Web aplikacija za ocjenjivanje kampova u Hrvatskoj

Ostoni, Eric

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Eric Ostoni

Web aplikacija za ocjenjivanje kampova u Hrvatskoj

Završni rad

Pula, 15. rujan 2023 godine

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Eric Ostoni

Web aplikacija za ocjenjivanje kampova u Hrvatskoj

Završni rad

JMB: 0303096688, redoviti student
Studijski smjer: Računarstvo

Predmet: Inženjerska grafika i konstruiranje
Znanstveno područje: Tehničke znanosti
Znanstveno polje: Računarstvo
Mentor: doc. dr. sc. tech. Marko Kršulja

Pula, 15. rujan 2023. godine

Sažetak

U ovome istraživanju prikazan je razvoj online aplikacije za pregled, recenziju i usporedbu kampova u Hrvatskoj. Rad se može podijeliti na teorijski i praktični segment. U teorijskom dijelu rada predstavljeni su alati, tehnologije i programski jezici koji su implementirani u praktičnom dijelu, kao i osnovna razmatranja o potrebi takve aplikacije na tržištu. U svijetu računarstva postoji širok spektar tehnologija za razvoj web aplikacija, a za potrebe ovog rada odabrane su tehnologije koje su se pokazale efikasnima i pouzdanima u praksi.

Praktični dio aplikacije izgrađen je koristeći Node.js za backend, Vue.js kao templating engine, Express.js za kreiranje web servera, te MongoDB za bazom podataka. Osim toga, koristili su se i alati poput JSON web token za autentikaciju korisnika. Ove tehnologije omogućile su efikasnu izgradnju aplikacije koja je skalabilna, sigurna i optimizirana za suvremene web standarde.

Ključne riječi: web aplikacija, Node.js, Express.js, Vue.js, MongoDB, JWT(JSON web token), kampiranje…

Sadržaj

1. Uvod

1.1. Hipoteza

U ovome radu kreirati će se web online aplikacija čiji je cilj da omogući konkurentnost u usporedbi s ostalim web stranicama na tržištu.
Online aplikacija za recenzije i usporedbu kampova u Hrvatskoj je internetska stranica na kojoj će korisnici moći pregledati kampove u Hrvatskoj, vidjeti prosječne ocjene koje su im dodijelili drugi korisnici te saznati cijene. Ova online aplikacija će olakšati planiranje kampiranja putem jednostavnog pregleda svih raspoloživih informacija o pojedinim kampovima na jednom mjestu. Korisnici će moći usporediti ocjene, komentare i preporuke drugih kampera te na temelju toga jednostavnije odabrati kamp koji najviše odgovara njihovim kriterijima i proračunu. Aplikacija će omogućiti i sastavljanje vlastitih recenzija nakon posjeta kampu kako bi i druge kampere informirali o svojem iskustvu. Time će doprinijeti kvalitetnijem odabirom te boljem iskustvu kampiranja za sve korisnike ove online usluge.

1.2. Predmet istraživanja

U ovome radu istražiti će se izrada web stranice za turističke svrhe na primjeru kampiranja. Web aplikacije postale su ključni dio poslovanja u današnjem svijetu. Koristeći online programe, kompanije se mogu mnogo brže razvijati i ostvarivati svoje ciljeve. Koriste se za poboljšanje učinkovitosti, dijeljenje i pružanje informacija. Web aplikacija je interaktivna aplikacija koja radi na skupu poslužitelja na internetu koju korisnici mogu pristupiti koristeći softver preglednika. Mogu im se pristupiti sa bilo kojeg uređaja koji ima pristup internetu, poput stacionarnih i prijenosnih računala, pametnih telefona itd. Obično su kodirani u programskim jezicima koje podržava preglednik kao što su JavaScript i HTML, budući da se ti jezici koriste za prikazivanje izvršnog programa u pregledniku.

1.3. Problem istraživanja

Problem istraživanja je atraktivna i funkcionalna turistička aplikacija za kampiranje, koja treba zadovoljiti zahtjeve tržišta. Tržište je promjenjivo te je potrebno ergonomski postaviti web dizajn koji će odgovarati osobama koje su potencijalno zainteresirane.
Razvoj online programa podrazumijeva utvrđivanje zahtjeva, dizajniranje, izradu, testiranje i implementaciju online programa. Svaki program počinje s idejom koja potencijalno rješava neki problem. Zahtjevi specificiraju svojstva koja su potrebna kako bi se riješio problem. Stoga identificiranjem zahtjeva utvrđuju se funkcionalni i nefunkcionalni zahtjevi online programa te postavljaju ciljevi koje treba ostvariti. Dizajnom se utvrđuje arhitektura sustava, komponente, sučelje i druge karakteristike sustava. Zatim se implementira rješenje korištenjem programskih jezika i alata.

1.4. Ciljevi istraživanja:

- HTML – ispitati mogućnost upotrebe na odabranom primjeru.
- Visual Studio Code - ispitati mogućnost upotrebe na odabranom primjeru.
- CSS - ispitati mogućnost upotrebe na odabranom primjeru.
- Javascript - ispitati mogućnost upotrebe na odabranom primjeru.
- Vue.js - ispitati mogućnost upotrebe na odabranom primjeru.
- Mongo.db - ispitati mogućnost upotrebe na odabranom primjeru.
- Backend.
- Frontend.
- Baza podataka.

1.5. Metodologija istraživanja

Metoda promatranja će se koristiti kako bi se dokumentiralo pisanje programa i arhiviralo verzije i modele koji su predmet ovoga rada. To će se vršiti na način da će se spremati pojedine verzije te opisivati tekstualno što dobro funkcionira a što ne kako bi se mogao pratiti tijek programiranja a samim time i funkcionalnost aplikacije.
Eksperimentalna metoda će se koristiti u obliku izrade aplikacije koja će pružati specificirane usluge a na način da će se provjeriti funkcionalnost aplikacije nakon njene aktivacije tj. probe. Nekoliko proba na različitim sustavima poput chrome, mozila firefox ili Internet explorer i slično biti će uzorkovano kako bi se uočile greške u funkcioniranju.
Metoda modeliranja biti će korištena kako bi se stvorio model pomoću softverskih rješenja koji će pružiti informacije na web aplikaciji kako je to idejno zamišljeno. Pri tome pratiti će se ponašanje i rezultate funkcioniranja aplikacije.

2. Struktura završnog rada

U prvom dijelu rada opisane su tehnologije koje su korištene prilikom razvoja praktičnog dijela: HTML, CSS i JavaScript. Navedeni programski jezici i tehnologije imaju za cilj razvoj interaktivnog sučelja web aplikacije.

HTML služi za semantičko označavanje sadržaja pomoću različitih elemenata poput naslova, paragrafa, formi itd. CSS se koristi za stilizaciju i pozicioniranje elemenata te dizajn korisničkog sučelja. JavaScript omogućuje dodavanje dinamike i interaktivnosti poput validacije unos, asinkronih zahtjeva i prikaza podataka.
Vue.js je JavaScript library korištena za izradu komponentnog i reaktivnog sučelja web aplikacije.

Baza podataka predstavlja mjesto pohrane svih podataka koji se prikupljaju i obrađuju kroz aplikaciju. U ovom radu korištena je NoSQL baza podataka MongoDB zbog fleksibilne strukture podataka.

Na kraju je opisan postupak razvoja same web aplikacije putem prikupljanja zahtjeva, dizajniranja, programiranja i testiranja pojedinih modula i funkcionalnosti. Razvijena je informativna web aplikacija koja omogućuje pregled svih kampova u Hrvatskoj uključujući njihove osnovne podatke, ocjene i cijene.
Opis tehnologija koje su korištene prilikom izrade web aplikacije za ocjenjivanja kampova u Hrvatskoj . U ovom su poglavlju opisani alati, programski jezici i baza podataka koji su se koristili prilikom izrade web aplikacije.

2.1. Programski jezici i alati korišteni tijekom izrade web aplikacije

2.1.1. Visual Studio Code

Na slici 1 prikazan je logo softvera Visual studio Code, VS Code, uređivač je izvornog koda koji je napravio Microsoft s Electron Frameworkom za Windows, Linux i macOS. Značajke uključuju podršku za otklanjanje pogrešaka, isticanje sintakse, inteligentno dovršavanje koda, isječke, refaktoriranje koda i ugrađeni Git.

Slika 1. Logo Visual Studio Code

Izvor: https://logowik.com/visual-studio-code-vector-logo-1-5273.html

Visual Studio Code je besplatan uređivač koda otvorenog koda razvijen od strane Microsofta. Izdvaja se kao jedan od najpopularnijih IDE alata za razvojne inženjere zahvaljujući svojim naprednim funkcijama i prilagodljivosti.

VSC ima sučelje nalik Notepadu sa izvlačivim lateralnim panelima za istraživanje datoteka, kontrolu izvora i terminal. Nedostaje mu složenije promjene poput razvojnih okruženja kao što su Eclipse ili Visual Studio, ali zauzvrat nudi laku upotrebljivost i nisku potrošnju resursa.

Ključne mogućnosti uključuju:

- Podrška za više od 60 programskih jezika uz dodatke za sintaksu i precizno obojen tekst.
- Live Share za kolektivnu suradnju nad istim kodom u stvarnom vremenu.
- Integracija sa Gitom, Mercurialom i drugim sustavima verzija.
- Debugging alati kao što su breakpointovi, praćenje promjena itd.
- Automatska dopuna koda, refaktoring i ostale inteligentne funkcije.
- Terminal unutar prozora za rad sa shell komandama.
- Otvorena arhitektura koja omogućuje stvaranje vlastitih dodataka.
- Dostupnost za Windows, MacOS i Linux.

VSC je prilagođen za rad na velikom broju programskih okvira i tehnologija poput .NET-a, Node.js-a, Pythona, PHP-ja, Javascripta i drugih. Također podržava i kontekstualni pregled JSON, XML i drugih datoteka.

2.1.2. HTLM

HTML (Hypertext Markup Language) je prezentacijski jezik za izradu web stranica i predstavlja skraćenicu za hipertekstualni prezentacijski jezik. Koncepciju HTML-a i prvu verziju izradio je Tim Berners-Lee 1991. godine dok je standardizirani HTML 2.0 objavljen 1995.

HTML koristi oznake koje označavaju strukturu i semantiku sadržaja poput naslova, paragrafa, slika itd. Osnovnu strukturu HTML dokumenta čine deklaracija dokumenta <!DOCTYPE>, roditeljski element <html>, element <head> za metapodatke i element <body> za vidljivi sadržaj.

