Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektronsko poslovanje

"Klik do posla"

Internet tehnologije

Mentor Student

Jelena Škiljević 1052/18

Aleksandra Zdravković 440/14

Beograd, septembar 2022

Sadržaj

[1. Uvod 3](#_Toc114692408)

[2. Opis tehnologija 3](#_Toc114692409)

[2.1. MongoDB 3](#_Toc114692410)

[2.2. React 3](#_Toc114692411)

[2.3. Node 4](#_Toc114692412)

[2.4. Express 4](#_Toc114692413)

[3. "Klik do posla" 5](#_Toc114692414)

[3.1. Verbalni opis 5](#_Toc114692415)

[3.2. Slučajevi korišćenja 6](#_Toc114692416)

[3.1.1. SK1: Slučaj korišćenja – Registracija(register) na sistem 7](#_Toc114692417)

[3.1.2 SK2: Slučaj korišćenja – Odjava(Sign out) sa sistema 9](#_Toc114692418)

[3.1.3 SK3: Slučaj korišćenja – Prikaz svih objava 10](#_Toc114692419)

[3.1.4. SK4: Slučaj korišćenja – Dodavanje biografije 11](#_Toc114692420)

[3.1.5 SK5: Slučaj korišćenja – Dodavanje radnog iskustva 12](#_Toc114692421)

[3.1.6 SK6: Slučaj korišćenja - Kreiranje objave 13](#_Toc114692422)

[3.1.7 SK7: Slučaj korišćenja – Izmena objave 14](#_Toc114692423)

[3.1.8 SK8: Slučaj korišćenja – Komentarisanje objave 15](#_Toc114692424)

[3. Rest API 16](#_Toc114692425)

[4. Korisničko uputstvo i primer kodova 18](#_Toc114692426)

[4.1. Prikaz koda 18](#_Toc114692427)

[4.2. Korisničko uputstvo 30](#_Toc114692428)

[6. Zaključak 35](#_Toc114692429)

# Uvod

# Opis tehnologija

Prilikom izrade projekta , upotrebljene su tehnologije koje opisuju MERN Stack. MERN Stack tehnologije označavaju : MongoDB, Express, React, Node.

Prilikom izrade, korišćeni su MVC i Observer paterni.

## MongoDB

MongoDB je baza podataka orijentisana na više platformi koja pruža visoke performanse, visoku dostupnost i laku skalabilnost. MongoDB radi na konceptu kolekcije i dokumenta.

Baza podataka je fizički kontejner za kolekcije. Svaka baza podataka dobija svoj set datoteka na sistemu datoteka. Pojedinačni MongoDB server obično ima više baza podataka.

Kolekcija je grupa MongoDB dokumenata. To je ekvivalent RDBMS tabele. Kolekcija postoji u okviru jedne baze podataka. Kolekcije ne primenjuju šemu. Dokumenti u zbirci mogu imati različita polja. Tipično su svi dokumenti u zbirci slične ili srodne namene.

## React

ReactJS je JavaScript biblioteka koja se koristi za izgradnju komponenata korisničkog interfejsa za višekratnu upotrebu. Prema zvaničnoj dokumentaciji React-a, sledeća je definicija:

React je biblioteka za izgradnju korisničkih interfejsa koji se mogu komponovati. Podstiče stvaranje komponenata korisničkog interfejsa za višekratnu upotrebu, koje predstavljaju podatke koji se vremenom menjaju. Mnogi ljudi koriste React kao V u MVC. React apstrahova daleko od sebe, nudeći jednostavniji model programiranja i bolje performanse. React takođe može da se prikazuje na serveru pomoću Node-a i može pokretati matične aplikacije koristeći React Native. React implementira jednosmerni reaktivni protok podataka, koji smanjuje ploču i lakše je rasuđivati od tradicionalnog vezivanja podataka.

React koristi JSX za šabloniranje umesto uobičajenog JavaScript-a. Nije neophodno da ga koristite, međutim, u nastavku sledi nekoliko prednosti koje dolaze sa njim. Brži je jer vrši optimizaciju dok kompajlira kod u JavaScript. Takođe je bezbedan za tip i većina grešaka se može uočiti tokom kompajliranja. Olakšava i brže pisanje šablona ako ste upoznati sa HTML-om.

## Node

Node.js je platforma na strani servera izgrađena na Google Chrome-ovom JavaScript Engine (V8 Engine). Node.js je razvio Rian Dahl 2009. godine, a njegova najnovija verzija je v0.10.36. Definicija Node.js kako je data u njegovoj službenoj dokumentaciji je sledeća:

Node.js je platforma izgrađena na Chrome-ovom JavaScript izvođenju za lako stvaranje brzih i skalabilnih mrežnih aplikacija. Node.js koristi ne-blokirajući I / O model vođen događajima koji ga čini laganim i efikasnim, savršenim za aplikacije u realnom vremenu koje zahtevaju podatke i rade na distribuiranim uređajima.

Node.js je otvoreno razvojno okruženje izvršavanja za više platformi za razvoj aplikacija na strani servera i mreža. Node.js aplikacije su napisane na JavaScript-u i mogu se pokretati u okviru vremena izvršavanja Node.js na OS X, Microsoft Windows i Linux.

Node.js takođe nudi bogatu biblioteku različitih JavaScript modula koji u velikoj meri pojednostavljuju razvoj veb aplikacija pomoću Node.js-a.

Node.js = Runtime Environment + JavaScript Library.

## Express

Express pruža minimalan interfejs za izgradnju naših aplikacija. Pruža nam alate potrebne za izgradnju naše aplikacije. Fleksibilan je jer na npm postoje brojni moduli koji se mogu direktno priključiti na Express.

Express je razvio TJ Holovaichuk, a održavaju ga fondacija Node.js i brojni saradnici otvorenog koda.

Za razliku od konkurenata poput Rails-a i Django-a, koji imaju samopouzdan način izrade aplikacija, Ekpress nema „najbolji način“ da nešto učini. Veoma je fleksibilan i može se priključiti bilo kojoj front strani kao i bazi. U okviru ovog projekta, korišćena je nerelaciona Mongo baza koja je opisana u prethodnom poglavlju.

Reagiranje u kombinaciji sa industrijski testiranim tehnologijama na strani servera, kao što su Node, Express i MongoDB, omogućava vam da razvijete i primenite robusne veb aplikacije sa punim slogovima u stvarnom svetu.

# "Klik do posla"

U okviru ovog poglavlja, korisnik će moći da vidi više o verbalnom opisu aplikacije, kao i da pročita koji su slučajevi korišćenja.

## Verbalni opis

Aplikacija "Klik do posla" je web aplikacija koja služi za pronalazak posla, i koristi javni API kao i web aplikacija HelloWorld.rs . Korisnik će imati privilegiju da pregleda sve objave koje su ostali korisnici kreirali, kao i to da lajkuje, komentariše i uđe na određeni profil korisnika, isto tako moći će da doda svoju biografiju, da doda svoj nivo obrazovanja, kao i da gleda drugima njihove podatke, da postavlja oglase za posao, kao i da kači objavu gde će tražiti posao.

## Slučajevi korišćenja

Slučajevi korišćenja:

1. Registracija na sistem
2. Prijava(Sign in) na sistem
3. Odjava(Sign out) sa sistema
4. Prikaz svih objava
5. Dodavanje biografije
6. Dodavanje radnog iskustva
7. Kreiranje objave
8. Izmena objave
9. Komentarisanje objave

### SK1: Slučaj korišćenja – Registracija(register) na sistem

**Naziv** **SK**:

Registracija(register) na sistem

**Akteri SK:**

Korisnik

**Učesnici**:

Korisnik i sistem

**Preduslov**: /

Osnovni scenario:

1. Korisnik otvara odgovarajuću formu za registraciju.(APSO)
2. Sistem vraća formu za registraciju. (IA)
3. Nakon unosa podataka korisnik pritiska dugme “Registruj se.“(APSO)
4. Sistem prihvata podatke, proverava ih i šalje poruku „Uspešno ste se registrovali“.(IA)

**Dijagram** **sekvenci**:



Alternativni scenario:

* 1. Sistem javlja da uneto korisničko korisničko ili email ime već postoji u bazi porukom “Korisnik sa ovim korisničkim imenom već postoji" ili "Korisnik sa ovim emailom već postoji". (IA)

