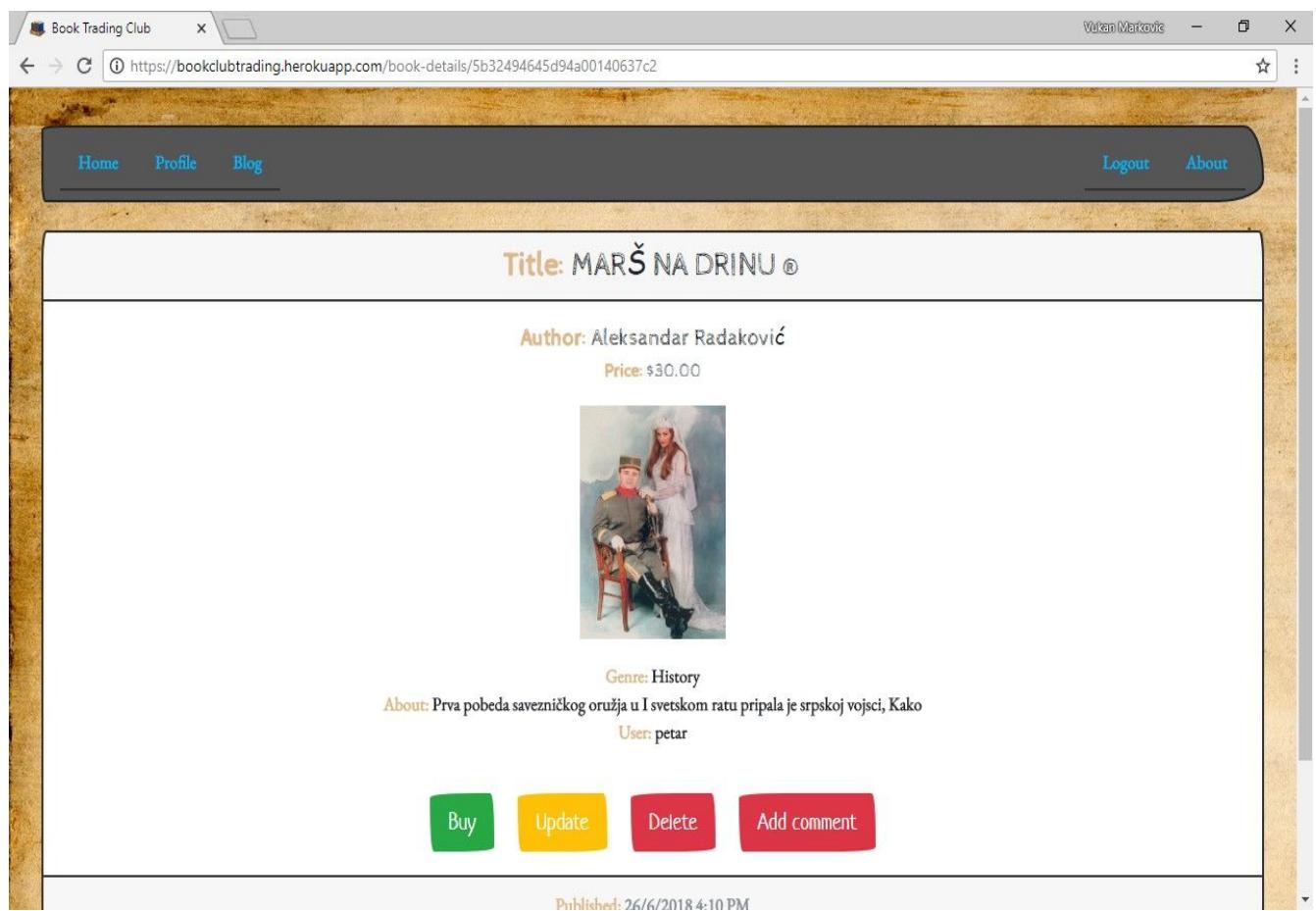
Email
vukan.markovic97@gmail.com

Password

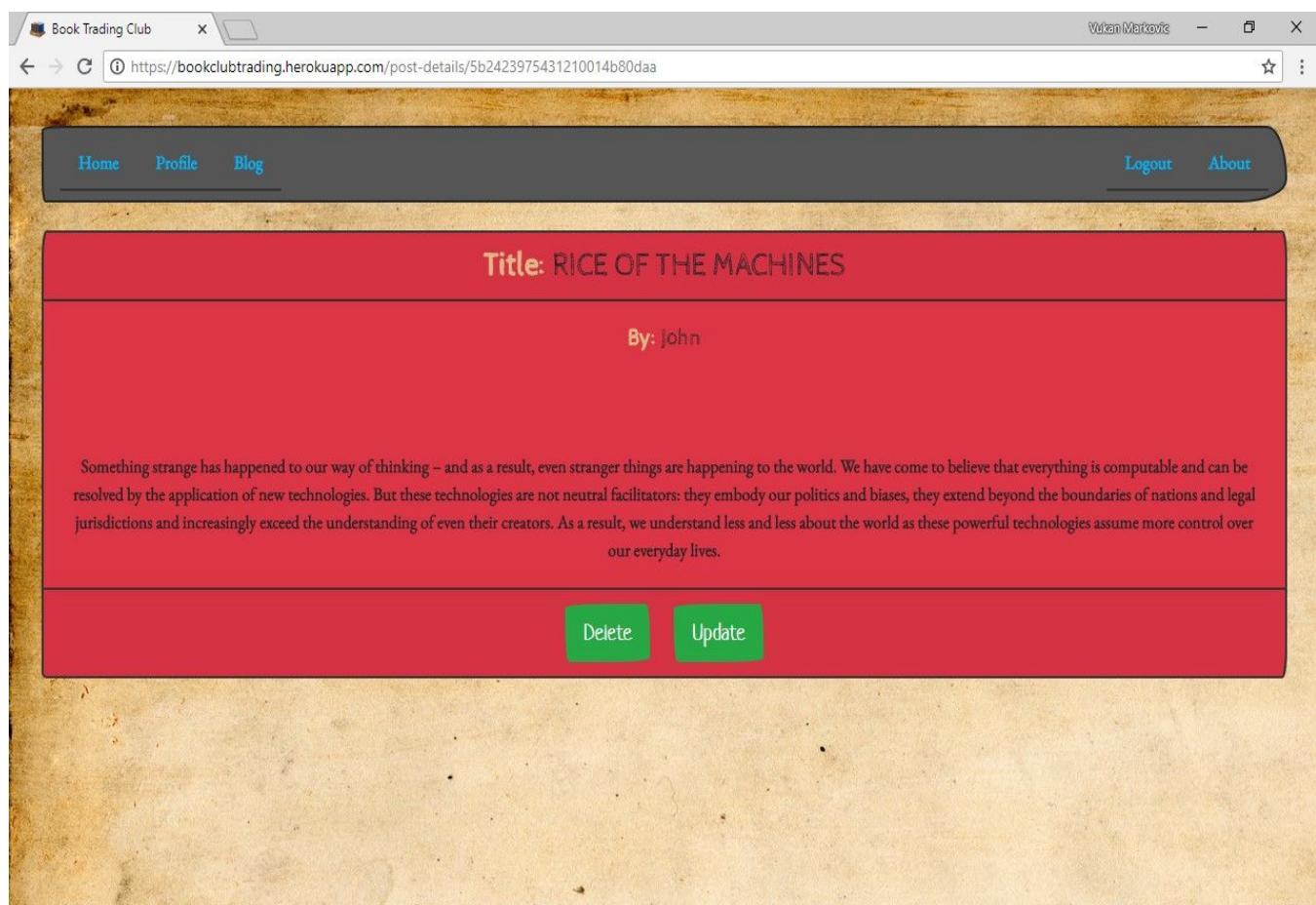
Submit

