Web aplikacija za freelancer usluge

Bičvić, Leopold

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB APLIKACIJA ZA FREELANCER USLUGE

Diplomski rad

Leopold Bičvić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 12.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Leopold Bičvić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1187R, 07.10.2021.
OIB studenta:	42922753826
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Robert Šojo, mag. ing. comp.
Naslov diplomskog rada:	Web aplikacija za freelancer usluge
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Opisati postupke postavljanja, pretraživanja i preuzimanja poslova koji se objavljuju kao 'freelancer' zadaci. Dizajnirati i opisati dizajn baze podataka koju će koristiti web aplikacija. Izraditi web aplikaciju za freelancer usluge i opisati postupak izrade. U sklopu web aplikacije izraditi upitnik kojim će registrirani korisnici moći ocijeniti iskustvo korištenja izrađene web aplikacije i obraditi rezultate anketiranja grupe korisnika. Tema rezervirana: Leopold Bičvić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	12.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 03.10.2023.

Ime i prezime studenta:	Leopold Bičvić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1187R, 07.10.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za freelancer usluge**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Nagli razvoj tehnologije zahtjeva brzu prilagodbu poslodavaca i radnika na izvršavanje posla s udaljenog mjesta rada. Kako bi se poslodavac i radnik lakše povezali uz poštivanje fleksibilnosti i kreativnosti, potražnja za *freelance* uslugama postaje sve popularnija.

Web aplikacija za freelancer usluge ispunjava osnovne funkcionalnosti poput postavljanja, preuzimanja i pretraživanja poslova odnosno usluga ovisno o odabranoj ulozi prijavljenog korisnika. Ulogom *freelancer* omogućeno je postavljanje, uređivanje i brisanje usluga, a ulogom *user* zapošljavanje vlasnika usluge odnosno *freelancera*. U ovoj aplikaciji, korisnik s ulogom *freelancer* je ponuditelj i izvršitelj usluga, dok je korisnik s ulogom *user* kupac usluge. Pretraživanje usluga dostupno je svim prijavljenim korisnicima. Postupci izrade aplikacije i sve navedene funkcionalnosti aplikacije opisane su u nastavku rada.

Motivacija za izradu ovakve vrste web aplikacije je nastala kao prilika za proširivanjem znanja s obzirom na sve funkcionalnosti koje aplikacija sadržava. Također, izrada ovakve vrste aplikacije pruža priliku za lakše konkuriranje na tržištu rada s obzirom na korištene tehnologije.

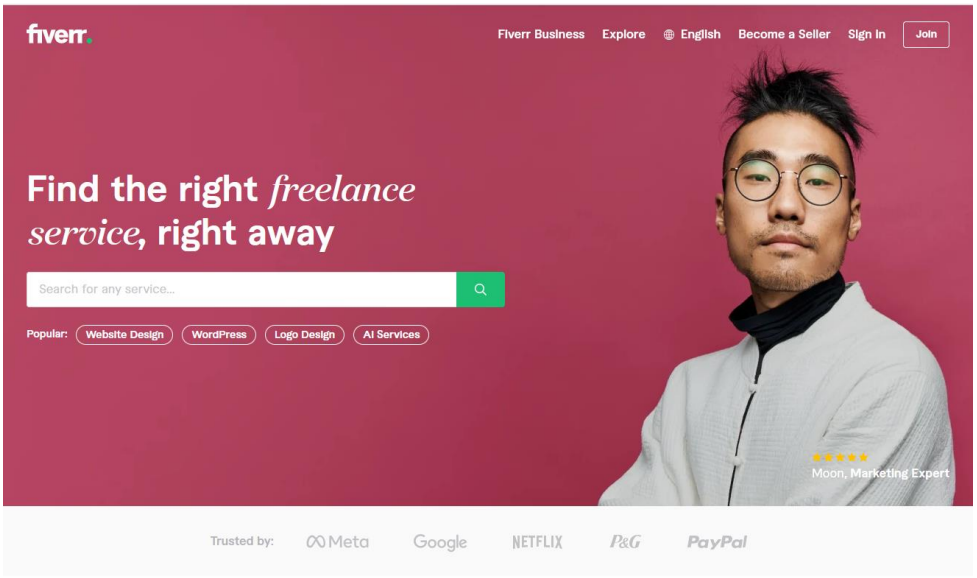
U drugom poglavlju nalaze se popularni primjeri sličnih rješenja web aplikacije uz njihov kratak opis funkcionalnosti. U trećem poglavlju objašnjeni su načini korištenja tehnologija i postupci izrade poslužitelja i klijenta uključujući funkcionalnosti i strukturu pojedinih dijelova web aplikacije. Nakon opisanih postupaka, četvrto poglavlje prikazuje rad web aplikacije. U petom poglavlju prikazani su obrađeni rezultati anketiranja.