**Dijagram sekvenci:**



### SK2: Slučaj korišćenja – Odjava(Sign out) sa sistema

**Naziv SK:**

Odjava(Sign out) sa sistema

**Akteri SK:**

Korisnik

**Učesnici:**

Korisnik i sistem

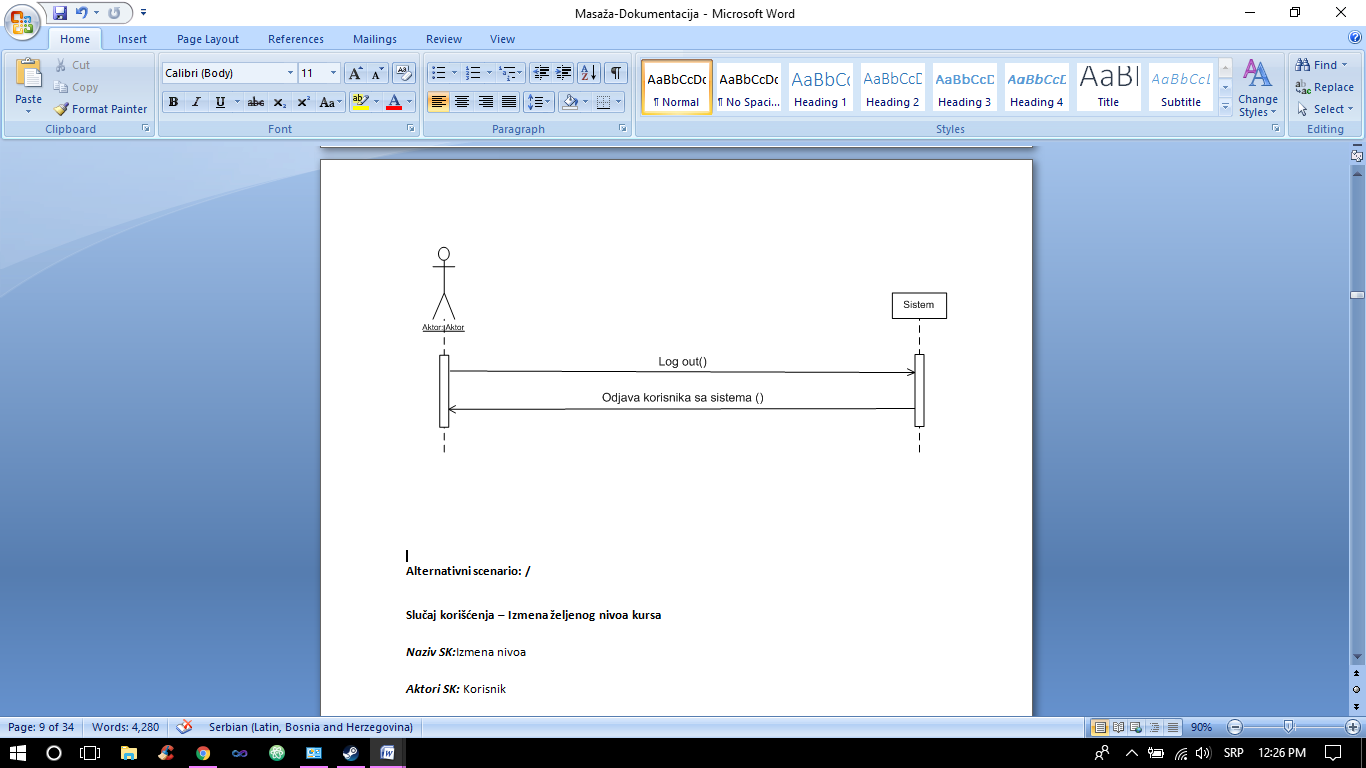
**Preduslov:**

Akter je predhodno ulogovan na sistem

Osnovni scenario:

1. Korisnik bira opciju „Log out“ iz korisničkog menija. (APSO)
2. Sistem odjavljuje korisnika i vraća ga na stranu za logovanje. (IA)

**Dijagram sekvenci:**



### SK3: Slučaj korišćenja – Prikaz svih objava

**Naziv** **SK**:

Prikaz svih objava

**Aktori SK:**

Korisnik

**Učesnici**:

Korisnik i sistem

**Preduslov**:

Sistem je uključen. Korisnik je ulogovan preko svog naloga..

Osnovni scenario SK:

1. Korisnik refreshuje početnu stranu(APSO)
2. Sistem prikazuje sve objave od svih korisnika(IA)

Alternativni scenario:

* 1. Nijedna objava ne postoji u sistemu(IA)

### SK4: Slučaj korišćenja – Dodavanje biografije

**Naziv SK:**

Dodavanje biografije

**Aktori** SK:

Korisnik

**Učesnici**:

Korisnik i sistem

**Preduslov**:

Sistem je uključen. Korisnik je ulogovan preko svog naloga.

Osnovni scenario SK:

1. Korisnik se loguje na sistem(APSO)
2. Sistem prikazuje sve formu za dodavanje biografije(SO)
3. Korisnik unosi podatke u formu(APUSO)
4. Korisnik poziva sistem da unese podatke(APSO)
5. Sistem prikazuje unetu biografiju(IA)

Alternativni scenario:

* 1. Korisnik ne postoji u sistemu(IA)

### 3.1.5 SK5: Slučaj korišćenja – Dodavanje radnog iskustva

**Naziv SK:**

Dodavanje radnog iskustva

**Aktori SK:**

Korisnik

**Učesnici SK:**

Korisnik i sistem

**Preduslov:**

Sistem je uključen. Korisnik je ulogovan na sistem.

Osnovni scenario SK:

1. Korisnik se loguje na sistem(APSO)
2. Sistem prikazuje sve formu za dodavanje biografije(SO)
3. Korisnik unosi podatke u formu(APUSO)
4. Korisnik poziva sistem da unese podatke(APSO)
5. Sistem prikazuje unetu biografiju(IA)

Alternativni scenario:

:

* 1. Ne postoje korisnik u sistemu(IA)

3.1. Korisnik nema radno iskustvo(IA)

### SK6: Slučaj korišćenja - Kreiranje objave

**Naziv SK:**

Kreiranje objave(Oglasa za posao)

**Aktori SK:**

Korisnik

**Učesnici:**

Korisnik i sistem

**Preduslov:**

Sistem je uključen. Korisnik je ulogovan preko svog naloga. Sistem prikazuje formu za rad za kreiranje novog posta.

Osnovni scenario SK:

1. Korisnik unosi podatke u formu za kreiranje posta(APUSO)
2. Korisnik poziva sistem da kreira post(APSO)
3. Sistem kreira post(SO)
4. Sistem pamti podatke o kreiranom postu i prikazuje taj post(IA)

Alternativni scenario:

3.1 Ukoliko ne može sistem da kreira post, sistem će resetovati podatke i neće kreirati post.

4.1 Ukoliko sistem ne može da zapamti podatke o kreiranom postu. Post neće biti zapamćen i neće biti unet.

### SK7: Slučaj korišćenja – Izmena objave

**Naziv SK:**

Izmena posta sa sistema.

**Aktori SK:**

Korisnik

**Učesnici:**

Korisnik i sistem

**Preduslov:**

Sistem je uključen. Korisnik je ulogovan preko svog naloga. Korisnik može da menja isključivo postove koje je objavio prethodno. Sistem prikazuje formu za rad za kreiranje novog posta.

Osnovni scenario SK:

1. Korisnik pretražuje svoje postojve(APSO)
2. Sistem vraća sve postove, i pored korisnikovih postavlja dugme za izmenu(SO)
3. Korisnik poziva sistem da izmeni post pritiskom na dugme(APSO)
4. Sistem vraća korisniku konkretan post(SO)
5. Korisnik unosi podatke za izmenu(APUSO)
6. Korisnik poziva sistem da izmeni podatke(APSO)
7. Sistem menja post i prikazuje novi izmenjen post(IA)

Alternativna scenarija:

1.1 Korisnik mora biti ulogovan da bi video koji su njegovi postovi

1.2 Korisnik nema nijedan objavljen post

5.1 Korisnik unosi podatke koji nisu validni

4.1. Server je pokrenut i konekcija prema bazi nije stabilna

### SK8: Slučaj korišćenja – komentarisanje objave

**Naziv SK:**

Komentarisanje objave

**Aktori SK:**

Korisnik

**Učesnici:**

Korisnik i sistem

**Preduslov:**

Sistem je uključen. Korisnik je ulogovan preko svog naloga.

Osnovni scenario SK:

1. Korisnik pretražuje postojve(APSO)
2. Sistem vraća sve postove(SO)
3. Korisnik poziva sistem da komentarise post pritiskom na dugme(APSO)
4. Sistem vraća korisniku formu za komentar(SO)
5. Korisnik unosi podatke u formu(APUSO)
6. Korisnik poziva sistem da sacuva komentar(APSO)
7. Sistem pamti komentar i prikazuje ga(IA)

Alternativna scenarija:

1.1 Korisnik mora biti ulogovan da bi komentarisao

1.2 Nema nijdan post na sistemu

# Rest API

|  |  |
| --- | --- |
| Opis funkcije | Lajkovanje objave |
| HTTP metoda | Put |
| URL | PUT api/posts/like/:id |
| URL parametri | (nema) |
| HTTP body parametri | User.Login |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  “Post successfully liked”  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Pregled svih objava |
| HTTP metoda | GET |
| URL | GET api/posts/ |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "\_id": {  "$oid": "556e2e321e5ea324317433e17"  },  "role": {  "$numberInt": "1"  },  "cart": [],  "name": "Jelena Skiljevic",  "email": "jelena@gmail.com",  “post”: “Objava za posao”  "password": "$2b$10$u7nWur3fRmYKOW8mO4lE5OH/ruwL7aKupBNyecoCEdipp2jf644TW",  "createdAt": {  "$date": {  "$numberLong": "25164168464"  }  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Brisanje postojećeg posta |
| HTTP metoda | DELETE |
| URL | GET api/posts/:id |
| URL parametri | (nema) |
| HTTP body parametri | user |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | “Post successfullly deleted” |
| Format izlaznih parametara | application/json |

# Korisničko uputstvo i primer kodova

U okviru ovog poglavlja biće prvo prikazani najinteresantniji delovi koda, a pritom i korisničko uputstvo.