U elementu <head> se nalaze oznake poput <title> za naslov koji se prikazuje u pregledniku, <meta> za opise stranice, <link> i <script> za povezivanje CSS/JS datoteka. Element <body> sadrži vidljivi sadržaj stranice kroz različite oznake poput <h1-6> za naslove, <p> za tekst, <img> za slike itd.

HTML opisuje strukturu dokumenata kroz elemente i atribute, ne opisuje izgled koji se specificira CSS-om. Dinamiku i ponašanje stranica dodaju skripte kao JavaScript.

Prednosti HTML-a su jednostavnost, neovisnost o platformi, podrška svih preglednika, integracija s CSS i JS. Nedostatak je pružanje samo statičkih stranica, te zahtjevan je ručni razvoj dinamičkih. Međutim, i dalje predstavlja osnovu Web-a i sabirnicu različitih tehnologija.

2.1.3. CSS

CSS (Cascading Style Sheets) je jezik za stilizaciju izgleda web stranica definiranjem formata, boja, rasporeda i dr. CSS pruža mogućnost dizajniranja stranice neovisno o strukturnom jeziku HTML. CSS opisuje primjenu stilova na određene elemente stranice kroz pravila stilizacije. Pravila se sastoje od selektora koji označava element i bloka deklaracija stilova. Deklaracije upisuju ime stilskog svojstva i vrijednost odvojene dvotočkom.

Stilovi olakšavaju održavanje stranice jer se promjene vrše na jednom mjestu umjesto unutar svakog HTML elementa. Također omogućuju kreiranje konzistentnog dizajna na više stranica.

Spominju različite vrste selektora za odabir elemenata na koje se primjenjuje stil - po imenu, ID-u, klasi, atributu elementa itd. To omogućuje fleksibilnu i specifičnu stilizaciju.

Prednosti CSS-a su jednostavno održavanje dizajna, izvanmrežno korištenje, širok skup stilskih svojstava i podrška svih preglednika. CSS razdjeljuje semantiku sadržaja definiranu HTML-om od njegovog prikaza što olakšava razvoj pristupačnih web stranica.

2.1.4. Javascript

JavaScript je programski jezik dizajniran za dodavanje interaktivnosti i dinamičnih elemenata na web stranicama. Može se izvršavati u svim modernim web preglednicima i drugim okruženjima bez preglednika, te koristiti za razvoj frontend i backend aplikacija.

Jezični konstrukti poput promjenjivih, operatora, kontrolnih struktura i funkcija omogućuju imperativno programiranje kojim se definira tok izvršavanja skripte. Također podržava deklarativni stil pisanjem funkcija kao objekata.

Jedna od glavnih značajki je manipulacija DOM-om za uređivanje dinamičkih sadržaja stranice nakon učitavanja. To uključuje dodavanje/uklanjanje elemenata, dodavanje obrada događaja i sl. Ugrađene biblioteke olakšavaju rad s raznim formatima podataka.

Na serverskoj strani omogućuje izradu aplikacija pomoću okruženja kao Node.js. To uključuje rad s bazama, obradu zahtjeva, generiranje dinamičkih stranica i sl.

Popularne biblioteke za razvoj su React, Angular i Vue za izradu kompleksnijih SPA i mobilnih aplikacija. Također se koristi u drugim područjima poput IoT, igara, umjetne inteligencije.

Performanse izvođenja su nešto lošije od statički tipiziranih jezika, ali se optimizacijama u novijim implementacijama značajno poboljšale. Slabija tipizacija olakšava razvoj ali može dovesti do grešaka.

Na slici 2 prikazan je logo softvera JavaScript, on predstavlja jezik široke primjene zahvaljujući interaktivnosti, jednostavnosti i potpori svim vodećim platformama. Koristi se kako u razvoju frontenda tako i složenijih aplikativnih rješenja.

Slika 2. JavaScript logo

Izvor: https://upload.wikimedia.org/wikipedia/commons/9/99/Unofficial_JavaScript_logo_2.svg

2.1.5. Vue.js

Vue.js predstavlja napredni i prilagodljivi JavaScript framework za izradu jednostraničnih aplikacija i interaktivnih web stranica. Omogućava stvaranje složenih komponentno orijentiranih UI-jeva na jednostavan i intuitivan način.
Glavne karakteristike Vue.js-a uključuju reaktivne promjenjive, komponente, prijelaze, dinamički učitavanje podataka, enkapsulaciju podataka i metode te laganu učenicu krivulju zahvaljujući dobro dokumentiranoj biblioteci.
Instalacija se može obaviti uz pomoć CDN-a za brzu upotrebu u HTML stranicama, kao i putem NPM-a za razvoj složenijih aplikacija. Vue CLI olakšava stvaranje novih projekata sa svim potrebnim konfiguracijama i alatima.
Vue.js je kompatibilan s nizom drugih biblioteka i tehnologija poput Reacta, Angulara, Node.js-a i dr. Omogućuje interoperabilnost kroz ugrađene komponente, directive i sistem modula. Podržava sve vodeće preglednike.
Prednosti Vue.js-a su:

- Prilagodljivost različitim projektima,
- Moduli koji omogućuju kreiranje zasebnih dijelova aplikacije,
- Prijelazi između modula,
- Detaljne upute koje olakšavaju učenje i korištenje.

Vue.js omogućuje fleksibilnu strukturu i jednostavnu migraciju. Moduli Vue.js-a pomažu u oblikovanju prilagođenih dijelova koji se mogu ponovno iskoristiti u HTML-u. On nudi različite načine primjene prijelaza između modula kada su uključeni ili isključeni iz izgleda stranice. Vue.js daje lagani početak putem koraka po korak uputa.
Postoje tri načina instalacije Vue.js-a:

- Datoteka spremljena na CDN-u,
- korištenjem NPM-a.
- alata za stvaranje projekata (CLI).

2.1.6. Mongo db

MongoDB je vodeća NoSQL baza podataka dizajnirana da se nosi s velikim količinama podataka, pružajući visoku dostupnost, skalabilnost i fleksibilnost. Za razliku od tradicionalnih relacijskih baza podataka, MongoDB koristi dokumente u JSON-formatu (BSON unutar baze), što omogućava bržu i fleksibilniju obradu podataka.
Karakteristike:

- Dokumentno orijentirana baza podataka: Umjesto tradicionalnih tablica i redaka, MongoDB koristi kolekcije i dokumente. Dokumenti su slični objektima u programskim jezicima i omogućuju ugrađivanje podataka u nizove i druge dokumente.
- Skalabilnost: MongoDB je dizajniran da bude vrlo skalabilan putem horizontalnog širenja. To se postiže dodavanjem više servera u MongoDB klaster, omogućujući bazi da raste kako se povećava potražnja.
- Visoka dostupnost: S mogućnošću konfiguracije replikacijskih setova, MongoDB pruža visoku dostupnost podataka. Ako jedan čvor (server) postane nedostupan, upiti se mogu automatski preusmjeriti na drugi dostupan čvor.
- Dinamička shema: Za razliku od relacijskih baza podataka koje zahtijevaju definiranu shemu, MongoDB omogućava kolekcijama da imaju dokumente s različitim setom polja.
- Podrška za ad-hoc upite: MongoDB podržava širok spektar upita, uključujući ad-hoc, indeksiranje, regularne izraze i agregacije.

MongoDB je NoSQL baza podataka, dok su SQL baze tradicionalne relacijske baze podataka.
Glavna razlika je u načinu spremanja i pristupa podacima. MongoDB je dokumentno orijentirana baza koja sprema podatke u BSON (Binary JSON) dokumente unutar kolekcija. SQL baze koriste sheme s tabelama, redcima i stupcima.
MongoDB je fleksibilnija jer ne zahtijeva fiksnu shemu kao SQL. Podaci mogu varirati u strukturi unutar iste kolekcije, što je korisno za nestrukturirane podatke. SQL baze podatke moraju strogo prilagoditi shemi.
Upiti su također različiti. MongoDB koristi JSON sintaksu za upite dokumenta i

indeksiranje, dok SQL koristi standardni jezik upita (SQL).
MongoDB nudi nižu latenciju za unos i čitanje podataka zahvaljujući manjoj složenosti operacija u odnosu na transakcije SQL-a. Međutim, transakcije u MongoDB-u nisu podržane.
Pristup podacima je drugačiji - MongoDB vraća dokumente, dok SQL vraća tablice redova. MongoDB također podržava dinamičko dodavanje novih polja unutar iste kolekcije.
Skalabilnost je jedna od glavnih prednosti NoSQL baza poput MongoDB-a, čija se kolekcija može razdijeliti na više čvorova. SQL baze zahtijevaju složeniju konfiguraciju za skaliranje.

3. Opis elemenata i funkcionalnosti web aplikacije

U ovom poglavlju će se opisati praktični dio završnog rada, tj. opisati će se postupak izrade web aplikacija za ocjenjivanje kampova u Hrvatskoj.

Link na web stranicu: https://infocamps.onrender.com
Link na GitHub projekta: https://github.com/EricOstoni/Zavrsni-rad

3.1. Backend

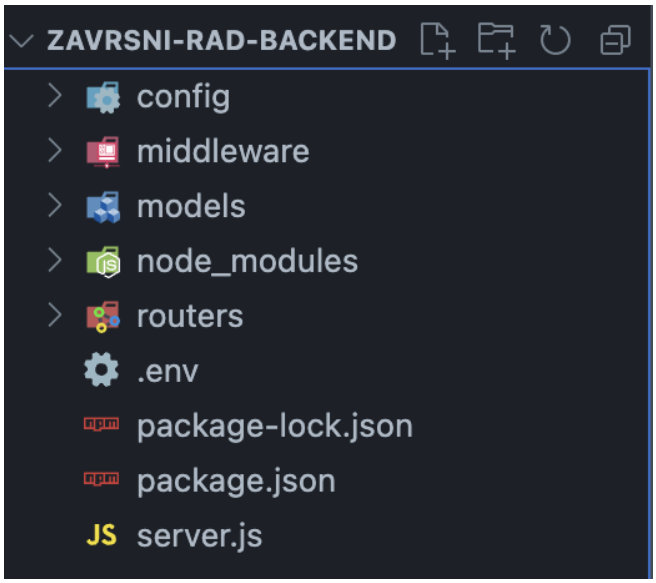
3.1.1. Uvod u backend

Backend predstavlja poslužiteljski (server-side) dio web aplikacije koji se nalazi između klijentskog (frontend) dijela i baze podataka. Njegova uloga je da pruža podatkovne i poslovne logike aplikacije te komunikaciju s bazom podataka i korisničkim sučeljem.
Kod web aplikacije za ocjenjivanje kampova u Hrvatskoj, backend je implementiran pomoću Node.js okvirne i Express.js biblioteke za razvoj web aplikacija uz MongoDB bazu podataka.