Not registered. Click [here](#) to join us!

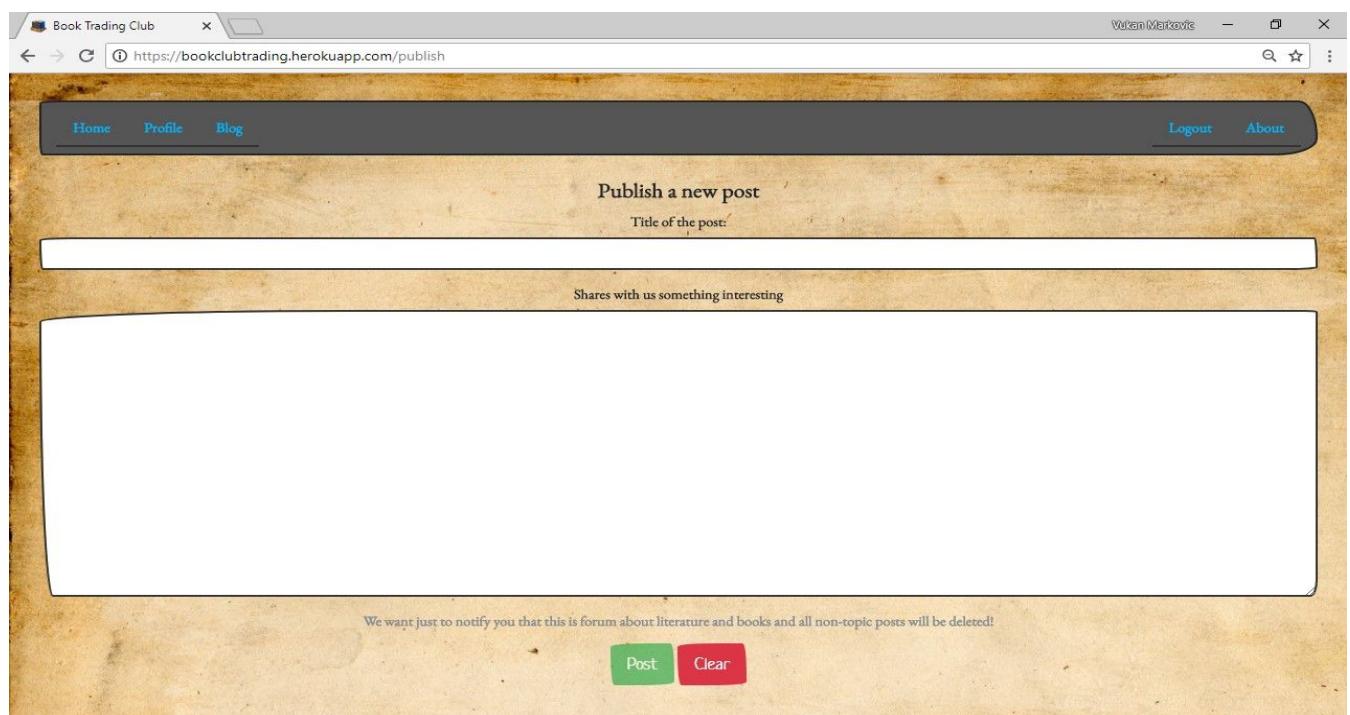
Slika 102. Administrator se identificuje preko email-a koji je jedinstven