## Prikaz koda

Server.js:

const express = require('express');

const connectDB = require('./config/db');

const path = require('path');

const app = express();

//connect database

connectDB();

//Initialise middleware

app.use(express.json({ extended: false }));

//Define routes

app.use('/api/users', require('./routes/api/users'));

app.use('/api/auth', require('./routes/api/auth'));

app.use('/api/profile', require('./routes/api/profile'));

app.use('/api/posts', require('./routes/api/posts'));

//Serve Static Assets

if (process.env.NODE\_ENV === 'production') {

//set static folder

app.use(express.static('client/build'));

app.get('\*', (req, res) => {

res.sendFile(path.resolve(\_\_dirname, 'client', 'build', 'index.html'));

});

}

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server started on port ${PORT}`));

Ovde se vrši konekcija na bazu, osluškivanje, kao i definisanje route-a za logiku i API pozive. Konekcija na bazu:

const moongose = require('mongoose');

const config = require('config');

const db = config.get('mongoURI');

//calling database using async await

const connectDB = async () => {

try {

await moongose.connect('mongodb+srv://user:user123@cluster0.dfynd.mongodb.net/?retryWrites=true&w=majority', {

useNewUrlParser: true,

useUnifiedTopology: true,

useCreateIndex: true,

useFindAndModify: false,

});

console.log('MongoDb Connected..');

} catch (err) {

console.error(err.message);

//exit process with failure

process.exit(1);

}

};

module.exports = connectDB;

Route:

app.use('/api/users', require('./routes/api/users'));

app.use('/api/auth', require('./routes/api/auth'));

app.use('/api/profile', require('./routes/api/profile'));

app.use('/api/posts', require('./routes/api/posts'));

U nastavku će biti definisano samo za jednu posts.js U njoj se definišu putanje i api metode koje omogućavaju klijentu da izvrši određenu operaciju na određenoj ruti i da dobije određene povratne info od servera iz baze. U okviru rutiranja je ubačena logika sistema, može logika posebno da se izvdvoji u folder controller.

const express = require('express');

const router = express.Router();

const auth = require('../../middleware/auth');

const { check, validationResult } = require('express-validator');

const Post = require('../../models/Post');

const Profile = require('../../models/Profile');

const User = require('../../models/User');

// @route Post api/posts

// @desc Create a post

//@access Private

router.post(

'/',

[auth, [check('text', 'Text is required').not().isEmpty()]],

async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

try {

const user = await User.findById(req.user.id).select('-password');

const newPost = new Post({

text: req.body.text,

name: user.name,

avatar: user.avatar,

user: req.user.id,

});

const post = await newPost.save();

res.json(post);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

}

);

// @route GET api/posts

// @desc See all posts

//@access Private

router.get('/', auth, async (req, res) => {

try {

const posts = await Post.find().sort({ date: -1 });

res.json(posts);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

});

// @route GET api/posts/:id

// @desc get post by id

//@access Private

router.get('/:id', auth, async (req, res) => {

try {

const post = await Post.findById(req.params.id);

if (!post) {

return res.status(404).json({ msg: 'Post not found' });

}

res.json(post);

} catch (err) {

console.error(err.message);

if (err.kind === 'ObjectId') {

return res.status(404).json({ msg: 'Post not found' });

}

res.status(500).send('Server Error');

}

});

// @route GET api/posts/:id

// @desc get post by id

//@access Private

router.delete('/:id', auth, async (req, res) => {

try {

const post = await Post.findById(req.params.id);

if (!post) {

return res.status(404).json({ msg: 'Post not found' });

}

if (post.user.toString() !== req.user.id) {

return res.status(401).json({ msg: 'User not authorized' });

}

await post.remove();

res.json({ msg: 'Post removed' });

} catch (err) {

console.error(err.message);

if (err.kind === 'ObjectId') {

return res.status(404).json({ msg: 'Post not found' });

}

res.status(500).send('Server Error');

}

});

// @route PUT api/posts/like/:id

// @desc like a post

//@access Private

router.put('/like/:id', auth, async (req, res) => {

try {

const post = await Post.findById(req.params.id);

//check if post has already been liked

if (

post.likes.filter((like) => like.user.toString() === req.user.id)

.length > 0

) {

return res.status(400).json({ msg: 'Post already liked' });

}

post.likes.unshift({

user: req.user.id,

});

await post.save();

res.json(post.likes);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

});

// @route PUT api/posts/unlike/:id

// @desc Unlike a post

//@access Private

router.put('/unlike/:id', auth, async (req, res) => {

try {

const post = await Post.findById(req.params.id);

//check if post has already been liked

if (

post.likes.filter((like) => like.user.toString() === req.user.id)

.length === 0

) {

return res.status(400).json({ msg: 'Post has not yet been liked' });

}

//get remove index

const removeIndex = post.likes

.map((like) => like.user.toString())

.indexOf(req.user.id);

post.likes.splice(removeIndex, 1);

await post.save();

res.json(post.likes);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

});

// @route POST api/posts/comment/:id

// @desc Comment on a post

//@access Private

router.post(

'/comment/:id',

[auth, [check('text', 'Text is required').not().isEmpty()]],

async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

try {

const user = await User.findById(req.user.id).select('-password');

const post = await Post.findById(req.params.id);

const newComment = {

text: req.body.text,

name: user.name,

avatar: user.avatar,

user: req.user.id,

};

post.comments.unshift(newComment);

await post.save();

res.json(post.comments);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

}

);

// @route DELETE api/posts/comment/:id/:comment\_id

// @desc Delete comment

//@access Private

router.delete('/comment/:id/:comment\_id', auth, async (req, res) => {

try {

const post = await Post.findById(req.params.id);

const comment = post.comments.find(

(comment) => comment.id === req.params.comment\_id

);

if (!comment) {

return res.status(404).json({ msg: 'Comment does not exists' });

}

//check for user

if (comment.user.toString() !== req.user.id) {

return res.status(401).json({ msg: 'User not authorized' });

}

//get remove index

const removeIndex = post.comments

.map((comment) => comment.user.toString())

.indexOf(req.user.id);

post.comments.splice(removeIndex, 1);

await post.save();

res.json(post.comments);

} catch (err) {

console.error(err.message);

res.status(500).send('Server Error');

}

});

module.exports = router;

posts.js Model:

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const PostSchema = new Schema({

user: {

type: Schema.Types.ObjectId,

ref: 'user',

},

text: {

type: String,

required: true,

},

name: {

type: String,

},

avatar: {

type: String,

},

likes: [

{

user: {

type: Schema.Types.ObjectId,

ref: 'user',

},

},

],

comments: [

{

user: {

type: Schema.Types.ObjectId,

ref: 'user',

},

text: {

type: String,

required: true,

},

name: {

type: String,

},

avatar: {

type: String,

},

date: {

type: Date,

default: Date.now,

},

},

],

date: {

type: Date,

default: Date.now,

},

});

module.exports = Post = mongoose.model('post', PostSchema);

Na klijentskom delu se nalazi ulepšavanje komponenti putem React-a. Ovde će biti ispraćeno sve oko komponente vezane za objavu kao i samo slanje klijentskog zahteva na serversku stranu.

Komponenta Posts.js:

import React, { Fragment, useEffect } from 'react';

import PropTypes from 'prop-types';

import { connect } from 'react-redux';

import Spinner from '../layout/Spinner';

import PostItem from './PostItem';

import { getPosts } from '../../actions/post';

import PostForm from './PostForm';

const Posts = ({ getPosts, post: { posts, loading } }) => {

useEffect(() => {

getPosts();

}, [getPosts]);

return loading ? (

<Spinner />

) : (

<Fragment>

<h1 className='large text-primary'>Posts</h1>

<p className='lead'>

<i className='fas fa-user'></i> Welcome to the Community

</p>

<PostForm />

<div className='posts'>

{posts.map((post) => (

<PostItem key={post.\_id} post={post} />

))}

</div>

</Fragment>

);

};

Posts.propTypes = {

getPosts: PropTypes.func.isRequired,

post: PropTypes.object.isRequired,

};

const mapStateToProps = (state) => ({

post: state.post,

});

export default connect(mapStateToProps, { getPosts })(Posts);

Prikaz jedne objave je komponenta PostItem.js

import React, { Fragment } from 'react';

import PropTypes from 'prop-types';

import { Link } from 'react-router-dom';

import Moment from 'react-moment';

import { addLike, removeLike, deletePost } from '../../actions/post';

import { connect } from 'react-redux';

const PostItem = ({

auth,

addLike,

removeLike,

deletePost,

showActions,

post: { \_id, text, name, avatar, user, likes, comments, date },

}) => {

return (

<div class='post bg-white p-1 my-1'>

<div>

<Link to={`/profile/${user}`}>

<img class='round-img' src={avatar} alt='' />

<h4>{name}</h4>

</Link>

</div>

<div>

<p class='my-1'>{text}</p>

<p class='post-date'>

Posted on <Moment format='DD/MM/YYYY'>{date}</Moment>

</p>

{showActions && (

<Fragment>

<button

onClick={(e) => addLike(\_id)}

type='button'

class='btn btn-light'

>

<i class='fas fa-thumbs-up'></i>{' '}

{likes.length > 0 && <span>{likes.length}</span>}

</button>

<button

onClick={(e) => removeLike(\_id)}

type='button'

class='btn btn-light'

>

<i class='fas fa-thumbs-down'></i>

</button>

<Link to={`/posts/${\_id}`} class='btn btn-primary'>

Discussion{' '}

{comments.length > 0 && (

<span class='comment-count'>

{comments.length}

</span>

)}

</Link>

{!auth.loading && user === auth.user.\_id && (

<button

onClick={(e) => deletePost(\_id)}

type='button'

class='btn btn-danger'

>

<i class='fas fa-times'></i>

</button>

)}

</Fragment>

)}

</div>

</div>

);

};

PostItem.defaultProps = {

showActions: true,

};