3.1.2. Struktura backend-a za predmetnu aplikaciju

U ovome radu koristiti će se backend kao što je prikazano na slici 3.

Slika 3. Struktura backend-a



Izvor : obrada autora

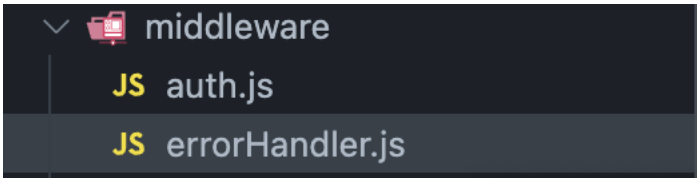
Backend je strukturiran u sljedeće glavne dijelove:

- config - Sadrži konfiguracijske datoteke za bazu podataka
- middleware - Sadrži aplikacijske middleware rutine kao što je autentifikacija
- models - Definira Mongoose modele za entitete pohranjene u bazi
- routers - Sadrži definirane rute za pojedine CRUD operacije
- .env - Datoteka s osjetljivim varijablama okoline
- server.js - Glavna inicijalizacijska točka poslužitelja.

3.1.3. Detaljni opis dijelova strukture

Prvo što se opisuje je middleware folder u kojem se nalaze rute za validaciju, autorizaciju i druge primjene poput logiranja. Kao što se vidi na slici (slika 4.) middleware folder se sastoji od „auth.js“ i „errorHandler.js“ datoteka.

Slika 4. Struktura middleware folder-a



Izvor : obrada autora

Auth.js je datoteka koja sadrži middleware za provjeru autentičnosti korisnika (authorization). Koristi se JWT (JSON Web Token) za potvrdu identiteta korisnika.
JSON web token je otvoreni standard za prijenos podataka putem tokena. Token sadrži JSON objekt podijeljen na header, payload i signature dijelove. Header sadrži tip i algoritam za potpis, payload korisničke podatke poput ID-a, a signature služi za provjeru autentičnosti i cjelovitosti. Digitalni potpis čini token sigurnim te onemogućuje falsificiranje bez poznavanja tajnog ključa korištenog za njegovo potpisivanje.

Slika 5. Struktura auth.js

```
const jwt = require("jsonwebtoken");
const { jwtSecret } = require("../config/auth");
const User = require("../models/User");

module.exports = async function (req, res, next) {
  // dobivanje tokena iz header-a
  const token = req.header("x-auth-token");
  // provjera ako nije token
  if (!token) {
    return res.status(401).json({ msg: "No token, authorization denied" });
  }
  // provjera token-a
  try {
    const decoded = jwt.verify(token, jwtSecret);
    // provjera ako korisnik postoji u bazi
    const user = await User.findById(decoded.user.id);
    if (!user) {
      return res
        .status(401)
        .json({ msg: "Token is not valid: user does not exist" });
    }
    req.user = { id: decoded.user.id };
    next();
  } catch (err) {
    res.status(401).json({ msg: "Token is not valid" });
  }
};
```

Izvor : obrada autora

Auth.js je strukturiran tako da prvo dohvaća token iz zaglavlja zahtjeva (header), zatim se token dekodira korištenjem „jwtSecret“ varijable iz config/auth.js datoteke. Token sadrži informaciju o id-u korisnika (user id).

Zatim se preko Mongoose modela provjerava postoji li korisnik s tim id-jem u bazi podataka. Ako postoji, id korisnika se sprema u „req.user“ objekt i prosljeđuje se u daljnje obrade.

U suprotnom se vraća „401 Unauthorized status“ kodom uz odgovarajuću poruku kao što se vidi na slici (slika 5.).

Osim „auth.js“ u middleware strukturi imamo i „errorHandler.js“. ErrorHandler.js je opći middleware za obradu pogrešaka. Ukoliko dođe do pogreške tijekom obrade zahtjeva, logira se u konzolu zajedno s porukom o pogrešci i vraća se „500 Internal Server Error“ status s porukom o pogrešci (slika 6.).

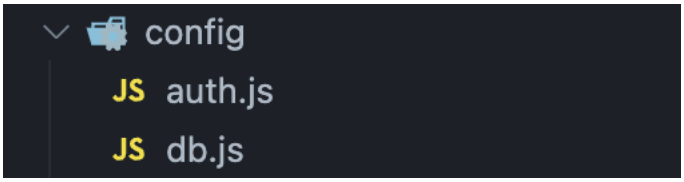
Slika 6. Struktura "errorHandler.js"

```
module.exports = function(err, req, res, next) {
    console.error(err.stack);
    res.status(500).send('Something went wrong!');
};
```

Izvor : obrada autora

Drugo što se opisuje je struktura config folder-a koji sadrži konfiguracijske datoteke programa. Config folder se sastoji od „auth.js“ i „db.js“ datoteke (slika 7.).

Slika 7. Struktura config folder-a



Izvor : obrada autora

Auth.js sadrži tajni ključ koji se koristi za generiranje JWT potpisa. Taj se ključ koristi prilikom provjere potpisa token-a. Zbog sigurnosti taj ključ je tajan i nije u pristupačnosti javnosti (slika 8.).

Slika 8. Struktura "auth.js", tajni ključ

```
// config/auth.js
module.exports = {
    jwtSecret: '0409key'
};
```

Izvor : obrada autora

Db.js datoteka sadrži konfiguraciju za povezivanje na MongoDB bazu podataka pomoću Mongoose ORM biblioteke. Mongoose omogućava definiranje sheme i modela podataka te komunikaciju s bazom kroz metode za upite, dodavanje, izmjenu i brisanje dokumenata. Funkcija connectDB prilikom pokretanja aplikacije poziva se „connect“ metoda mongoosea. Tu se prosljeđuje URL baze koji se uzima iz „.env“ varijable kako bi se osjetljivi podaci čuvali van repozitorija.

Također se definiraju opcije za povezivanje poput novijeg protokola (useNewUrlParser) i topologije (useUnifiedTopology) radi podrške najnovijim verzijama baze.

U slučaju pogreške prilikom spajanja, ispisuje se poruka o grešci u konzolu radi lakšeg otklanjanja problema. Proces se zatim prekida radi sprječavanja daljnje neispravne izvođenje koda (slika 9.).

Slika 9. Struktura "db.js" datoteke, povezivanje na bazu podataka

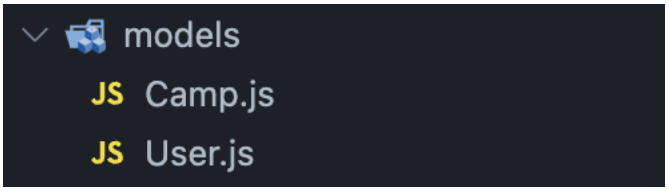
```
const mongoose = require("mongoose");

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log("MongoDB connected...");
  } catch (err) {
    console.error(err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Izvor : obrada autora

Ovim datotekama definira se tajni ključ za JWT i način spajanja na bazu - dvije osnovne konfiguracijske stavke backend sustava. Oba dvije datoteke nisu vidljive korisnicima.

Treće što se opisuje je struktura datoteke models. Models datoteka sadrži Mongoose modele koji definiraju shemu podataka pohranjenih u kolekcijama baze podataka. Sastoji se od dvije sheme a to su „User.js“ i „Camp.js“ (slika 10.).

Slika 10. Struktura models datoteke



Izvor : obrada autora

Datoteka „Camps.js“ služi za definiranje Mongoose sheme za dokumente kampova u bazi podataka MongoDB. Najprije se inicijaliziraju potrebne biblioteke poput „mongoose“ koja omogućava definiranje shema i modela. Zatim se stvara nova instanca mongoose.Schema koja predstavlja osnovnu strukturu za dokument kampa. Unutar objekta koji se prosljeđuje „Schemi“ definiraju se polja dokumenta poput imena, slike, cijene i lokacije kampa. Također se definiraju polja za ocjene i komentare koja sadrže reference na korisnike. Potom se funkcijom „mongoose.model“ iz definirane sheme „CampSchema“ stvara novi „Model“ pod nazivom „Camp“. Ovaj se model može koristiti u drugim datotekama za izvođenje operacija nad kampovima u bazi podataka, kao što su kreiranje, čitanje, ažuriranje i brisanje kampova. Shema definira strukturu dokumenata kampova, dok model omogućava programski pristup toj vrsti podataka u bazi. Na ovaj način datotekom Camps.js modelira se struktura dokumenata kampova koja će se koristiti u bazi podataka MongoDB (slika 11.).

Slika 11. Struktura Camp.js datoteke

```
const mongoose = require("mongoose");
const CampSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  picture: {
    type: String,
    default: "",
  },
  price: {
    type: Number,
    required: true,
  },
  location: {
    type: String,
    required: true,
  },
  ratings: [
    {
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
      score: {
        type: Number,
        min: 1,
        max: 5,
      },
    },
  ],
  comments: [
    {
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
      text: String,
    },
  ],
});
module.exports = mongoose.model("Camp", CampSchema);
```

Izvor : obrada autora

Datoteka User.js služi za definiranje sheme korisničkih dokumenata u bazi podataka. Prvo se uvode potrebne biblioteke – „mongoose“ za rad s bazom i „bcryptjs“ za šifriranje lozinki. Zatim se pomoću „mongoose.Schema“ definira struktura dokumenata kroz polja poput korisničkog imena, lozinke, e-maila i ostalih podataka. Polje „username“ je jedinstveno, dok su password i email obavezna polja.

Shema također sadrži nepotpisana polja poput imena i prezimena te avatara korisnika.

Slika 12. Struktura User.js datoteke

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  firstName: {
    type: String,
    required: false,
  },
  lastName: {
    type: String,
    required: false,
  },
  avatar: {
    type: String,
    required: false,
  },
});
// Hash lozinke prije spremanja u bazi
userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    return next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
module.exports = mongoose.model("User", userSchema);
```

Izvor : obrada autora

Posebno je važno spomenuti kako se kod spremanja novog korisnika ili ažuriranja lozinke, pomoću „pre(save)“ hooka šifrira lozinka prije upisa u bazu. Koristi se bcryptjs biblioteka za generiranje „salt“ zapreke i šifriranje unesene lozinke.

Slika 13. Funkcija za hash-iranje lozinke prije spemanja u bazi

```
// Hash lozinke prije spremanja u bazi
userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    return next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
module.exports = mongoose.model("User", userSchema);
```

Izvor : obrada autora

Taj dio koda služi za šifriranje lozinki korisnika prije njihovog spremanja u bazu podataka. Koristi se „mongoose hook“ „.pre('save')" koji se izvršava prije svakog spremanja ili ažuriranja korisničkog dokumenta.