2. PREGLED SLIČNIH RJEŠENJA

U poglavlju su ukratko opisana slična rješenja uz prikaz njihovih početnih stranica. U opisima sličnih rješenja navedena je i usporedba s izrađenom web aplikacijom kojom su istaknute sličnosti i razlike aplikacija.

2.1. Fiverr

Fiverr je među najpoznatijim web aplikacijama za pružanje *freelancer* usluga. Aplikacija nudi razne funkcionalnosti poput pretraživanja objavljenih usluga i preuzimanja poslova. Uz osnovne funkcionalnosti, Fiverr nudi dodatno usavršavanje znanja pomoću internog bloga dostupnog svim korisnicima. Sličnost Fiverr aplikacije i izrađene web aplikacije odnosi se na funkcionalnost postavljanja usluga i pretragu po ključnoj riječi. Razlika je u mogućnosti pružanja dodatnog usavršavanja znanja korisnika unutar Fiverr aplikacije. Slikom 2.1. prikazana je početna stranica Fiverr-a [1].

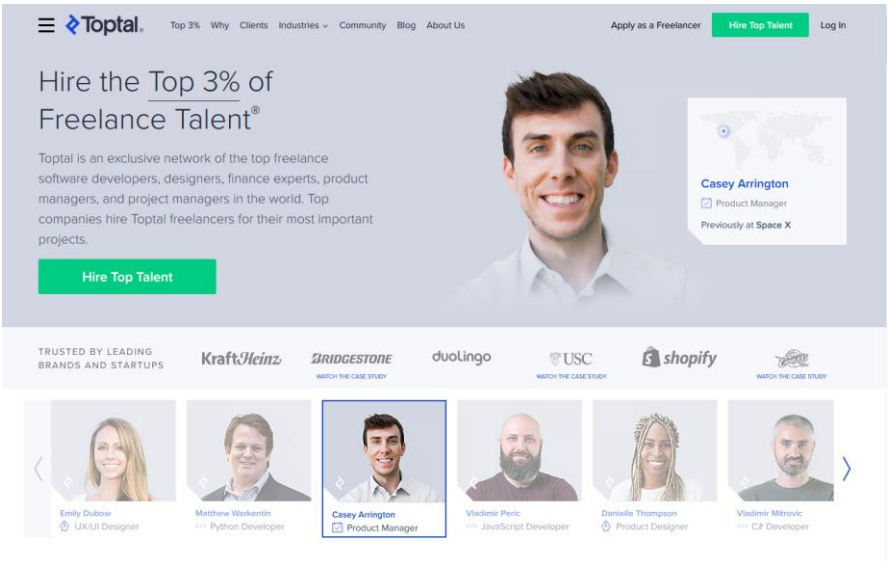


Slika 2.1. *Prikaz početne stranice web aplikacije Fiverr*

2.2. Toptal

Toptal je također jedna od najpoznatijih aplikacija za *freelancer* usluge. Pokriva širok spektar mogućnosti davanja usluga u raznim područjima tehnologije. Nudi brzu povezanost registriranih korisnika. Toptal pruža kvalitetnu pretragu objavljenih poslova i korisnika pomoću upitnika s zadanim ključnim riječima. Ovakva vrsta pretrage je različita u odnosu na pretragu po unosu ključne riječi unutar izrađene web aplikacije. Sličnost između Toptala i izrađene web aplikacije