PostItem.propTypes = {

post: PropTypes.object.isRequired,

auth: PropTypes.object.isRequired,

addLike: PropTypes.func.isRequired,

removeLike: PropTypes.func.isRequired,

deletePost: PropTypes.func.isRequired,

};

const mapStateToProps = (state) => ({

auth: state.auth,

});

export default connect(mapStateToProps, { addLike, removeLike, deletePost })(

PostItem

);

U okviru action dela se šalju klijentski zahtevi, i dalje će biti prikaz samo za jednu putanju tj. post.js akciju:

import axios from 'axios';

import { setAlert } from './alert';

import {

ADD\_POST,

DELETE\_POST,

GET\_POSTS,

POST\_ERROR,

UPDATE\_LIKES,

GET\_POST,

ADD\_COMMENT,

REMOVE\_COMMENT,

} from './types';

//get posts

export const getPosts = () => async (dispatch) => {

try {

const res = await axios.get('/api/posts');

dispatch({

type: GET\_POSTS,

payload: res.data,

});

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Add like

export const addLike = (postId) => async (dispatch) => {

try {

const res = await axios.put(`/api/posts/like/${postId}`);

dispatch({

type: UPDATE\_LIKES,

payload: { postId, likes: res.data },

});

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Remove like

export const removeLike = (postId) => async (dispatch) => {

try {

const res = await axios.put(`/api/posts/unlike/${postId}`);

dispatch({

type: UPDATE\_LIKES,

payload: { postId, likes: res.data },

});

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Delete post

export const deletePost = (postId) => async (dispatch) => {

try {

const res = await axios.delete(`/api/posts/${postId}`);

dispatch({

type: DELETE\_POST,

payload: postId,

});

dispatch(setAlert('Post Removed', 'success'));

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Add post

export const addPost = (formData) => async (dispatch) => {

const config = {

headers: {

'Content-Type': 'application/json',

},

};

try {

const res = await axios.post(`/api/posts`, formData, config);

dispatch({

type: ADD\_POST,

payload: res.data,

});

dispatch(setAlert('Post Created', 'success'));

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//get post

export const getPost = (id) => async (dispatch) => {

try {

const res = await axios.get(`/api/posts/${id}`);

dispatch({

type: GET\_POST,

payload: res.data,

});

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Add comment

export const addComment = (postId, formData) => async (dispatch) => {

const config = {

headers: {

'Content-Type': 'application/json',

},

};

try {

const res = await axios.post(

`/api/posts/comment/${postId}`,

formData,

config

);

dispatch({

type: ADD\_COMMENT,

payload: res.data,

});

dispatch(setAlert('Comment Added', 'success'));

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

},

});

}

};

//Delete comment

export const deleteComment = (postId, commentId) => async (dispatch) => {

try {

const res = await axios.delete(

`/api/posts/comment/${postId}/${commentId}`

);

dispatch({

type: REMOVE\_COMMENT,

payload: commentId,

});

dispatch(setAlert('Comment Removed', 'success'));

} catch (err) {

dispatch({

type: POST\_ERROR,

payload: {

msg: err.response.statusText,

status: err.response.status,

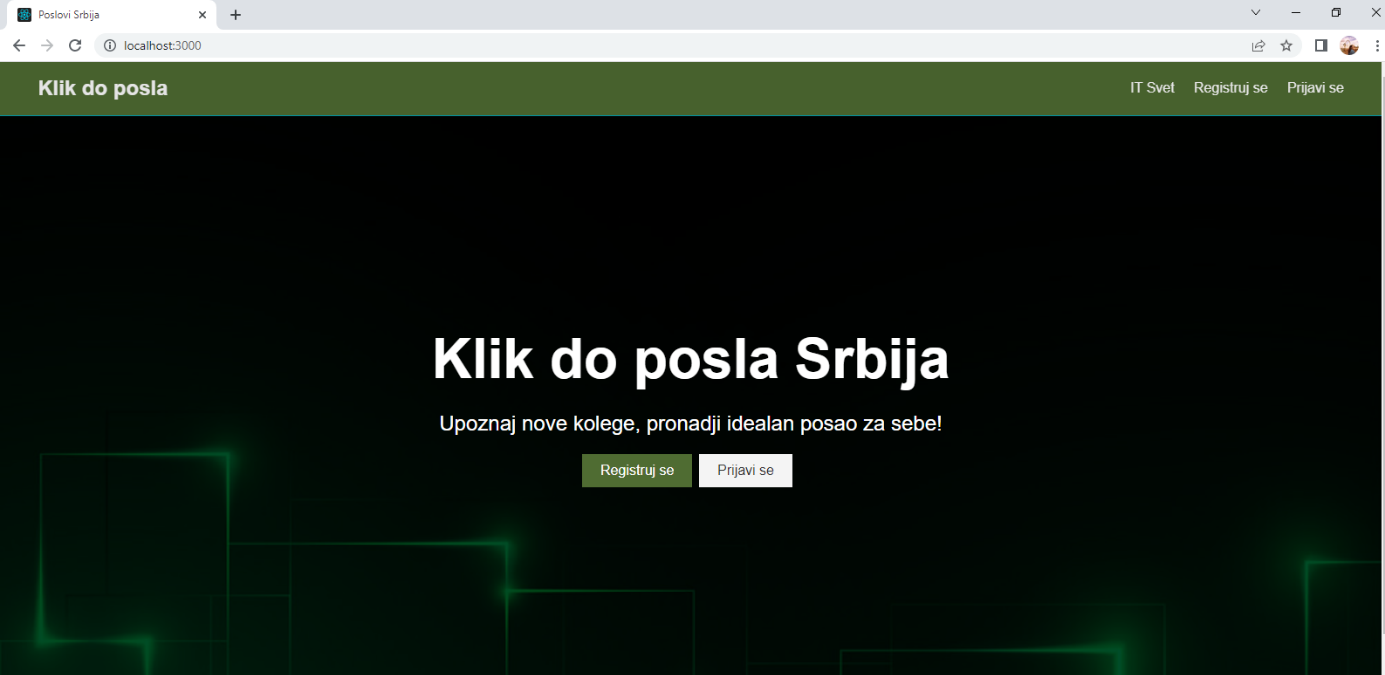
},

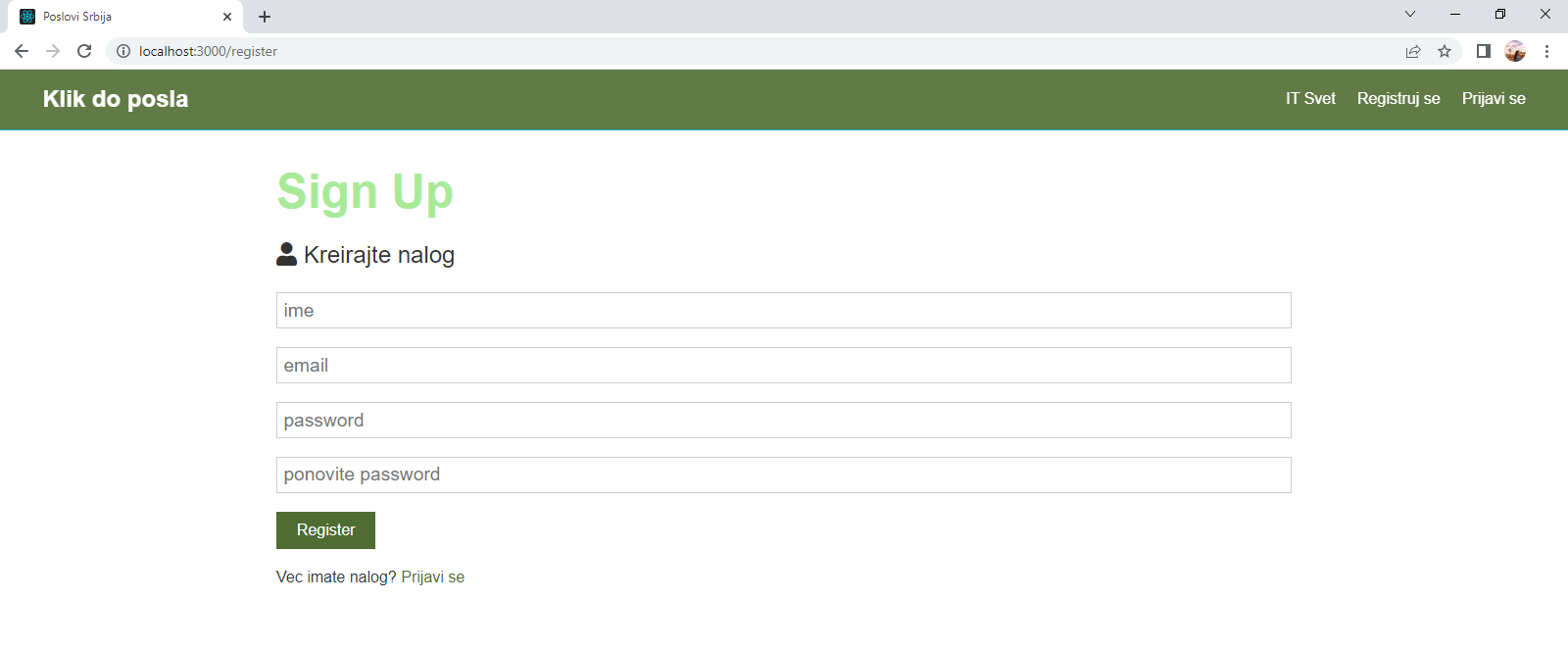
});