Unutar hooka prvo se provjerava je li lozinka uopće promijenjena putem „.isModified('password')“. Koristi se bcrypt biblioteka za generiranje slučajne „salt“ vrijednosti koja će poslužiti za šifriranje. Vrijednost generiranog salt-a se sprema u promjenjivu „salt“. Zatim se uz pomoć bcrypt funkcije „.hash()" uzima trenutna lozinka iz sheme i šifrira koristeći prije generirani „salt“.

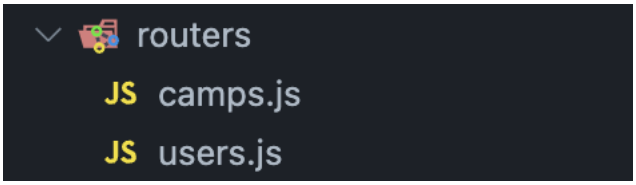
Rezultat šifrirane lozinke piše natrag u shemu umjesto tekst lozinke.

Na kraju se poziva next() kako bi se nastavio proces spremanja korisnika u bazu, ali sa zaštićenom lozinkom. Ovim pristupom osigurava se da su sve lozinke u bazi čuvanje u sigurnom hashed obliku (slika 13.).

Nakon definiranja sheme, pomoću „mongoose.model“ iz iste se stvara Model pod imenom „User". Ovime je modelom osnovna struktura korisničkih dokumenata u skladu s potrebama za prijavu i autorizaciju korisnika u aplikaciji.

Datoteka routers predstavlja jedan od glavnih dijelova backend-a koja koristi koncept rutiranja za mapiranje HTTP zahtjeva. U njemu se nalaze datoteke „users.js“ i „camps.js“ („router files“) koje služe za grupiranje i definiranje različitih „endpoint-a“ odnosno URL ruta koje aplikacija nudi (slika 14.).

Slika 14. Struktura datoteke routers



Izvor : obrada autora

Svaka datoteka predstavlja jedan resurs poput korisnika i kampova. Unutar njih se izjašnjavaju različite HTTP metode poput GET, POST, PUT, DELETE kojima se vrše osnovne CRUD operacije (Create, Read, Update, Delete).

Datoteka camps.js sadrži HTTP metode koje su potrebne za osnovne funkcije kampa.

Slika 15. GET metoda za dohvat svih kampova u bazi

```
// Dohvat svih kampova
router.get("/", async (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 10;

  try {
    const camps = await Camp.find()
      .skip((page - 1) * limit)
      .limit(limit);
    const total = await Camp.countDocuments();

    res.json({
      total,
      pages: Math.ceil(total / limit),
      camps,
    });
  } catch (err) {
    res.status(500).send("Server Error");
  }
});
```

Izvor : obrada autora

Dohvat svih kampova je HTTP metoda koja se koristi kada korisnik želi da vidi sve dostupne kampove, koristeće GET metodu na osnovnoj ruti „/“. Ova ruta je osmišljena tako da pretražuje bazu podataka i dohvata sve kampove. Pomoću parametara page i limit, korisnik može da navodi koju stranicu rezultata želi i koliko kampova po stranici želi da vidi. Implementacijom „Mongoose“, aplikacija koristi metode find(), skip(), i limit() za efikasno dohvaćanje podataka uz paginaciju. U odgovoru, korisnik dobiva informaciju o ukupnom broju kampova, ukupnom broju stranica, kao i listu kampova za traženu stranicu (slika 15.).

Slika 16. POST metoda za dodavanje novog kampa

```
//Dodavanje novog kampa
router.post(
  "/",
  authMiddleware,
  [
    check("name", "Name is required").not().isEmpty(),
    check("picture", "Picture URL is required").not().isEmpty(),
    check("price", "Price is required and must be a number").isNumeric(),
    check("location", "Location is required").not().isEmpty(),
  ],
  async (req, res) => {
    const errors = validationResult(req);

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

    const { name, picture, price, location } = req.body;

    try {
      let camp = new Camp({
        name,
        picture,
        price,
        location,
      });

      await camp.save();

      res.json(camp);
    } catch (err) {
      res.status(500).send("Server Error");
    }
  }
);
```

Izvor : obrada autora

Za dodavanje novog kampa korisnik koristit će POST metodu na istoj osnovnoj ruti „/“. Prije dodavanja kampa, koristi se authMiddleware kako bi se osiguralo da je korisnik autentificiran. Pored toga, podaci koji se šalju prolaze kroz validaciju s pomoću express-validator. Ako podaci nisu u ispravnom formatu, korisniku se šalje odgovarajuća poruka o grešci. Ako je sve u redu, novi kamp se kreira i pohranjuje u bazi, a korisniku se vraća informacija o uspješno kreiranom kampu (slika 16.).

Slika 17. POST metoda za dodavanje komentara

```
//Dodati novi komentar kampu
router.post(
  "/:campId/comment",
  authMiddleware,
  [check("text", "Comment text is required").not().isEmpty()],
  async (req, res) => {
    const errors = validationResult(req);

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

    const { text } = req.body;

    try {
      const camp = await Camp.findById(req.params.campId);

      if (!camp) {
        return res.status(404).send("Camp Not Found");
      }

      camp.comments.push({ user: req.user.id, text: text });
      await camp.save();
      res.json(camp.comments);
    } catch (err) {
      console.error(err.message);
      res.status(500).send("Server Error");
    }
  }
);
```

Izvor : obrada autora

Korisnik može dodati komentar određenom kampu koristeći POST metodu na ruti „/:campId/comment“. Prvo se koristi „authMiddleware“ kako bi se osigurala autentifikacija. Nakon toga, podaci za komentar prolaze kroz validaciju. Ako određeni kamp nije pronađen ili postoji problem s komentarom, korisniku se šalje odgovarajuća poruka o grešci. Ako je sve u redu, komentar se dodaje kampu, pohranjuju se promjene, te korisniku se vraća ažurirana lista komentara za taj kamp (slika 17.).

Slika 18. POST metoda za ocjenjivanje kampa

```
//Dati ocjenu kampu
router.post(
  "/:campId/rate",
  authMiddleware,
  [
    check(
      "score",
      "Score is required and must be a number between 1 and 5"
    ).isInt({ min: 1, max: 5 }),
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    if (!req.user || !req.user.id) {
      return res.status(400).send("User information is missing");
    }
    const { score } = req.body;
    try {
      const camp = await Camp.findById(req.params.campId);
      if (!camp) {
        return res.status(404).send("Camp Not Found");
      }
      const existingRating = camp.ratings.find(
        (r) => r.user && r.user.toString() === req.user.id.toString()
      );
      if (existingRating) {
        existingRating.score = score;
      } else {
        camp.ratings.push({ user: req.user.id, score });
      }
      await camp.save();
      res.json(camp.ratings);
    } catch (err) {
      console.error(err.message);
      res.status(500).send("Server Error");
    }
  }
);
```

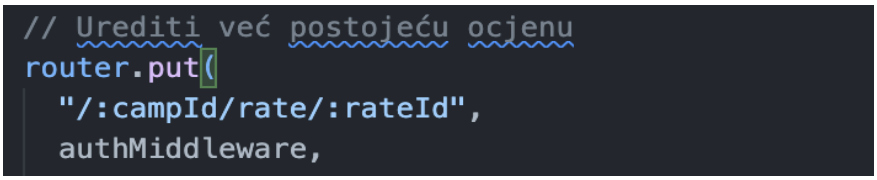
Izvor : obrada autora

Za dodjeljivanje ocjene kampu koristi se POST metodu na ruti „/:campId/rate“. Ova ruta također koristi authMiddleware za autentifikaciju. Ocjena koju korisnik šalje prolazi kroz validaciju. Ako ocjena nije u ispravnom formatu ili određeni kamp nije pronađen, korisnik dobiva poruku o grešci. Aplikacija također provjerava da li korisnik već ima postavljenu ocjenu za taj kamp. Na temelju toga, ocjena se ili dodaje ili ažurira. Na kraju, korisniku se vraća ažurirana lista ocjena za taj kamp (slika 18.).

Slika 19. PUT metoda za uređivanje već postojećeg komentara

Izvor : obrada autora

Slika 20. PUT metoda za uređivanje već postojeće ocjene



Izvor : obrada autora

Za urediti postojeći komentar ili ocjenu koristi se metoda PUT koja omogućava uređivanje postojećeg komentara na kampu ili ocijene.

Korisnik rutom šalje ID kampa i ID komentara koji želi urediti, iz baze se traže podaci o kampu i komentaru po tim ID-jevima. Provjerava se postoji li komentar i je li korisnik njegov vlasnik. Ukoliko je sve u redu, tekst postojećeg komentara ažurira se novim tekstom. Kamp sa izmijenjenim komentarom sprema se natrag u bazu podataka. Modificirani komentar se vraća korisniku (slika 19.).

Za urediti postojeću ocjenu Ruta sadrži ID kampa i ocjene koju korisnik želi promijeniti i radi na isti način kao za komentar (slika 20.).

Slika 21. DELETE metoda za brisanje komentara

```
//Izbrisati postojeći komentar
router.delete(
  "/:campId/comment/:commentId",
  authMiddleware,
  async (req, res) => {
    try {
      const camp = await Camp.findById(req.params.campId);
      const comment = camp.comments.id(req.params.commentId);
      if (!comment) {
        return res.status(404).json({ msg: "Comment not found" });
      }
      if (!comment.user) {
        return res
          .status(400)
          .json({ msg: "No user associated with this comment" });
      }
      if (comment.user.toString() !== req.user.id) {
        return res.status(403).json({ msg: "Unauthorized" });
      }
      camp.comments = camp.comments.filter(
        (comment) => comment.id !== req.params.commentId
      );
      await camp.save();
      res.json({ msg: "Comment removed" });
    } catch (err) {
      console.error(err.message);
      res.status(500).send("Server Error");
    }
  }
);
```

Izvor : obrada autora

Koristeći HTTP metodu DELETE na ruti koja sadrži ID kampa i ID komentara kojeg želi obrisati. Prije obrade zahtjeva, provjerava se autentifikacija korisnika putem „authentication“ middlewarea. Iz baze se traže podaci o kampu i komentaru koji korisnik želi obrisati, na temelju ID-jeva iz rute. Provjerava se postoji li pronađeni komentar i je li korisnik njegov vlasnik. Ukoliko komentar ne postoji ili korisnik nije vlasnik, vraća se odgovor s greškom. Ako su svi uvjeti zadovoljeni, komentar se briše iz niza komentara tog kampa. Briše se na način da se iz niza filtriraju svi komentari čiji ID nije jednak ID komentara koji se briše. Nakon brisanja, kamp sa osvježenim nizom komentara upisuje se natrag u bazu podataka. Korisniku se uspješno vraća poruka o obrisanom komentaru (slika 21.).