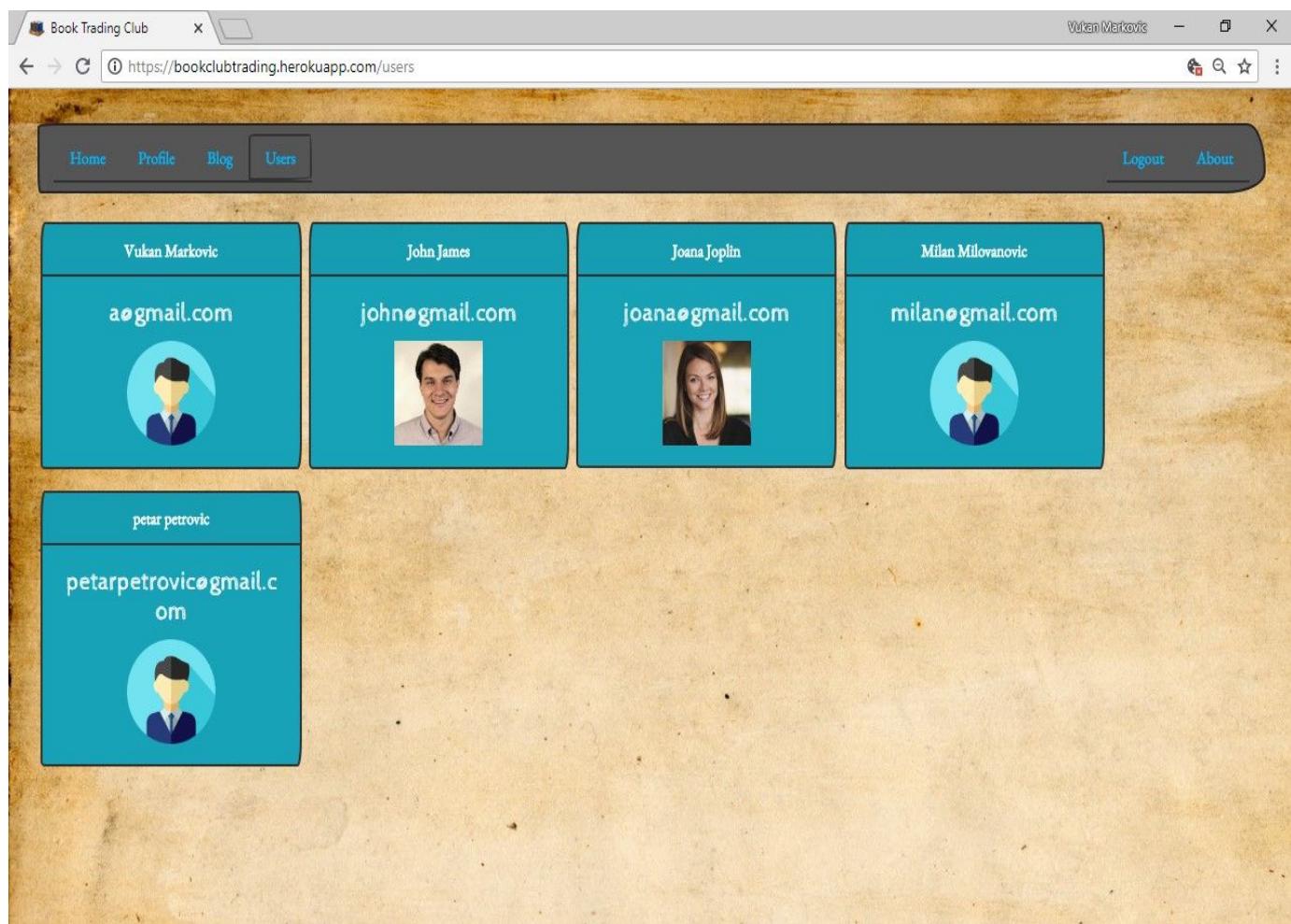
Slika 103. Detalji knjige sa aspekta administratora koji može da kupuje, briše, menja i komentariše sve knjige