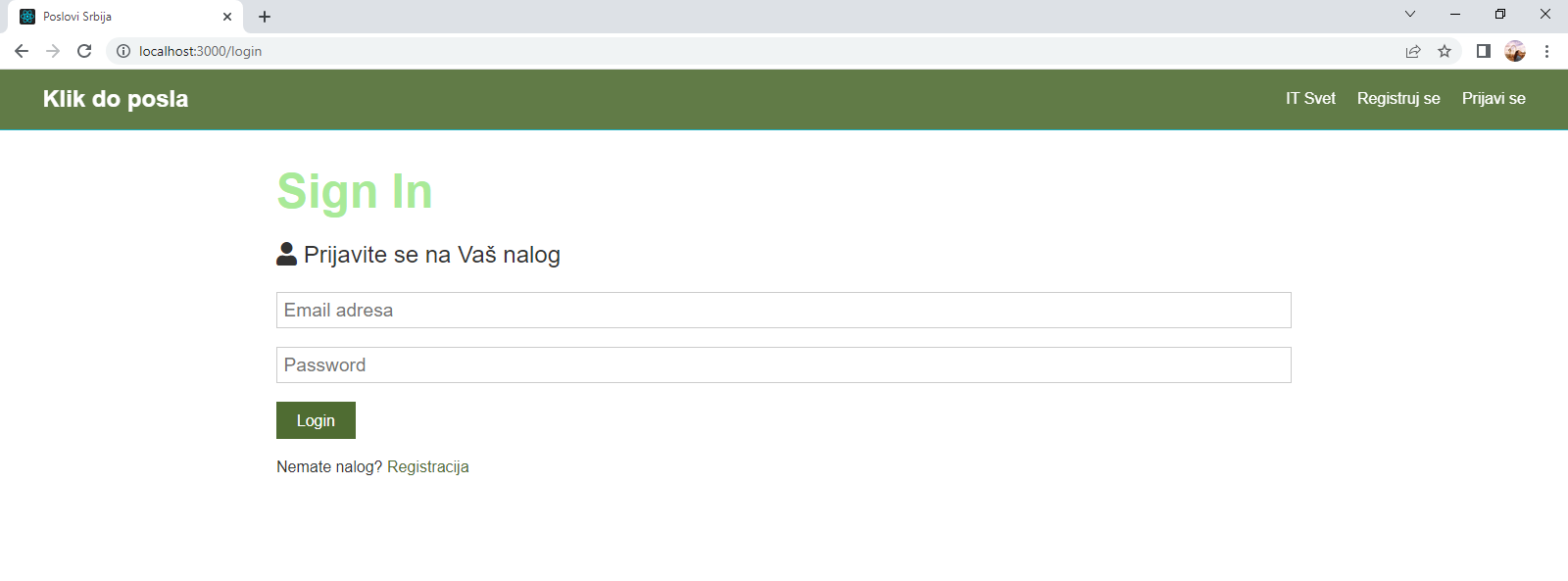
}

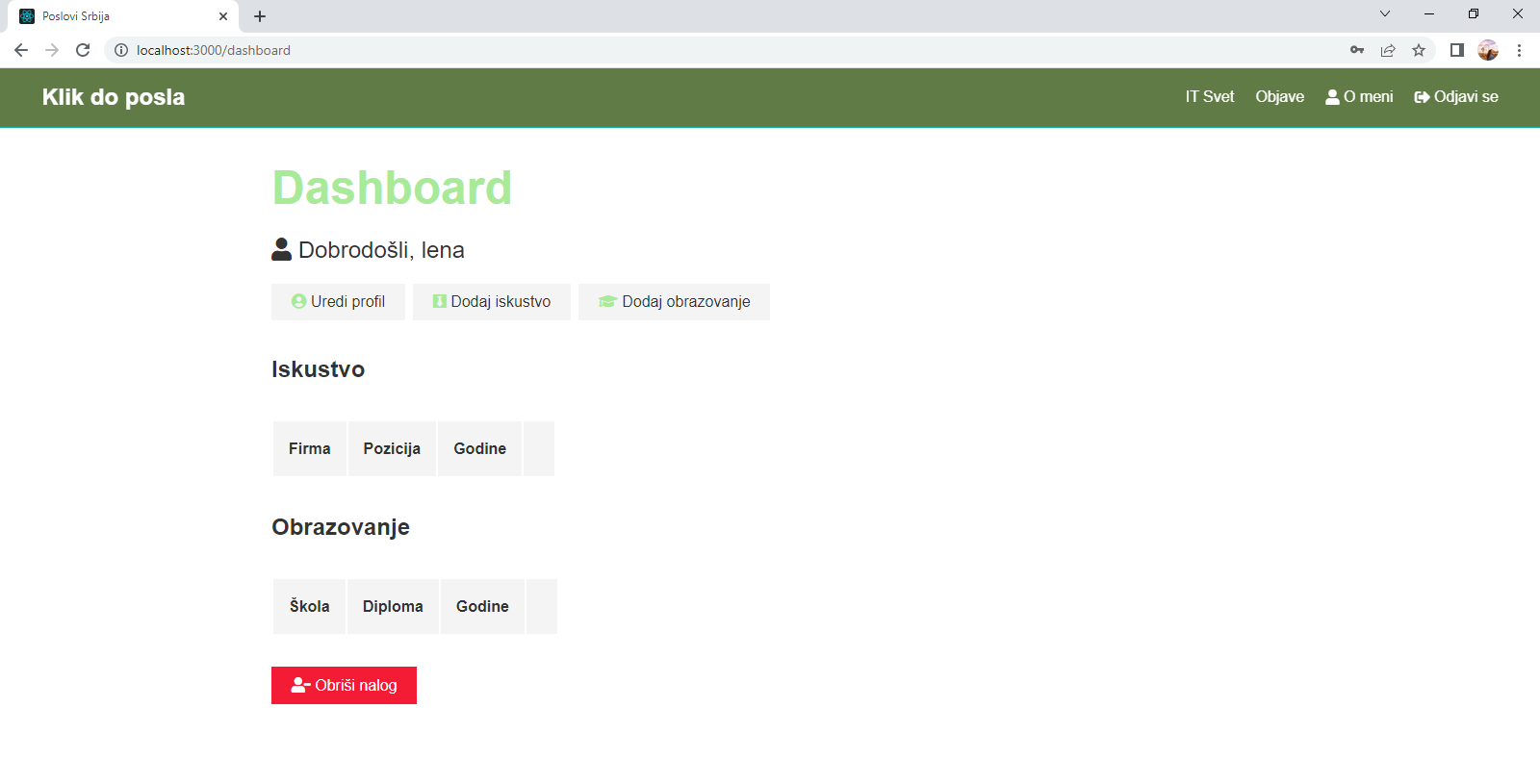
};