Slika 22. Slika 22. GET metoda za pretraživanje kampa

```
// Search
router.get("/search", async (req, res) => {
  try {
    const keyword = req.query.keyword;
    const camps = await Camp.find({
      $or: [
        { name: new RegExp(keyword, "i") },
        { location: new RegExp(keyword, "i") },
      ],
    });
    res.json(camps);
  } catch (err) {
    res.status(500).send("Server Error");
  }
});
```

Izvor : obrada autora

Ruta definirana pod „search“ u kodu omogućuje korisnicima da pretražuju kampove temeljem ključne riječi. Ta ključna riječ, koja se prenosi kao parametar upita s imenom „keyword", koristi se za pretragu imena i lokacija kampova unutar baze podataka. Da bi se postigla fleksibilnost pretrage, koristi se MongoDB „$or“ operator, koji pretražuje i ime kampa i njegovu lokaciju. Također, koristeći „RegExp“ s opcijom "i", pretraga postaje neosjetljiva na velika i mala slova. Kada sustav pronađe kampove koji odgovaraju kriteriju, vraća ih korisniku u obliku JSON odgovora. Ako naiđe na problem tijekom obrade, hvata grešku i vraća odgovor sa statusom 500, informirajući korisnika o unutarnjoj greški servera.

Slika 23. GET metoda za filtriranje i sortiranje kampova

```
//Filtriranje i sortiranje
router.get("/filter-sort", async (req, res) => {
  try {
    let query = Camp.find();
    if (req.query.minPrice) {
      query = query.where("price").gte(req.query.minPrice);
    }
    if (req.query.maxPrice) {
      query = query.where("price").lte(req.query.maxPrice);
    }
    if (req.query.sortBy) {
      query = query.sort(req.query.sortBy);
    }
    const camps = await query.exec();
    res.json(camps);
  } catch (err) {
    res.status(500).send("Server Error");
  }
});
```

Izvor : obrada autora

Ruta pod „/filter-sort“ osmišljena je kako bi korisnicima pružila mogućnost filtriranja i sortiranja kampova temeljem različitih kriterija. Izvršava se upit prema kolekciji „Camp“ da bi se dohvatili svi kampovi. Ako korisnik odredi minimalnu cijenu preko parametra upita „minPrice“, kod provjerava i filtrira sve kampove čija je cijena veća ili jednaka toj zadanoj vrijednosti. Slično tome, za maksimalnu cijenu, koristi se „maxPrice“ kako bi se izuzeli kampovi s cijenom iznad zadane. Ako je sortiranje specificirano kroz „sortBy“ parametar upita, kampovi će biti sortirani sukladno tom kriteriju. Nakon što su svi filteri i sortiranja primijenjeni, izvršava se finalni upit i korisniku se vraća lista kampova u obliku JSON-a (slika 23.).

Slika 24. Implementacija u datoteci users.js

```
const express = require("express");
const router = express.Router();
const { check, validationResult } = require("express-validator");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const { jwtSecret } = require("../config/auth");
const User = require("../models/User");
const authMiddleware = require("../middleware/auth");
```

Izvor : obrada autora

Datoteka users.js koristi se za upravljanje korisnicima unutar web aplikacije koja je izgrađena na Express.js okviru. Na početku datoteke, različiti moduli kao što su express, express-validator, bcryptjs, i jsonwebtoken su implementirani kako bi pružili potrebne funkcionalnosti za upravljanje korisnicima (slika 24.).

Slika 25. POST metoda za dodavanje novog korisnika

```
// Novi korisnik
router.post(
  "/register",
  [
    check("username", "Username is required").not().isEmpty(),
    check(
      "password",
      "Please enter a password with 6 or more characters"
    ).isLength({ min: 6 }),
    check("email", "Email is required").isEmail(),
    check("firstName", "First name is required").not().isEmpty(),
    check("lastName", "Last name is required").not().isEmpty(),
    check("avatar", "Avatar URL is required").isURL(),
  ],
```

Izvor : obrada autora

Prva ruta, „/register“, omogućuje registraciju novih korisnika. Koriste se razne provjere iz express-validator biblioteke da se osigura da korisnički unos odgovara očekivanim formatima i standardima, kao što je minimalna duljina lozinke ili validnost e-mail adrese. Kada korisnik pokuša registrirati se, prvo se provjerava postoji li korisnik s istim korisničkim imenom (slika 25.). Ako postoji, vraća se poruka greške. Ako ne postoji, novi korisnički račun se stvara i sprema u bazu podataka. Nakon uspješne registracije, aplikacija generira JWT (json web token), koji je potom poslan korisniku kao potvrda uspješne registracije i autentifikacije.

Slika 26. POST metoda za login

```
// Login user
router.post(
  "/login",
  [
    check("username", "Username is required").not().isEmpty(),
    check("password", "Password is required").exists(),
  ],
  async (req, res) => {
    const errors = validationResult(req);

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

    const { username, password } = req.body;

    try {
      const user = await User.findOne({ username });
      if (!user) {
        return res.status(400).json({ msg: "Invalid Credentials" });
      }

      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res.status(400).json({ msg: "Invalid Credentials" });
      }

      const payload = {
        user: {
          id: user.id,
        },
      };
      jwt.sign(payload, jwtSecret, { expiresIn: 3600 }, (err, token) => {
        if (err) throw err;
        res.json({ token });
      });
    } catch (err) {
      res.status(500).send("Server Error");
    }
  }
);
```

Izvor : obrada autora

Druga ruta, „/login“, omogućuje korisnicima da se prijave. Slično kao i kod registracije, koriste se provjere kako bi se osiguralo da su korisničko ime i lozinka navedeni. Zatim se provjerava postoji li korisnik s navedenim korisničkim imenom u bazi podataka. Ako korisnik postoji, provjerava se podudara li navedena lozinka s enkriptiranom lozinkom u bazi podataka. Ako se lozinke podudaraju, korisniku se izdaje JWT kao potvrda uspješne prijave. Ovaj token potom može koristiti korisnik za autentifikaciju u budućim zahtjevima.

Slika 27. Struktura datoteke server.js

```
JS server.js > ...
1   const express = require("express");
2   const bodyParser = require("body-parser");
3   const cors = require("cors");
4   const connectDB = require("./config/db");
5   const path = require("path");
6
7   require("dotenv").config();
8
9   const app = express();
10  app.use(express.json());
11  app.use(cors());
12
13  app.use(bodyParser.json());
14  app.use(cors());
15
16  app.use("/uploads", express.static(path.join(__dirname, "uploads")));
17
18  connectDB();
19
20  const errorHandler = require("./middleware/errorHandler");
21  app.use(errorHandler);
22
23  const campRoutes = require("./routers/camps");
24  const userRoutes = require("./routers/users");
25
26  app.use("/api/camps", campRoutes);
27  app.use("/api/users", userRoutes);
28
29  const PORT = process.env.PORT || 3000;
30
31  app.listen(PORT, () => {
32    console.log(`Server started on port ${PORT}`);
33  });
```

Izvor : obrada autora

Server.js je glavna datoteka za pokretanje backend-a. Implementacija modula koji će se koristiti unutar aplikacije. Modul „express“ učitava se za kreiranje i upravljanje web serverom, dok se „body-parser“ koristi za analiziranje dolaznih zahtjeva i pretvaranje zahtjeva u format koji je lakše razumjeti i obraditi.

Modul „cors“ koji omogućuje aplikaciji da komunicira s klijentima iz različitih izvora, prevladavajući tako ograničenja istog izvora. Modul „connectDB“ iz direktorija „config“ koristi se za uspostavljanje veze s bazom podataka, dok je „path“ standardni Node.js modul koji omogućuje rad s putanjama datoteka i direktorija.

Linija „require("dotenv").config();“ koristi se za učitavanje varijabli okoline iz „.env“ datoteke, omogućujući aplikaciji da koristi te varijable za konfiguraciju.

Koristi različite middleware. „Middleware express.json()“omogućuje aplikaciji da automatski analizira JSON tijela dolaznih zahtjeva. Zatim, app.use("/uploads", express.static(path.join(__dirname, "uploads"))); konfigurira aplikaciju da poslužuje statičke datoteke iz direktorija uploads, što omogućuje klijentima da pristupe tim

datotekama putem /uploads rute.
Rute za kampove i korisnike iz odgovarajućih direktorija i dodaju u aplikaciju pod rute „/api/camps“ i „/api/users“. Ovime se potvrđuje da je sve postavljeno i da aplikacija može pravilno obraditi dolazne zahtjeve (slika 27.).

3.2. Frontend

Frontend predstavlja korisnički (client-side) dio web aplikacije za ocjenjivanje kampova u Hrvatskoj. Razvijen je pomoću Vue.js, popularnog frameworka za izradu komponentnih aplikacija.

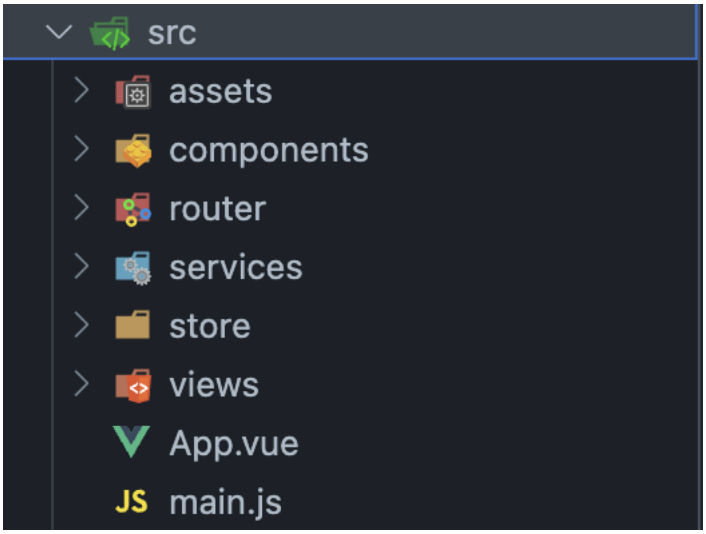
Cilj frontend dijela je omogućiti korištenje aplikacije kroz sučelje te učitavanje i manipulaciju podacima iz baze. Razvijen je kao jednostavna, ali funkcionalna platforma za pregledavanje, ocjenjivanje i komentiranje kampova.