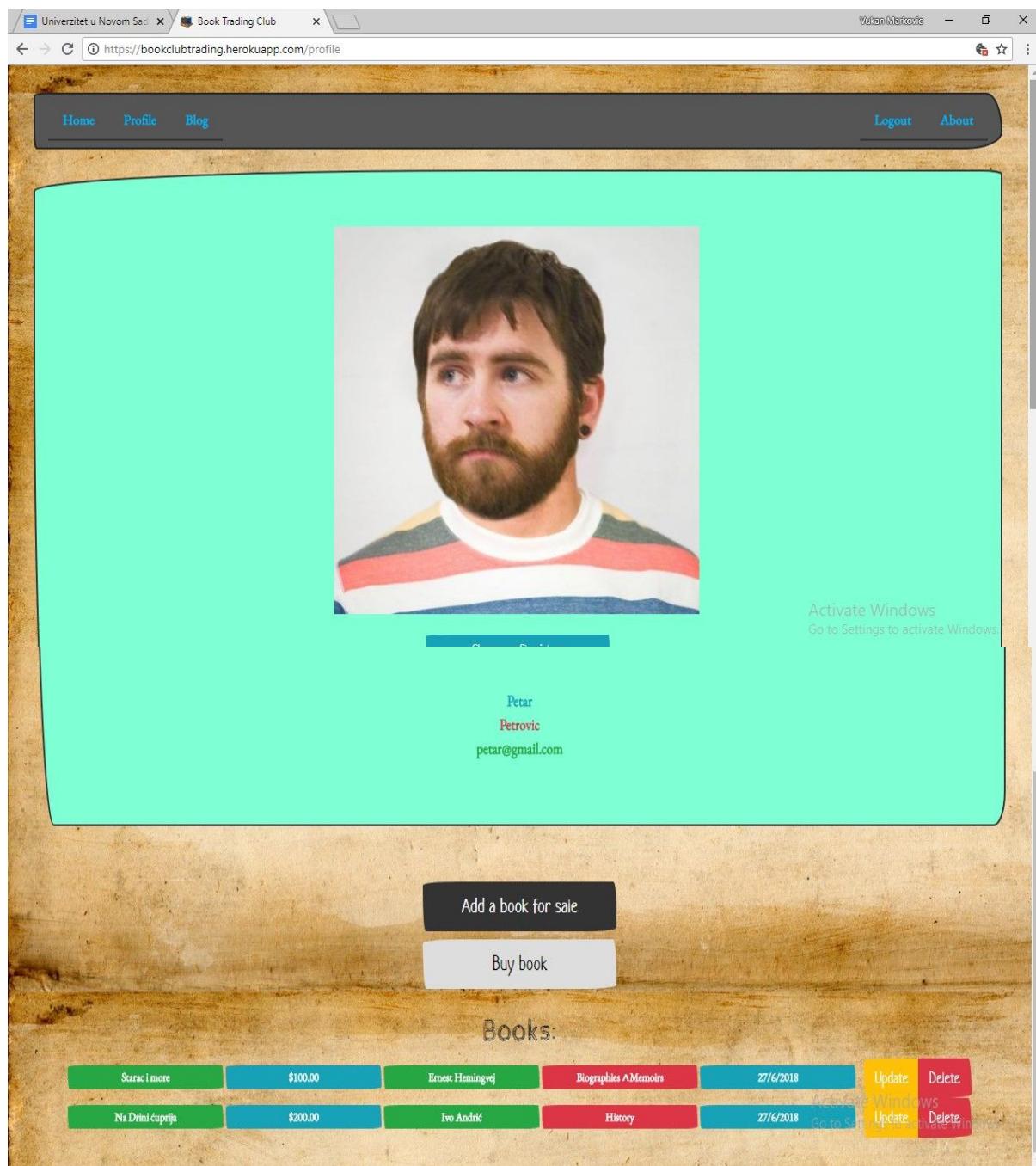
Slika 104. Objava na blogu sa aspekta administratora koji može da menja i briše sve objave



Slika 105. Objavljivanje novog posta, ovu mogućnost imaju samo registrovani korisnici



Slika 106. Administrator ima mogućnost pregleda svih registrovanih korisnika



Slika 107. Profilna strana

Univerzitet u Novom Sad Book Trading Club Vukan Marković

https://bookclubtrading.herokuapp.com/update-book/5b33042821ecc8001438f535

Update a book

Title:

Price:

Genre:

Author:

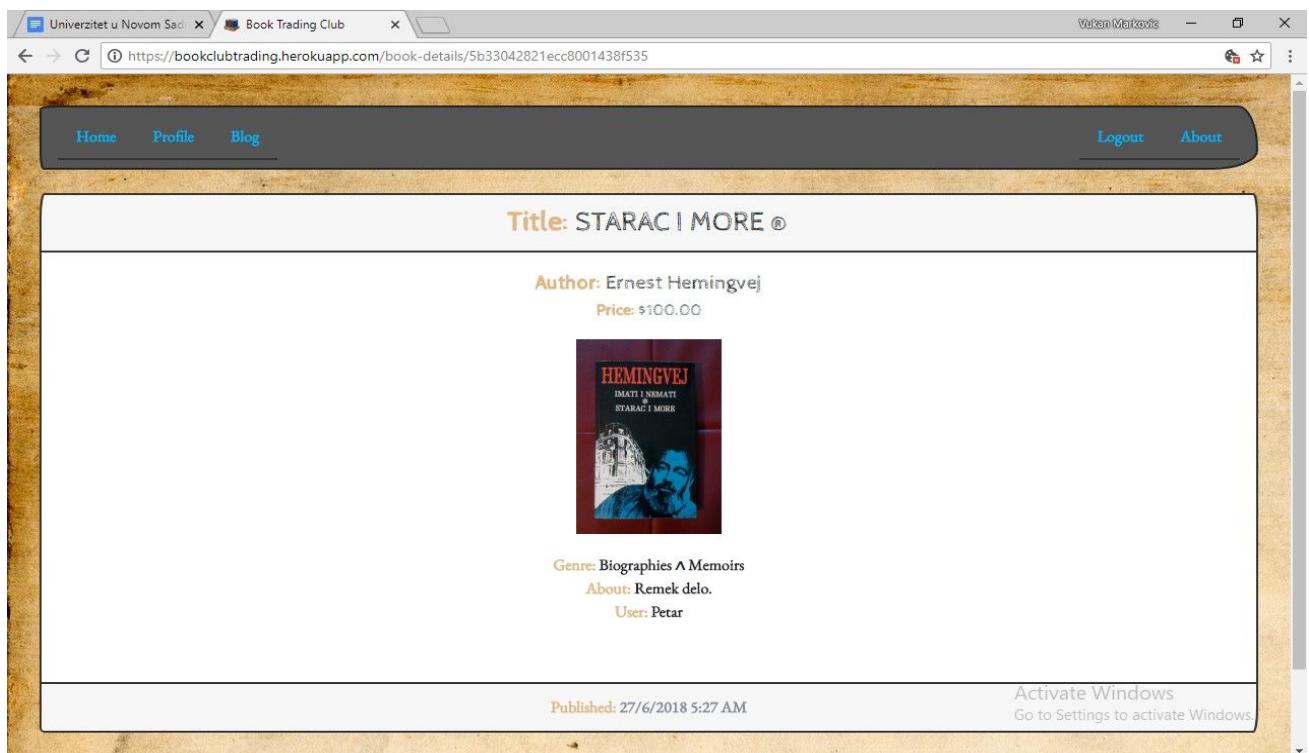
Write a couple of words about the book if you like:

Image:

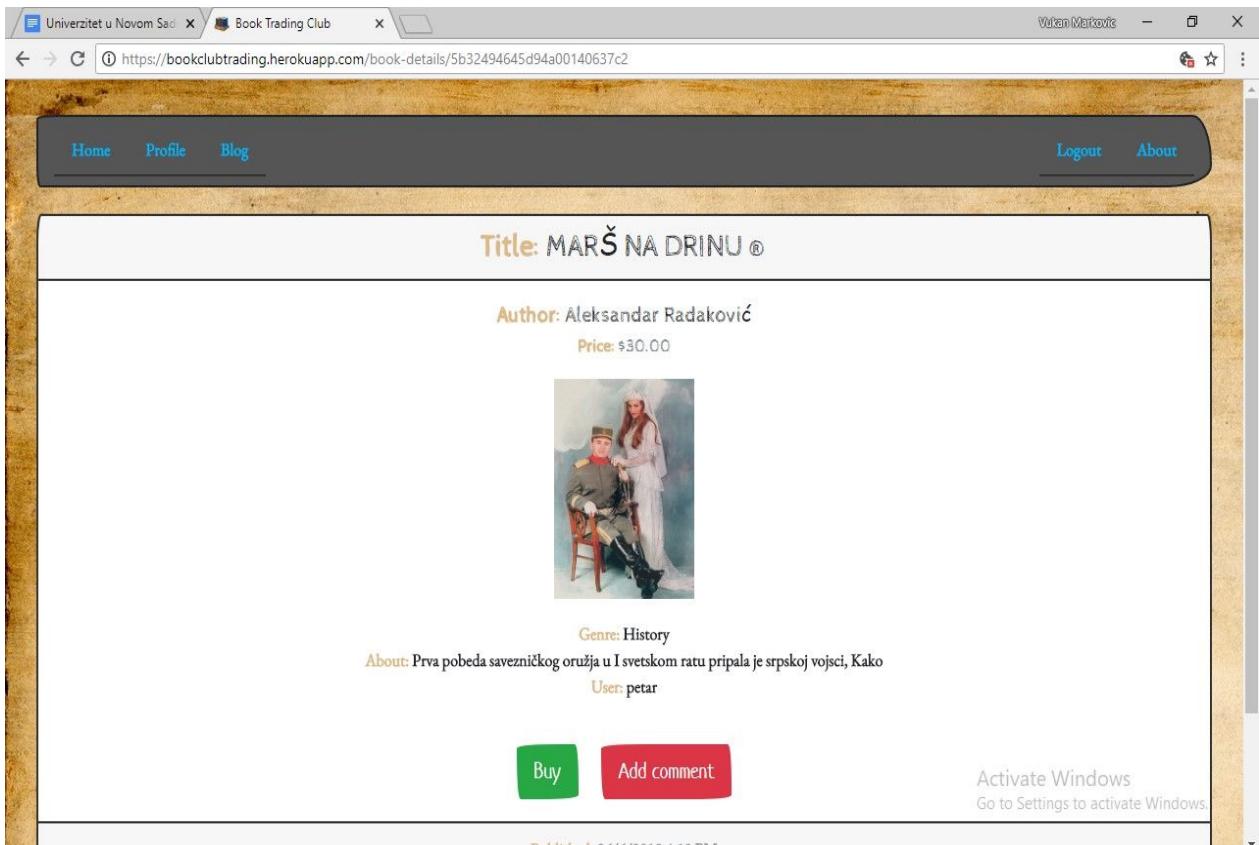
Update a book

Activate Windows
Go to Settings to activate Windows.

Slika 108. Ažuriranje knjige



Slika 109. Pregled korisnikove knjige



Slika 110. Pregled knjige od strane drugog korisnika

Univerzitet u Novom Sad Book Trading Club

Vukan Markovic

https://bookclubtrading.herokuapp.com/add-comment/5b32494645d94a00140637c2

Home Profile Blog Logout About

Add a new comment

This comment is about:

Book

Content:

Stvarno sjajna knjiga.