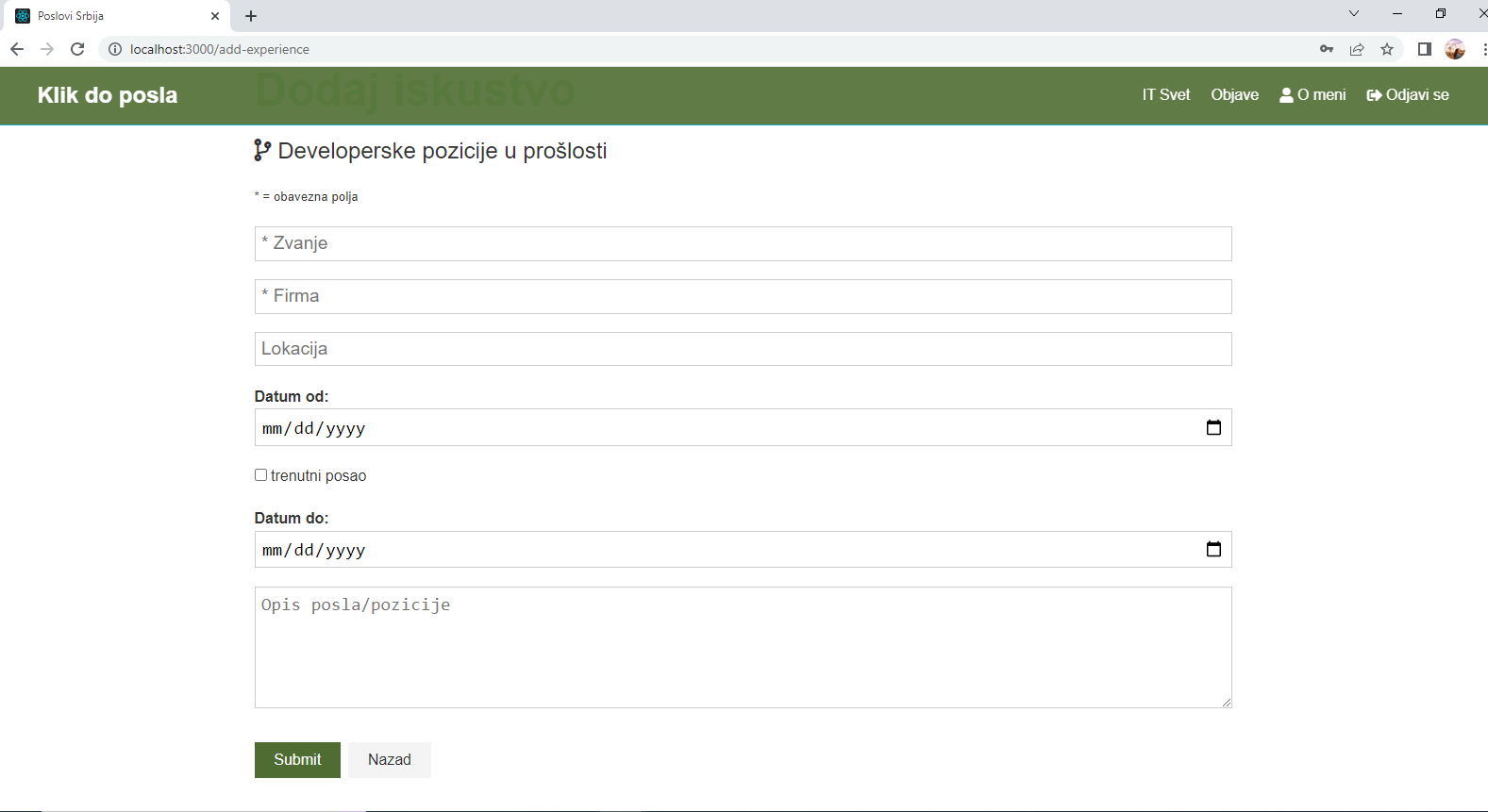
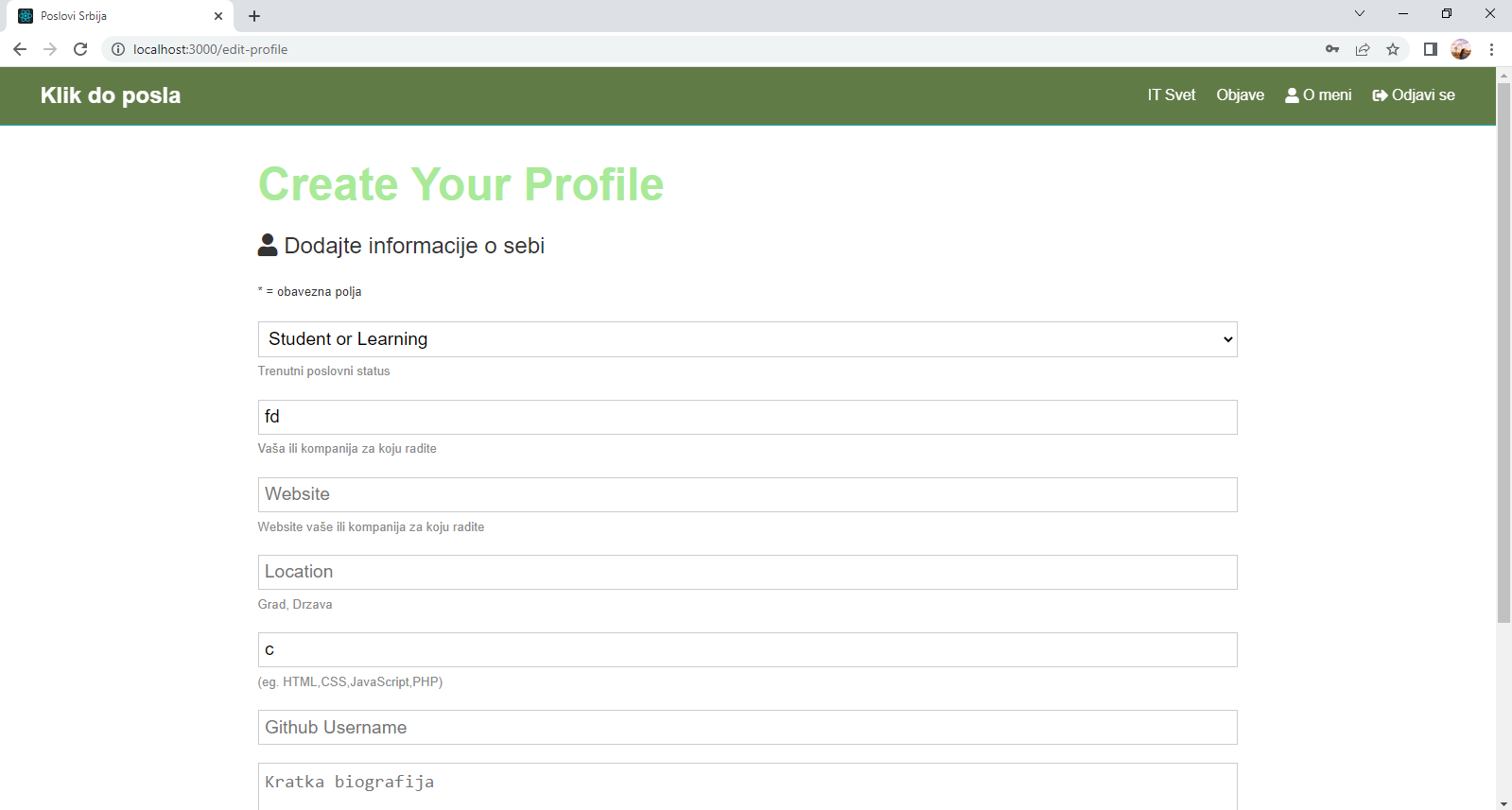
## Korisničko uputstvo

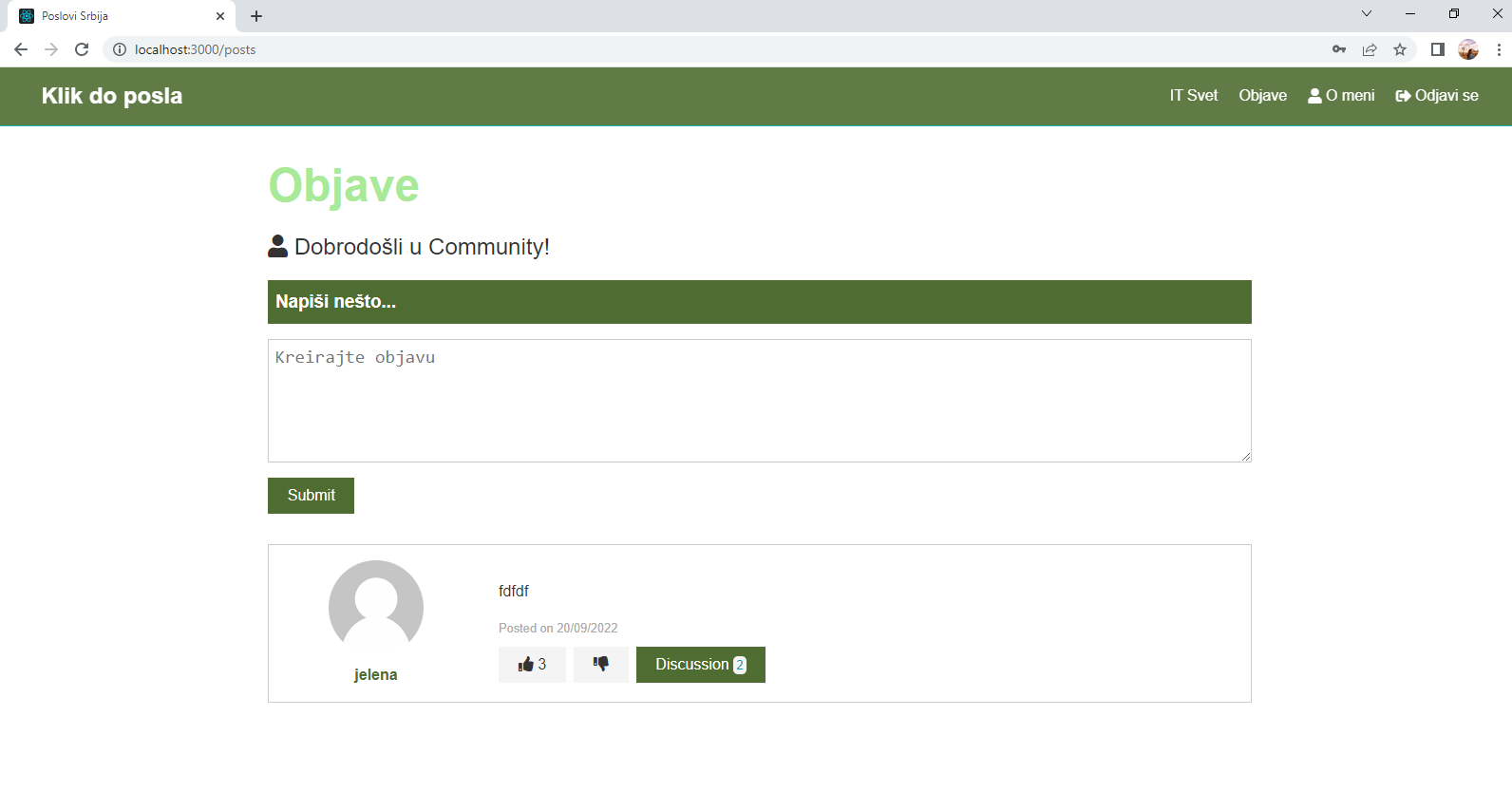
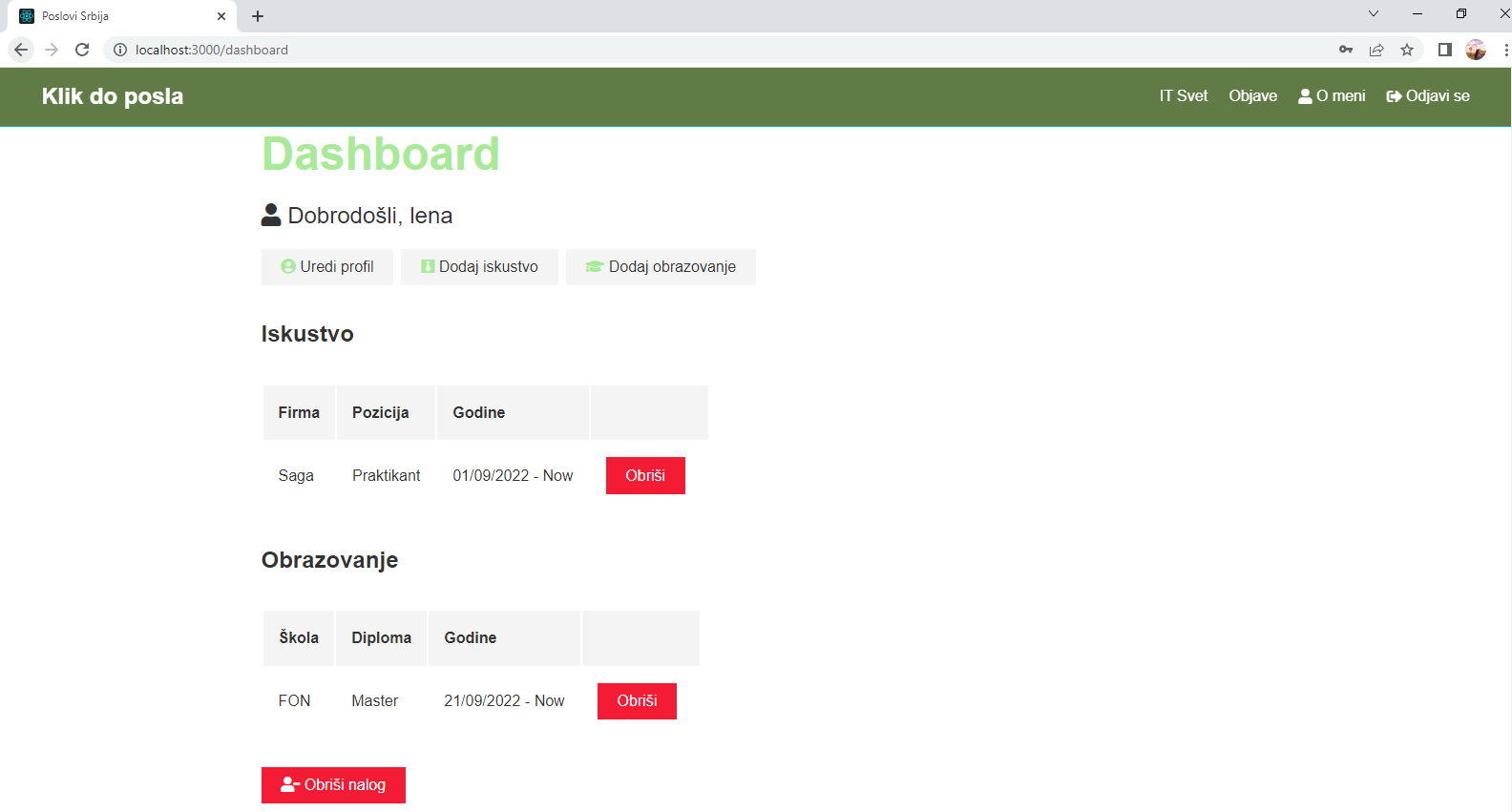