3.2.1. Arhitektura Vue aplikacije

Aplikacija je strukturirana pomoću Vue komponentnog modela, routinga i servisa kao što se vidi na slici 28. Sastoji se od:

Assets datoteke – sadrži statičke datoteke poput slika i CSS stilova.
Components datoteka – sadrži osnovne Vue komponente
Router datoteka – sadrži „Vue router modul“ koji definira glavne rute i njihove komponente.
Services – sadrži logiku za komunicirati sa serverom (backend).
Store datoteka – koristi se za držanje stanja poput korisničkih podataka
Views datoteka – sadrži se od glavnih Vue stranica.
App.js – glavna aplikacijska komponenta
Main.js – implementacija glavne instance i „router“ modula

Slika 28. Struktura frontend-a (Vue struktura)



Izvor : obrada autora

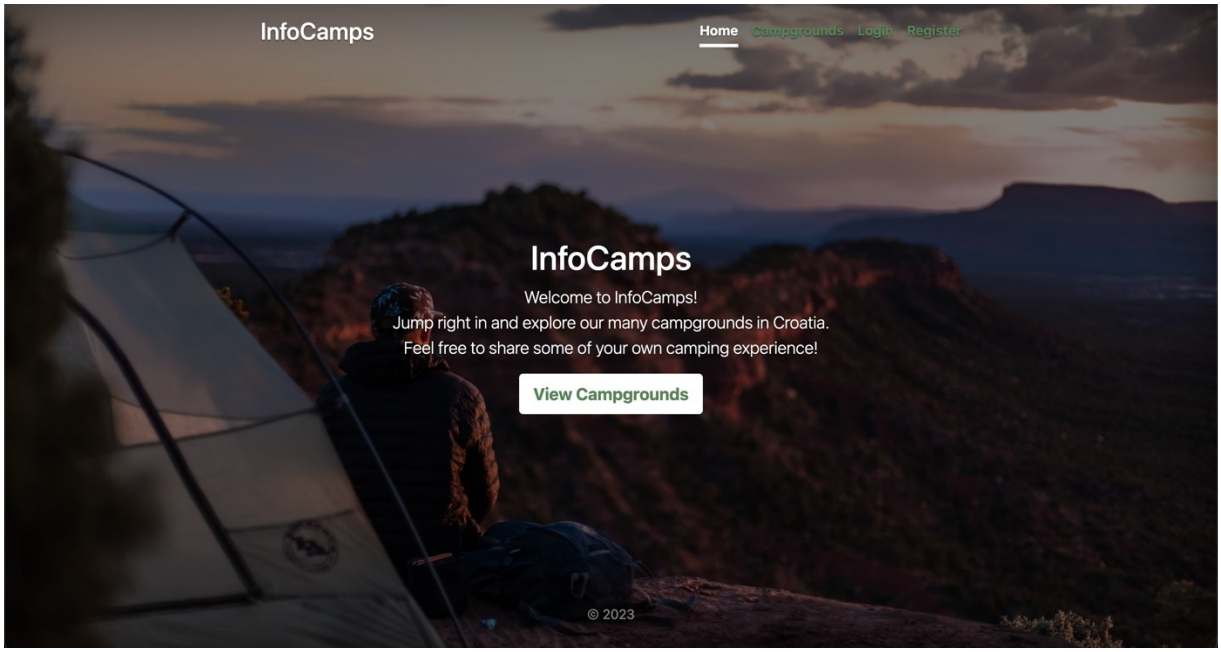
3.2.2. Detaljni opis Vue strukture

Prvo što se opisuje je App.vue datoteka. App.vue datoteka je početna stranica web aplikacije (slika 29.) i ključna komponenta svake Vue.js aplikacije. Predstavlja osnovnu komponentu koja pokreće sve ostale komponente unutar aplikacije. Uz pomoć Vue rutiranja, App.vue omogućuje navigaciju i prikaz različitih komponenata.

App.vue se sastoji od „templete“ dijela koji se označava elementima „<template>“ i „</template>“ koji definiraju HTML dio strukturu datoteke.

U HTML dio koji se sastoji od osnovnih elemenata kao što su navbar (navigacijska traka, označna elementom „<nav>“). Korištenjem „<router-link>“ komponenti, omogućena je navigacija kroz aplikaciju bez potrebe za ponovnim učitavanjem stranice. Pomoću <router-view> komponente, sadržaj se dinamički mijenja ovisno o aktivnoj ruti (slika 30.).

Slika 29. Počena stranica web aplikacije



Izvor : obrada autora

Slika 30. Struktura App.vue datoteke, templete dio datoteke

```
<template>
  <div id="app">
    <nav>
      <div class="container">
        <router-link to="/" class="title">InfoCamps</router-link>
        <ul class="nav-links">
          <li><router-link to="/">Home</router-link></li>
          <li><router-link to="/camps">Campgrounds</router-link></li>
          <li v-if="!loggedIn"><router-link to="/login">Login</router-link></li>
          <li v-if="!loggedIn">
            <router-link to="/register">Register</router-link>
          </li>
        </ul>
      </div>
    </nav>
    <h1>InfoCamps</h1>
    <h3>Welcome to InfoCamps!</h3>
    <h2>
      Jump right in and explore our many campgrounds in Croatia. Feel free to
      share some of your own camping experience!
    </h2>
    <router-link to="/camps"> View Campgrounds</router-link>
    <main>
      <router-view />
    </main>
  </div>
</template>
```

Izvor : obrada autora

Druga bitna komponenta Vue strukture je Vue router koji omogućava definiranje dinamičkih rutnih točaka i komponenti u Vue aplikaciji. Osnovna svrha je navigacija između različitih URL ruta na komponente koje želimo prikazati. U ovom routeru kao što vidimo na slici 32.

Inicijaliziramo potrebne komponente s naredbom „import { createRouter, createWebHistory } from "vue-router""(slika 31.), nakon toga definiramo rutne točke kao niz objekata, svaka sadrži:

- „path“: URL putanja.
- „name“: Ime rute.
- „component“: Komponenta koja će biti prikazana kada se ta ruta posjeti.
- „props“: Omogućuje prosljeđivanje ruta parametara kao props-a komponenti.
- „meta“: Koristi se za prilagođene meta informacije o ruti (npr. zaštita rute).

Komponenta „createRouter“ kreira instancu sa history modelom za lakšu URL navigaciju. Komponeta „router.beforeEach“ hook koristi se za provjeru autentifikacije.

Slika 31. Pozivanje funkcija iz paketa "vue-router"

```
import { createRouter, createWebHistory } from "vue-router";
```

Izvor : obrada autora

Slika 32. Struktura Vue routes

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
  },
  {
    path: "/camps",
    name: "Camps",
    component: Camps,
  },
  {
    path: "/camp/:id",
    name: "CampDetail",
    component: CampDetail,
    props: true,
  },
  {
    path: "/login",
    name: "Login",
    component: Login,
    meta: {
      guestOnly: true,
    },
  },
  {
    path: "/register",
    name: "Register",
    component: Register,
    meta: {
      guestOnly: true,
    },
  },
  {
    path: "/profile",
    name: "Profile",
    component: Profile,
    meta: {
      requiresAuth: true,
    },
  },
```

Izvor : obrada autora

Treći dio što se upisuje je „services datoteke“ u modernom razvoju web aplikacija, posebno kada radite s okruženjima poput Vue.js, „services“ su često ključni dio arhitekture. Services se koriste za upravljanje interakcijama između frontend-a i backend-a. Services mapa se sastoji od api.js, campService.js i userService.js.

Prvi koji se opisuje je api.js pošto se poziva axios paket s naredbom „import“.
Axios je popularna JavaScript biblioteka koja služi za slanje HTTP zahtjeva iz preglednika i Node.js aplikacija. Jedna od njegovih velikih prednosti je to što je potpuno nezavisan od bilo kog backend-a, te se može koristiti sa bilo kojim serverom koji podržava HTTP protokol. Svi zahtjevi koje izvršava vraćaju „Promise“ objekt što omogućava asinkrono rukovanje rezultatima. Također nudi mogućnost interceptiranja zahtjeva i odgovora kroz različite točke u životnom ciklusu zahtjeva poput prije slanja zahtjeva ili prije dobivanja odgovora. To je korisno za funkcije poput loggin-a ili dodavanja autentifikacije podataka u „header“. Biblioteka također podržava i transformaciju podataka tokom slanja i primanja podataka, na primjer serijalizaciju objekata u JSON formatu. Axios ima izvrsnu kompatibilnost sa svim modernim preglednicima te se stoga smatra najčešće korištenom bibliotekom za HTTP komunikaciju na frontend-u. Api.js ima dvije bitne funkcije koje se pozivaju a to su „create“ i „interceptors“ kao što se može vidjeti na slici.

Slika 33. Struktura api.js datoteke

```
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:3000/api",
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers["Authorization"] = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

export default api;
```

Izvor : obrada autora

Axios.create funkcija stvara novu instancu Axios klijenta s predefiniranom konfiguracijom. U ovom slučaju, baseURL je postavljen na lokaciju API-a.
Interceptors su funkcije koje Axios poziva prije ili nakon što zahtjev ili odgovor prođe. Ovdje je postavljen interceptor zahtjeva koji dodaje autorizacijski token iz lokalne pohrane (ako postoji) u zaglavlje svakog zahtjeva.

Datoteka campService.js je servisni modul čija je primarna svrha upravljanje svim interakcijama koje se odnose na kampove u aplikaciji. Služi kao komunikacija između frontend-a i backend-a, pružajući metode koje koriste Axios (kroz api modul) kako bi komunicirale s odgovarajućim backend rutama.
Prva stvar koju obavlja je uvoz „api“ modula, te sljedeće je konstanta CAMPS_URL, koja predstavlja osnovni URL endpoint-a za kampove (slika 34.).

Slika 34. import api iz api.js datoteke i konstanta CAMPS_URL

Izvor : obrada autora

Sljedeće metode koje koristi su :

- fetchCamps: - dohvaća listu svih kampova
- fetchCampDetail: - dohvaća detalje određenog kampa na temelju njegovog
- addComment: - dodaje komentar na određeni kamp.
- rateCamp: - ocjenjuje određeni kamp.
- createCamp: - kreira novi kamp.
- editComment: - uređuje komentar na određenom kampu.
- editRating: - uređuje ocjenu za određeni kamp.
- deleteComment - briše komentar na određenom kampu.

Slika 35. Primjer "fetchCamps" i "fetchCampDetail" metode