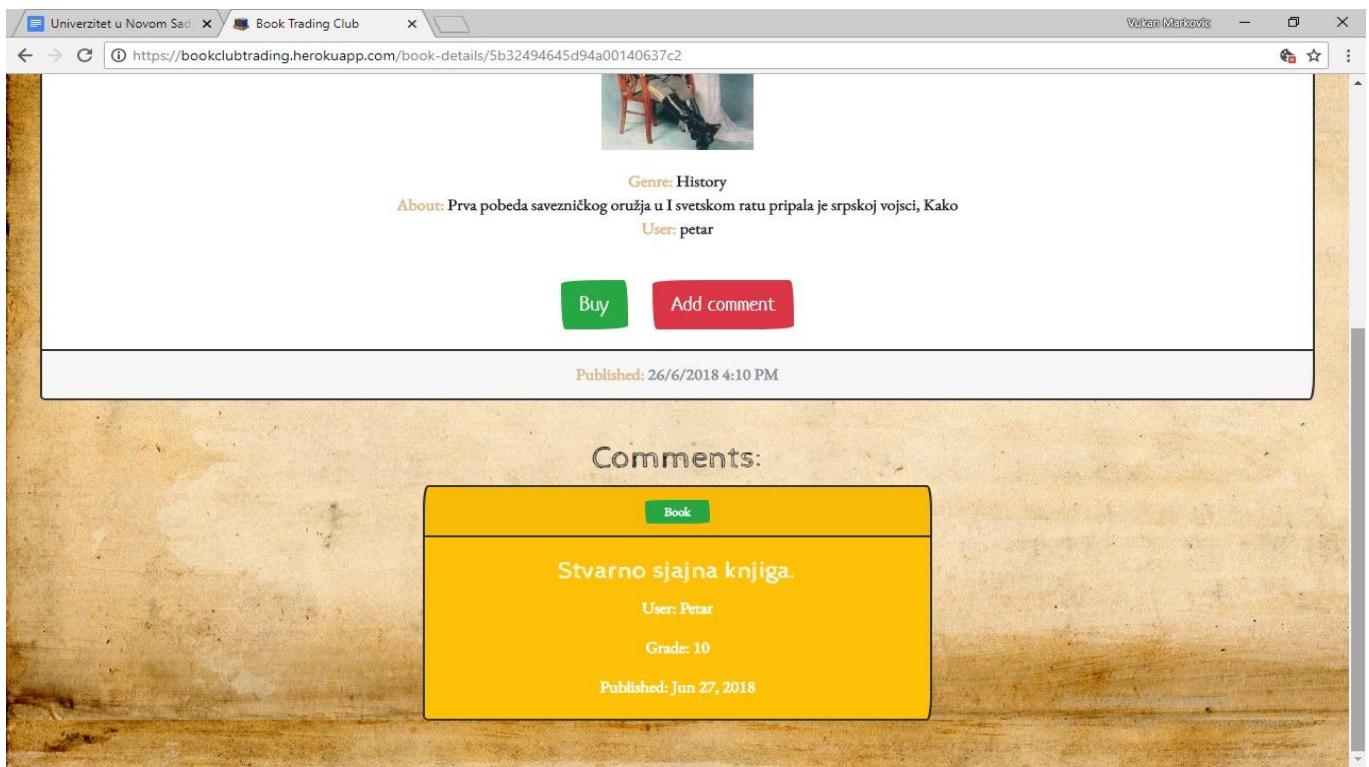
Grade:

10

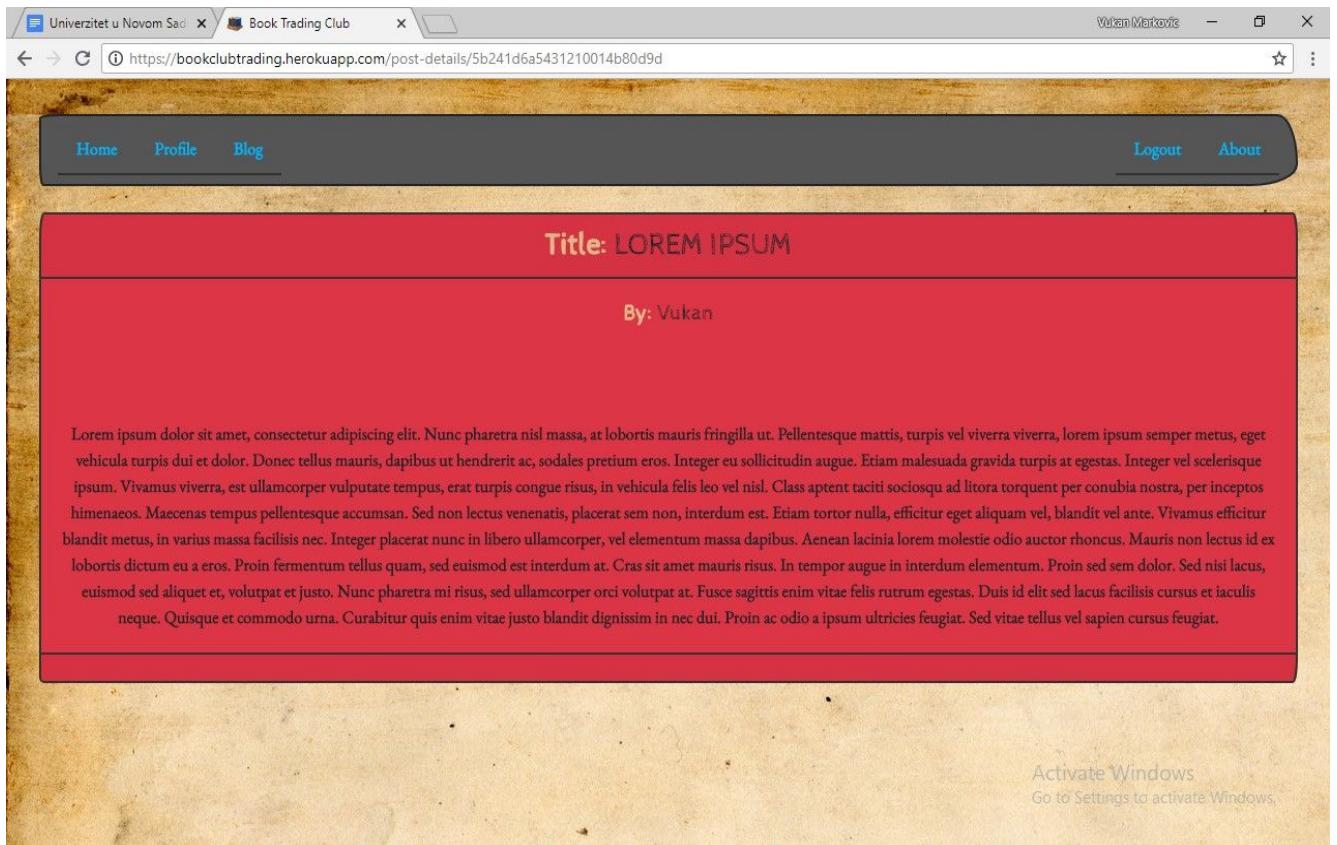
Clear Add a comment

Activate Windows
Go to Settings to activate Windows.

Slika 111. Dodavanje novog komentara



Slika 112. Dodani komentar

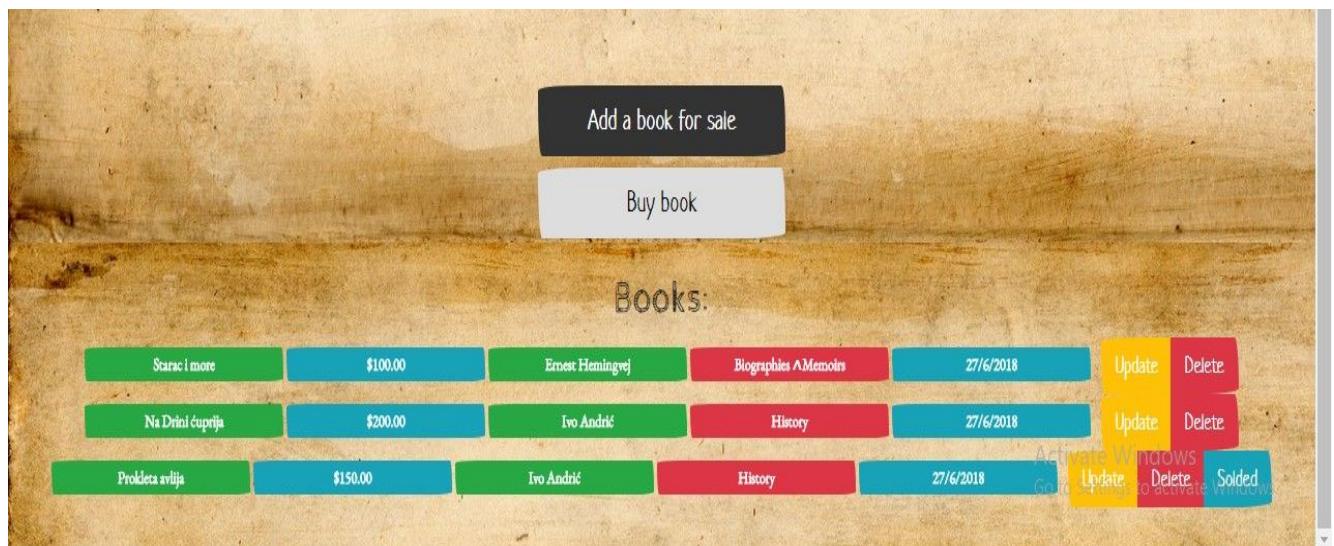


Slika 113. Pregled objave sa apseta korisnika koji nije administrator

Slika 114. Dodavanje nove knjige

A screenshot of a web browser window titled "Book Trading Club". The URL in the address bar is https://bookclubtrading.herokuapp.com/add-book. The page has a light brown background and is titled "Add a new book". It contains several input fields: "Title:" with the value "Prokleta avlja", "Price:" with the value "150", "Genre:" with the value "History" selected from a dropdown menu, "Author:" with the value "Ivo Andrić", and a text area for "Write a couple of words about the book if you like:" containing the text "Stvarno vredi pročitati.". Below these fields is an "Image:" input field containing the URL https://static.kupindoslike.com/Prokleta-avlja-Ivo-Andrić_slika_0_76314045.jl. At the bottom of the form are two buttons: a red "Clear" button and a green "Add a book" button. A watermark for Windows activation is visible at the bottom right.

Slika 115. Korisnik može da vidi podatke neophodne za obavljanje kupovine...



The screenshot shows a user profile page for "User: Jelena Petrovic". The page has a red background. At the top, it displays the user's name. Below the name is a large portrait photo of a woman with blonde hair, wearing a dark blazer over a white top. At the bottom of the page, there is a message from Microsoft encouraging activation of Windows. The message reads: "Activate Windows Go to Settings to activate Windows." Below the message, there is some small text: "Sombor 25000 Jevrejska 06231244341".