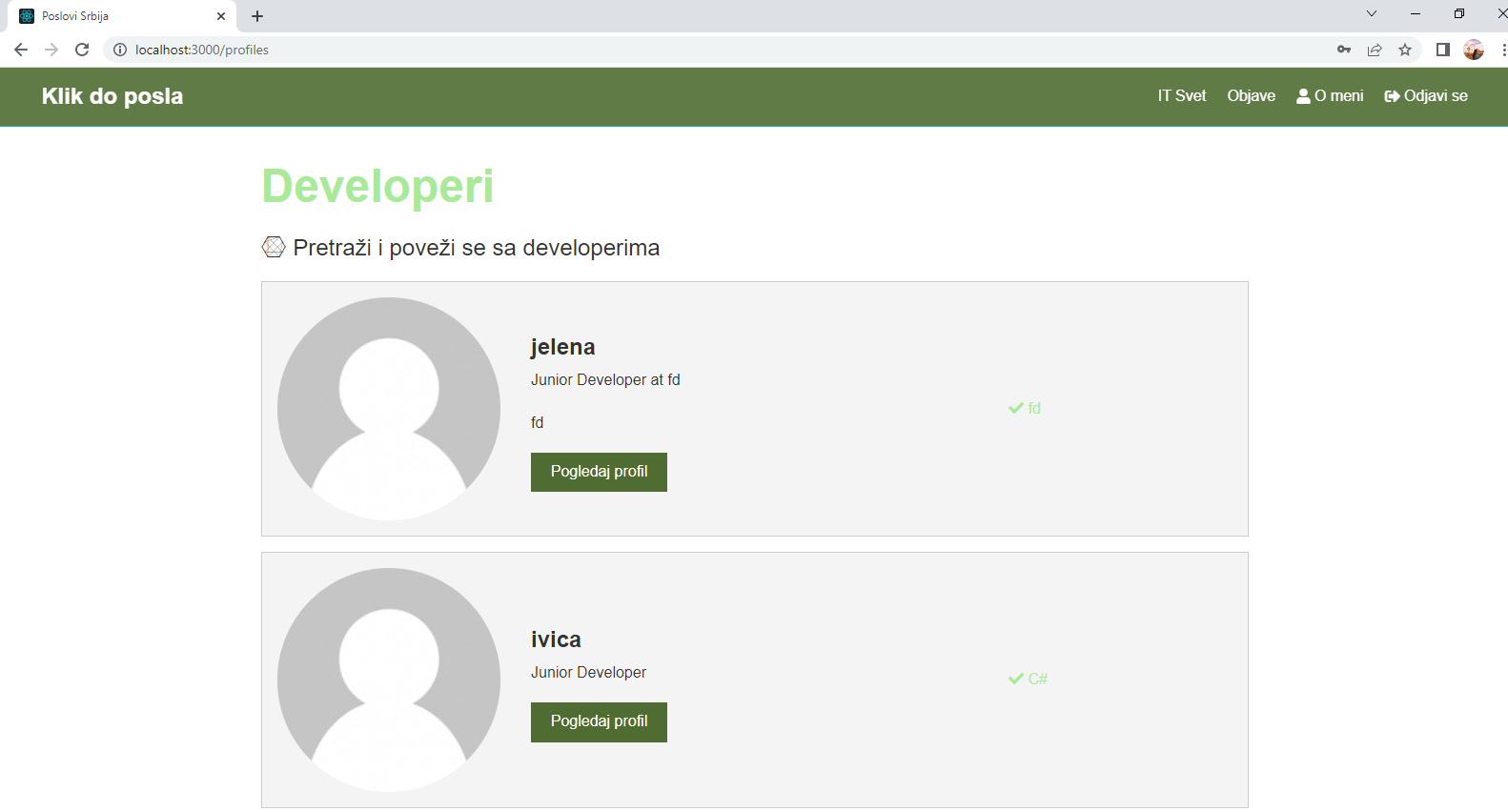
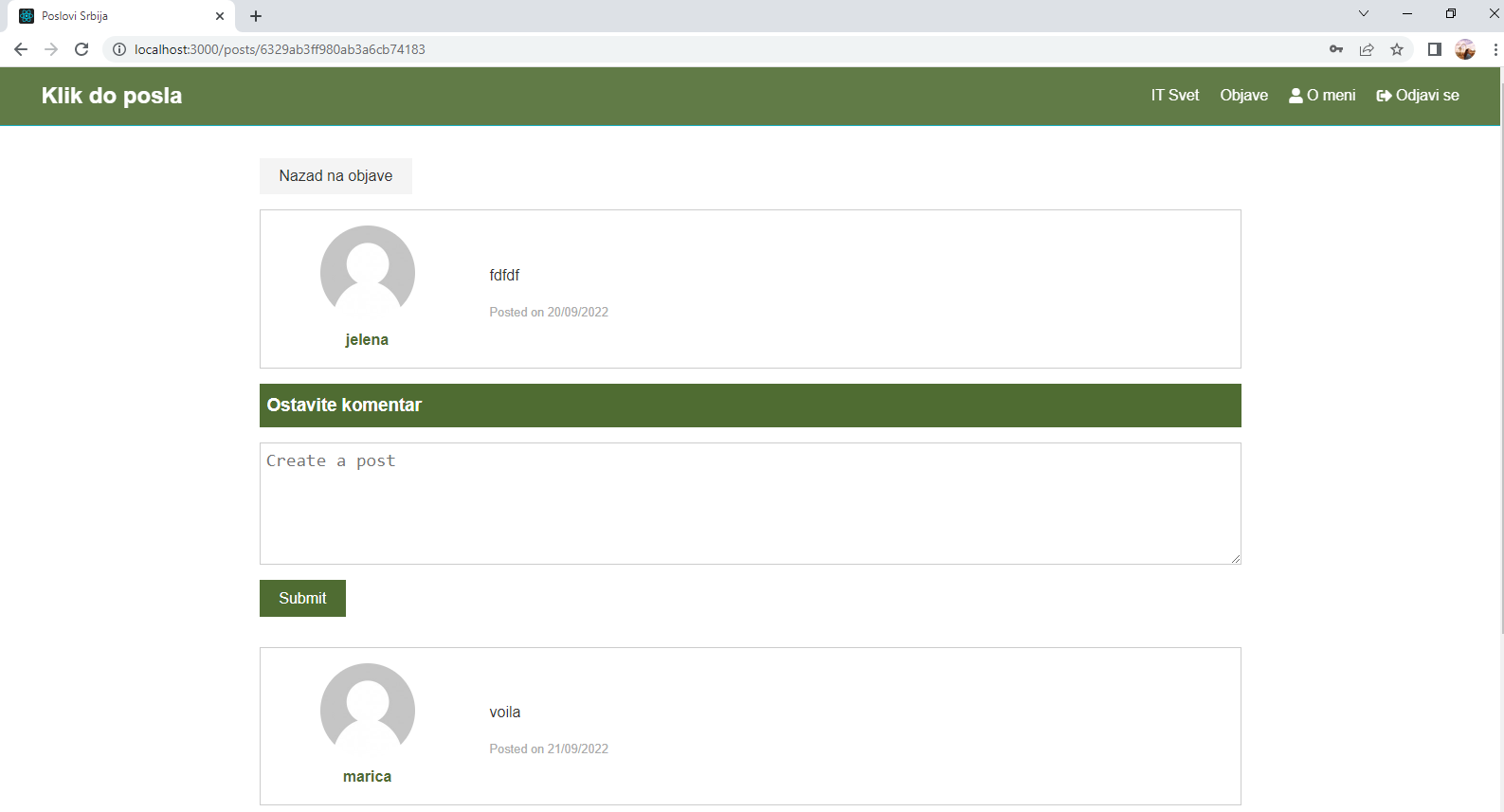