```
async fetchCamps() {
  try {
    const response = await api.get(CAMPS_URL);
    return response.data;
  } catch (error) {
    throw new Error(`Failed to fetch camps: ${error.message}`);
  }
},

async fetchCampDetail(id) {
  try {
    const response = await api.get(`${CAMPS_URL}/${id}`);
    return response.data;
  } catch (error) {
    throw new Error(`Failed to fetch camp detail: ${error.message}`);
  }
},
```

Izvor : obrada autora

Isto tako datoteka „userService.js“ ima metode:

- Login - omogućuje korisniku da se prijavi
- Register - omogućuje novom korisniku registraciju pružanjem potrebnih podataka.
- FetchProfile - dohvaća profil trenutno prijavljenog korisnika.

Četvrto što se upisuje su mape „components“ ili vue komponente i „views“ ili prikazi. Vue.js je progresivni JavaScript okvir koji omogućuje izgradnju korisničkih sučelja s visokom interaktivnošću. Temelji se na komponentnom pristupu, gdje svaki dio korisničkog sučelja može biti predstavljen kao samostalna, ponovno iskoristiva komponenta. Kada aplikacije postaju složenije, često je korisno razlikovati "komponente" od "prikaza" ili "views". Oba pojma su ključna za razumijevanje strukture Vue.js aplikacija. Komponente su ponovno iskoristive Vue instance s imenom. One su osnovni građevni blokovi Vue aplikacije. Jednom kada definirate komponentu, možete je koristiti na više mjesta unutar aplikacije ili čak izvan nje.

Svaka komponenta ima svoj vlastiti opseg i data svojstva. To znači da promjene unutar komponente ne utječu na ostatak aplikacije. Komponente se mogu ugnijezditi jedna unutar druge, omogućujući izgradnju složenih korisničkih sučelja iz jednostavnih dijelova. Primjeri su gumbi, kartice, trake za navigaciju, forme i ostale UI jedinice. U ovom primjeru imamo komponentu „AddCamp.vue“ koja služi za dodavanje novog

kampa u bazi, koja se poziva te koristi u prikazu (views) „CampView.vue“. Poziva se s naredbom „import AddCamp from "../components/AddCamp.vue" kao što se vidi na slici 36.

Slika 36. Import AddCamp.vue komponenta u CampView.vue

```
<script>
import CampList from "@/components/CampList.vue";
import campService from "@/services/campService.js";
import AddCamp from "@/components/AddCamp.vue";