...onog korisnika koji ju je izvršio.

4.3 Bazapodataka

The screenshot shows the mLab MongoDB interface for the 'books' collection. At the top, there are navigation links for 'WELCOME', 'PLANS & PRICING', 'DOCS & SUPPORT', 'ACCOUNT', and 'LOG OUT'. The user information '(user: "Vukan", account: "Vukan")' is also displayed. Below the header, the URL 'https://mlab.com/databases/booktradingclub/collections/books' is shown. The main area has tabs for 'Documents', 'Indexes', 'Stats', and 'Tools'. The 'Documents' tab is selected. A search bar says '-- Start new search --'. A button to 'Delete all documents in collection' and a link to '+ Add document' are also present. The data grid displays 8 documents, each with an '_id' field containing a MongoDB ObjectId. The first document's details are visible: title 'The Neon Lawyer', price '2.4', genre 'Biographies & Memoirs'. The right side of the interface contains a sidebar with the heading 'Documents (aka Objects)' and instructions about browsing, searching, and deleting documents.

The screenshot shows the mLab MongoDB interface for the 'comments' collection. The layout is identical to the 'books' collection page. The 'Documents' tab is selected. A search bar says '-- Start new search --'. A button to 'Delete all documents in collection' and a link to '+ Add document' are also present. The data grid displays 5 documents, each with an '_id' field containing a MongoDB ObjectId. The first document's details are visible: content 'null', publishDate '2018-06-15T20:35:08.931Z'. The right side of the interface contains a sidebar with the heading 'Documents (aka Objects)' and instructions about browsing, searching, and deleting documents.

ObjectLabs Corporation [US] | https://mlab.com/databases/booktradingclub/collections/posts

mLab

WELCOME PLANS & PRICING DOCS & SUPPORT ACCOUNT LOG OUT

{ user: "Vukan", account: "Vukan" }

Home : { db : "booktradingclub" } Collection: posts

Documents Indexes Stats Tools

Documents

Delete all documents in collection Add document

-- Start new search --

All Documents

Display mode: list table (edit table view)

records / page 10 [1 - 4 of 4]

```
{
  "_id": {
    "$oid": "5b241d6a5431210014b80d9d"
  },
  "title": "Lorem ipsum",
  "content": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc pharetra nisl massa, at lobortis mauris fringilla ut. Pellentesque mattis, turpis vel viverra viverra, lorem ipsum semper"
},
{
  "_id": {
    "$oid": "5b241e0b5431210014b80d9f"
  },
  "title": "什么是Lorem Ipsum",
  "content": "Lorem Ipsum, 也称乱数假文或者哑元文本，是印刷及排版领域所常用的虚拟文字。由于曾经一台著名的打印机到香港打制了一套印刷字体从而造出一本字体样品书。Lorem Ipsum从西元15世纪起就作为此颜"
},
{
  "_id": {
    "$oid": "5b2423975431210014b80daa"
  },
  "title": "Rice of the machines",
  "content": "Something strange has happened to our way of thinking \u2013 and as a result, even stranger things are happening to the world. We have come to believe that everything is"
}
```

Documents (aka Objects)

From the "Documents" tab you can browse and search for objects in this collection. All standard query constructs are supported except for map/reduce queries. To use map/reduce, use the MongoDB shell (note that temporary result collections will be viewable in mLab).

You can also add, edit, and delete individual documents from here. Bulk collection updates are not yet supported in this UI (although they are supported in the shell).

ObjectLabs Corporation [US] | https://mlab.com/databases/booktradingclub/collections/users

mLab

WELCOME PLANS & PRICING DOCS & SUPPORT ACCOUNT LOG OUT

{ user: "Vukan", account: "Vukan" }

Home : { db : "booktradingclub" } Collection: users

Documents Indexes Stats Tools

Documents

Delete all documents in collection Add document

-- Start new search --

All Documents

Display mode: list table (edit table view)

records / page 10 [1 - 6 of 6]

```
{
  "_id": {
    "$oid": "5b23c39ecb7dfc0014934308"
  },
  "firstName": "Vukan",
  "lastName": "Markovic",
  "password": "$2a$10$MIkP0ZjdAqdhOhPLubM7utKEmrhCMPNnosaLG/RiSnz7eA4RffG",
},
{
  "_id": {
    "$oid": "5b241d205431210014b80d9c"
  },
  "firstName": "Vukan",
  "lastName": "Markovic",
  "password": "$2a$10$XnhuiI8woqpbGo1Kp29KyewLny1hj9kmo1z5ZE9ZiyFseng9/Hu/v",
},
{
  "_id": {
    "$oid": "5b2421bf5431210014b80da4"
  },
  "firstName": "John",
  "lastName": "James",
  "password": "$2a$10$mm1aFd3hTuNT5LtMJC8ui.LabXUV10OGIMd1YAT4uV8VMvtLV3Hba"
}
```

Documents (aka Objects)

From the "Documents" tab you can browse and search for objects in this collection. All standard query constructs are supported except for map/reduce queries. To use map/reduce, use the MongoDB shell (note that temporary result collections will be viewable in mLab).

You can also add, edit, and delete individual documents from here. Bulk collection updates are not yet supported in this UI (although they are supported in the shell).