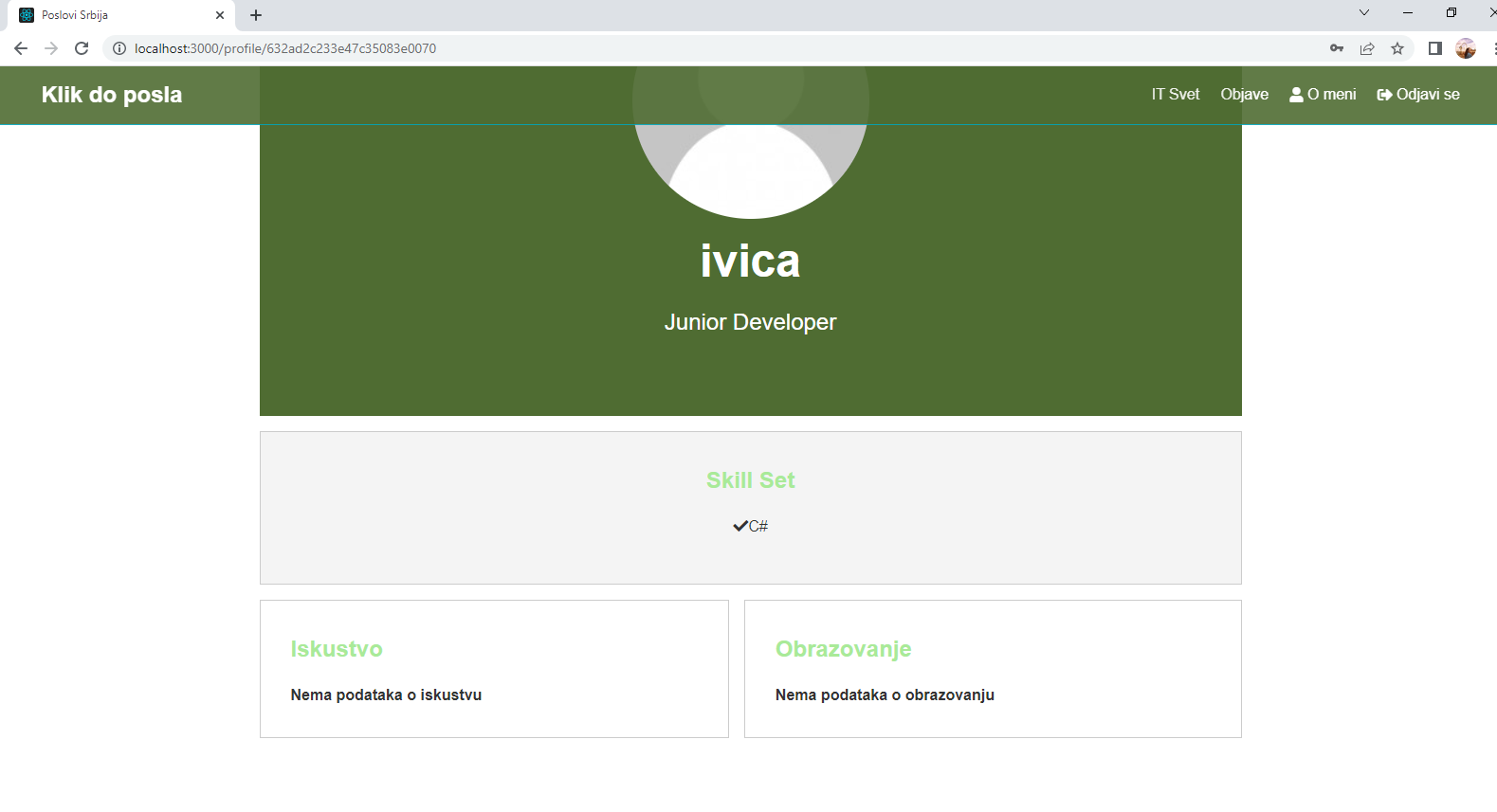
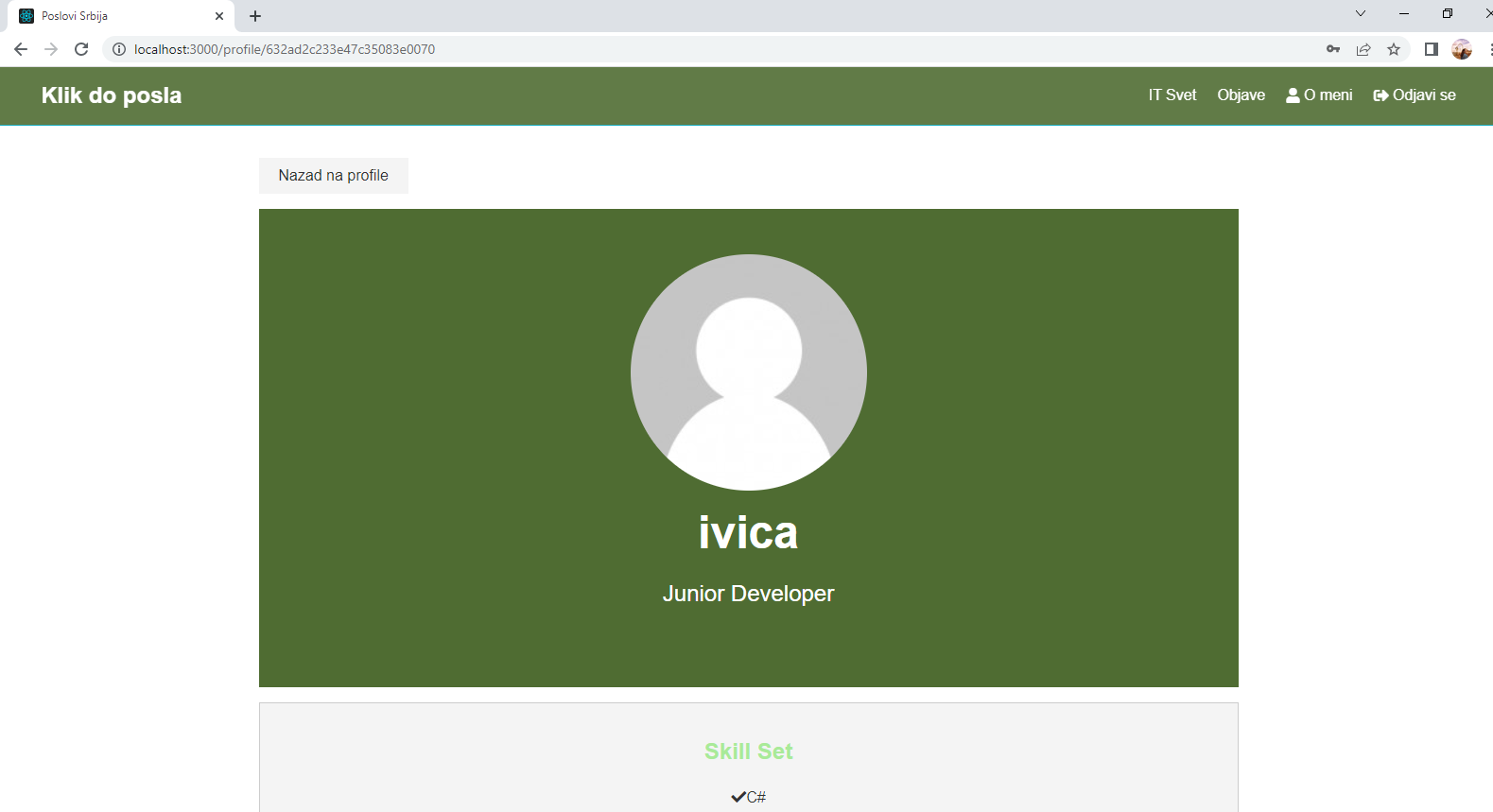












# 6. Zaključak

Aplikacija "Klik do posla" može da se unapredi na nekoliko načina:

* Mogućnost da zaposleni pišu svoja iskustva na određenim pozicijama u kompanijama
* Ocenjivanje kompanija na osnovu uslova rada i zadovoljstva zaposlenih, što će pomoći budućim kandidatima
* Pisanje bloga sa temama koje će pomoći kandidatima u potrazi za poslom.