export default {
  components: {
    CampList,
    AddCamp,
  },
```

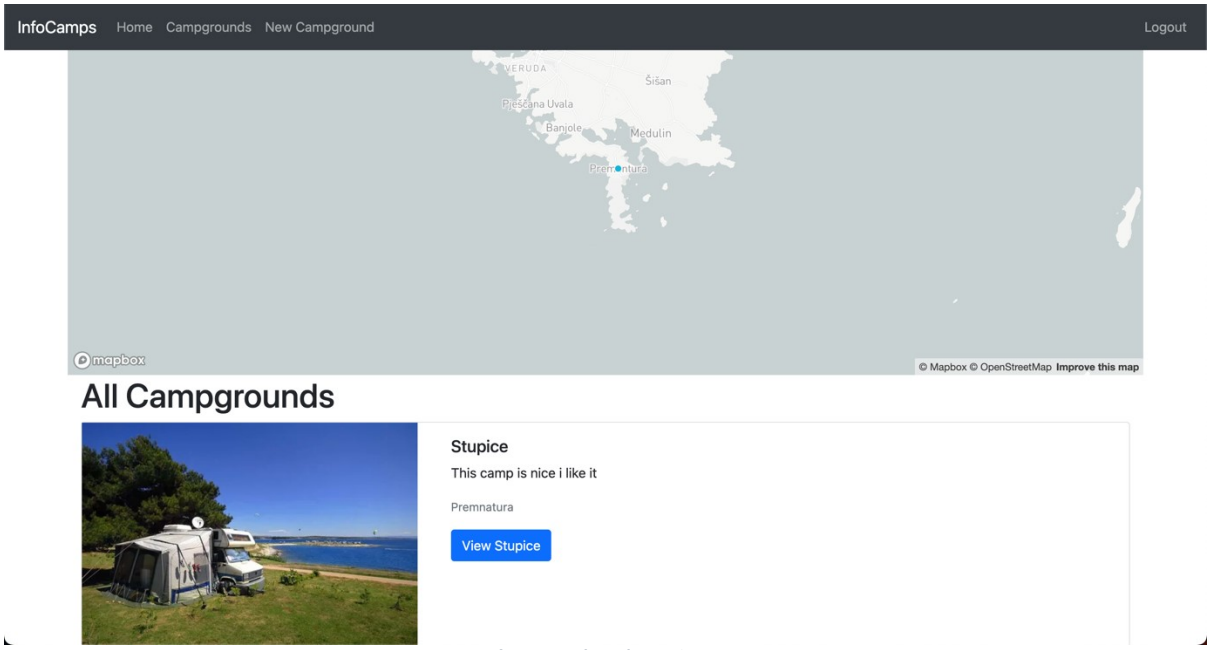
Izvor : obrada autora

Mapa „componente“ ili komponente se sastoji od :
AddCamp.vue – ova komponenta služi za dodavanje nove kampove
CampComment.vue – ova komponenta služi za dodavanje komentara
CampItem.vue – služi za vidjeti jedan određeni kamp, svi detalji tog kampa
CampList.vue – služi za izlistani sve kampove
CampRating.vue – služi za ocjenjivanje kampa

Mapa „views“ ili prikazi sastoji se od:

- CampViews.vue – poziva se komponenta AddCamp i CampList pošto nam ovaj prikaz služi za vidjeti sve kampove iz baze i mogućnost dodavanja kampa(slika 37.).

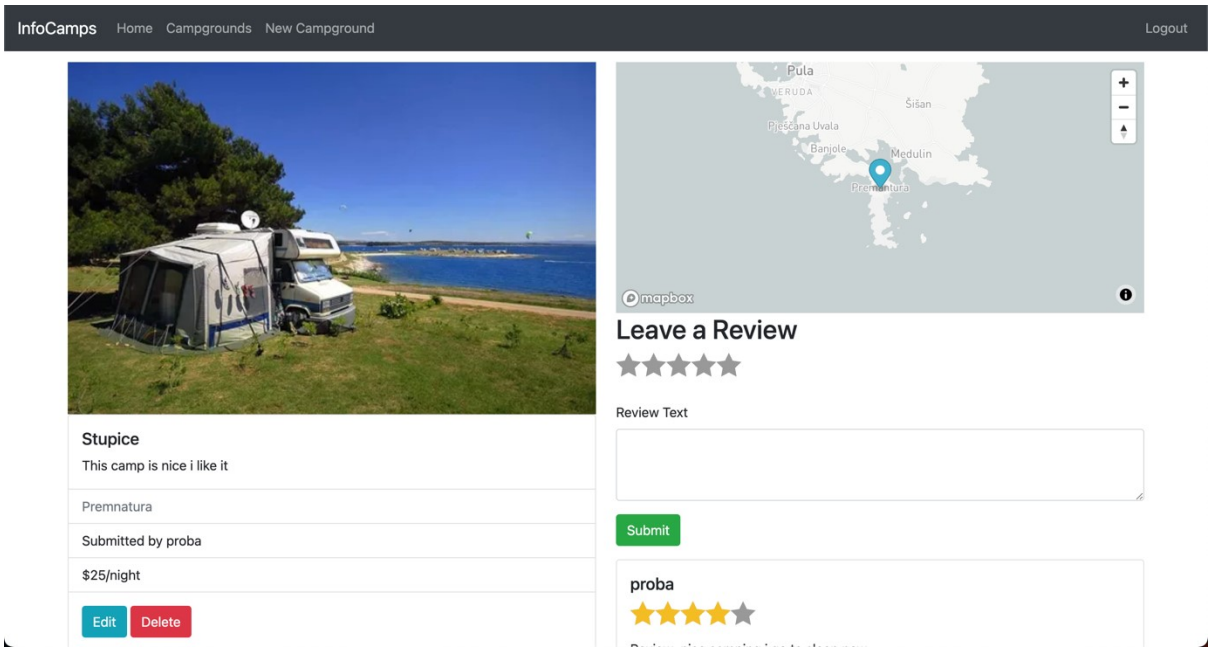
Slika 37. Prikaz lista kampova i dodavanje novog kampa



Izvor : obrada autora

- CampDetails.vue – poziva se komponenta CampItem koja pokazuje svi detalji tog kampa(slika 38.).

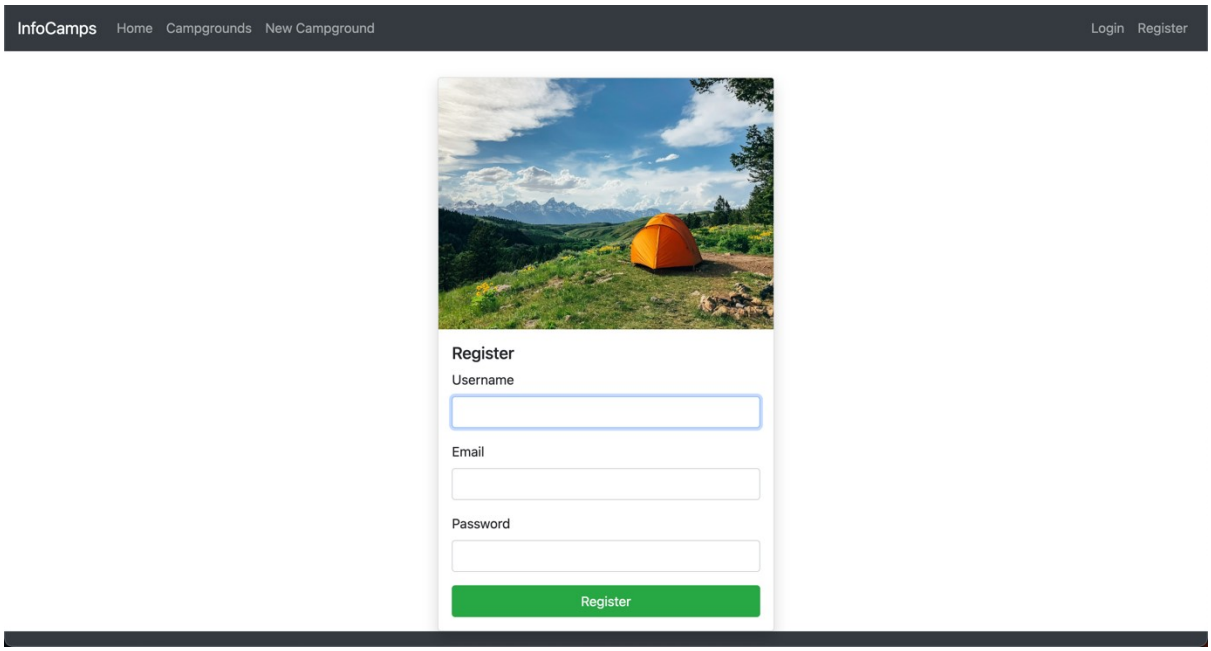
Slika 38. Prikaz CampDetails pokazu je nam se detalje kampa



Izvor : obrada autora

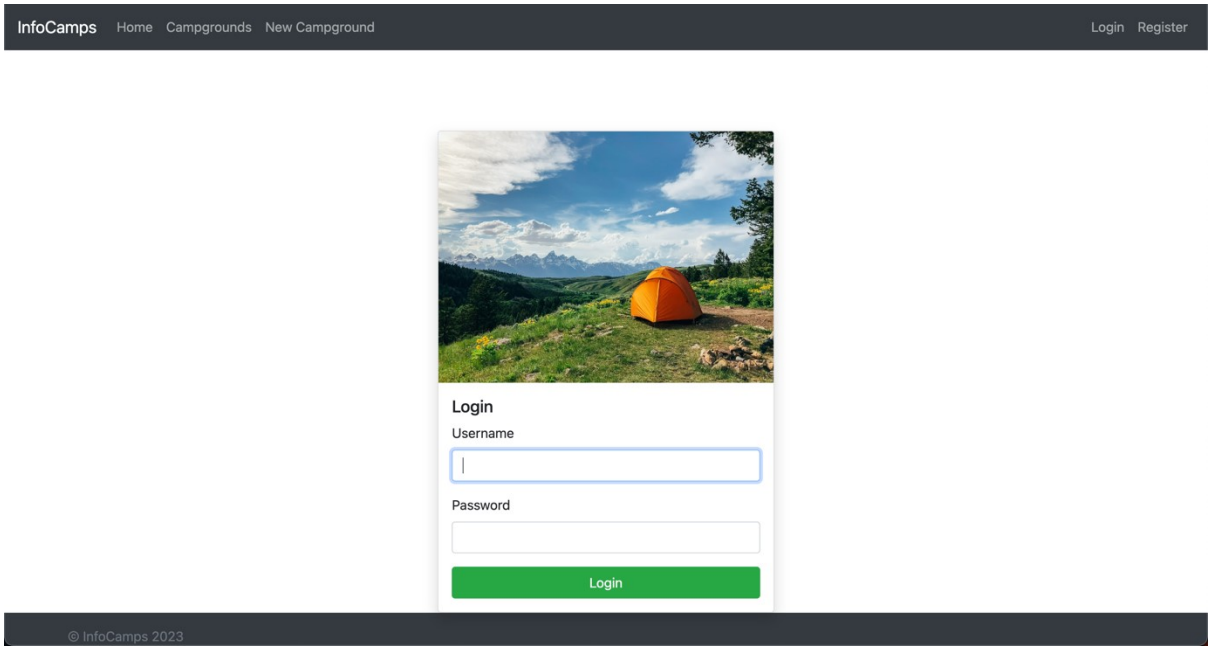
- Datoteke LoginView.vue i Register.vue – koji nam služe kao prijava i registracija korisnika(slika 39. i slika 40.)

Slika 39. Prikaz registracija korisnika



Izvor : obrada autora

Slika 40. Prikaz prijava korisnika



Izvor : obrada autora

4. Baza podataka

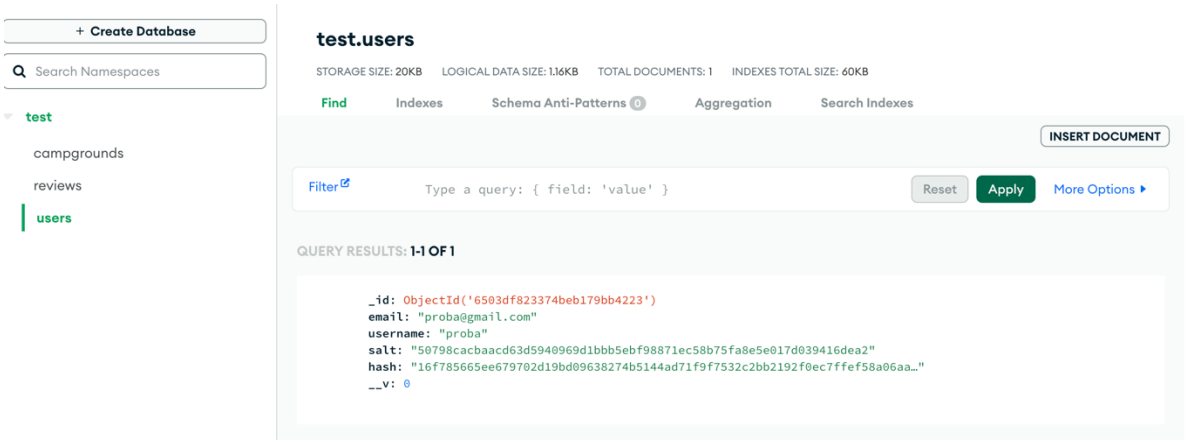
MongoDB je dokumentna baza podataka koja pripada skupini NoSQL baza. Umjesto tradicionalnih tabela i redova, koristi dokumente i kolekcije za pohranu podataka. Glavna prednost MongoDB-a je fleksibilnost koju nudi prilikom skladištenja podataka jer ne zahtijeva fiksnu shemu.

Koristi se „Mongoose“ biblioteka. „Mongoose“ biblioteka je ODM (Object Data Modeling) biblioteka za MongoDB i Node.js. Omogućuje povezivanje aplikacije s MongoDB bazom te definiranje modela i shema za podatke unutar aplikacije.

Za spremanje podataka koristi varijantu JSON-a poznatu kao BSON (Binary JSON). BSON (slika 41.) je binarni format koji proširuje JSON kako bi pružio dodatne tipove podataka. Glavni razlog za korištenje BSON-a umjesto čistog JSON-a je da pruža bolje performanse prilikom serijalizacije i deserijalizacije podataka.

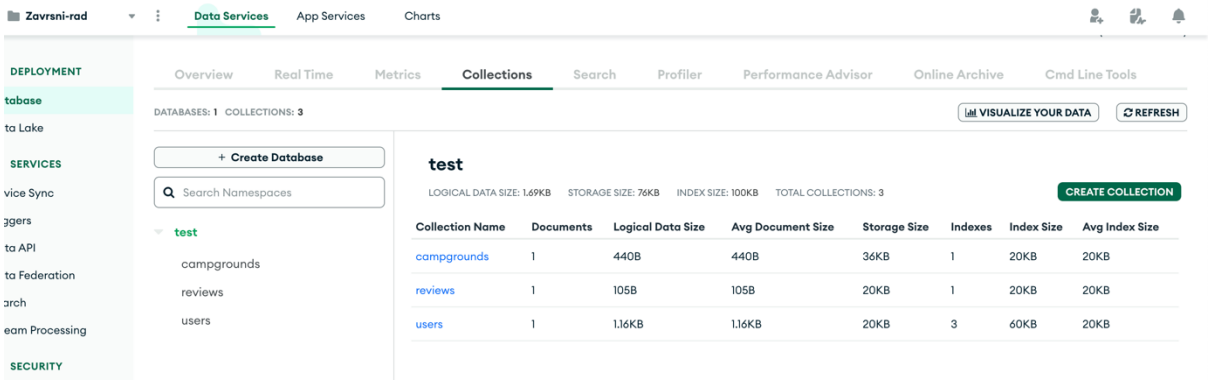
U MongoDB-u, baza podataka je skup kolekcija. Kolekcije su ekvivalent tablicama u relacijskim bazama podataka, ali s bitnom razlikom - dok su tablice u relacijskim bazama podataka strogo definirane shemama, kolekcije u MongoDB-u su fleksibilnije jer BSON dokumenti unutar iste kolekcije mogu imati različite skupove polja. Kolekcija za ovu web aplikaciju se sastoji od campground, reviews i users kao što se može vidjet na slici 42.

Slika 41. Prikaz BSON datoteke u MongoDB bazi



Izvor : obrada autora

Slika 42. Struktura MongoDB baze



Izvor : obrada autora

5. Zaključak

Kroz razvoj web aplikacije namijenjena za ocjenjivanju kampova u Hrvatskoj, ovaj završni rad demonstrira snagu digitalnih alata u turističkom sektoru. Kampiranje je tijekom godina postalo sve popularnije među domaćim stanovništvom, ali i turistima, te pruža jedinstveni doživljaj prirodnih ljepota Hrvatske. S obzirom na sve veću potražnju za informacijama o kampovima, važnost pružanja pouzdanih recenzija i ocjena nikada nije bila veća.

Aplikacija koja je razvijena tijekom ovog rada pruža korisnicima platformu na kojoj mogu dijeliti svoja iskustva, dok pružatelji usluga kampiranja dobivaju dragocjene povratne informacije koje mogu koristiti za poboljšanje svojih usluga. Aplikacija također služi kao centralno mjesto za one koji traže informacije o kampiranju, pružajući im sve što im je potrebno na jednom mjestu.

Razvoj aplikacije na temelju Vue.js-a osigurao je brzu i odzivnu platformu s intuitivnim korisničkim sučeljem, što doprinosi ukupnom korisničkom iskustvu. Uz integraciju MongoDB baze podataka, aplikacija je u mogućnosti pohranjivati i obraditi velike količine podataka u JSON formatu, što osigurava fleksibilnost i skalabilnost potrebnu za ovakve vrste projekata.

S obzirom na sve veći broj turista koji dolaze u Hrvatsku, aplikacija ima potencijal da postane neizostavan alat za planiranje kampiranja. U budućnosti, ovakva platforma može se proširiti i na druge turističke aktivnosti, čime se dodatno potiče turizam u Hrvatskoj.

Uzimajući sve ovo u obzir, jasno je da digitalna transformacija u turističkom sektoru ima ključnu ulogu u oblikovanju budućnosti turizma u Hrvatskoj. Razvoj aplikacija poput ove ne samo da pomaže individualnim korisnicima u donošenju informiranih odluka već također potiče cijelu industriju da teži izvrsnosti, stvarajući bolje iskustvo za sve.

6. Literatura

1. Axios

Web pristup: https://axios-http.com/docs/intro

2. Developer Mozilla

Web pristup: https://developer.mozilla.org/en-US/docs/Web/HTML

3. MongoDB

Web pristup: https://www.mongodb.com/docs/

4. Multer

Web pristup: https://www.npmjs.com/package/multer

5. Node.js

Web pristup: https://nodejs.org/en/docs

6. Vue.js

Web pristup: https://vuejs.org/guide/introduction.html

7. Vue.js routes

Web pristup: https://vuejs.org/guide/scaling-up/routing.html

7. Popis slika