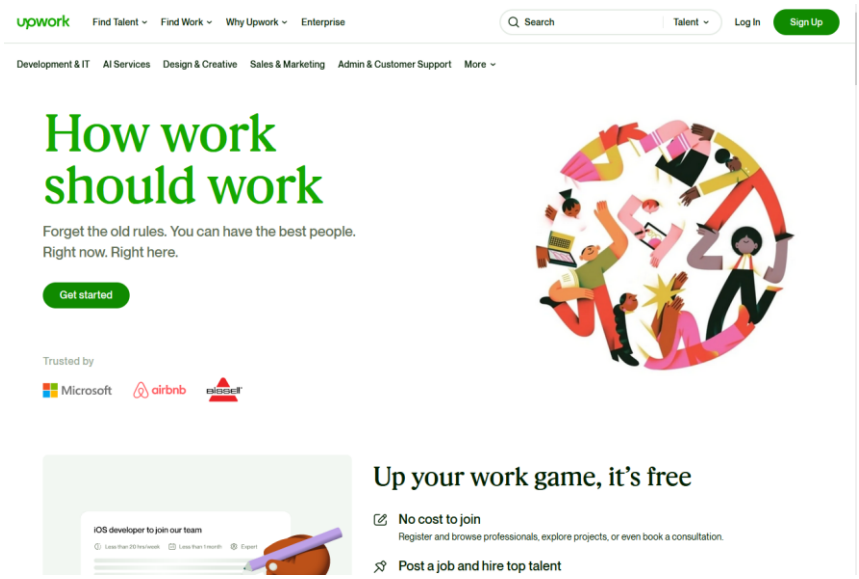
je mogućnost uvida u detalje korisničkog profila. Na slici 2.2. prikazan je izgled naslovne stranice Toptal [2].



Slika 2.2. *Prikaz naslovne stranice Toptal*

2.3. Upwork

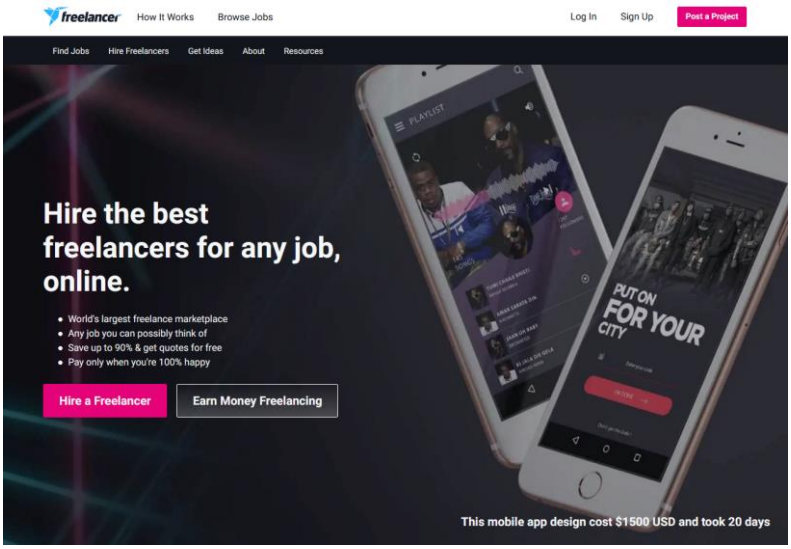
Upwork kroz svoju platformu nudi širok izbor kategorija rada. Također sadrži pretraživanje objavljenih usluga po ključnoj riječi i jednostavno korisničko sučelje. Sličnost Upworka i izrađene web aplikacije je ponuđen izbor pretrage usluga pomoću kategorija na naslovnoj stranici, dok je razlika u mogućnosti postavljanja recenzija unutar Upwork aplikacije. Na slici 2.3. prikazana je početna stranica Upwork-a [3].



Slika 2.3. *Prikaz početne stranice Upwork*

2.4. Freelancer

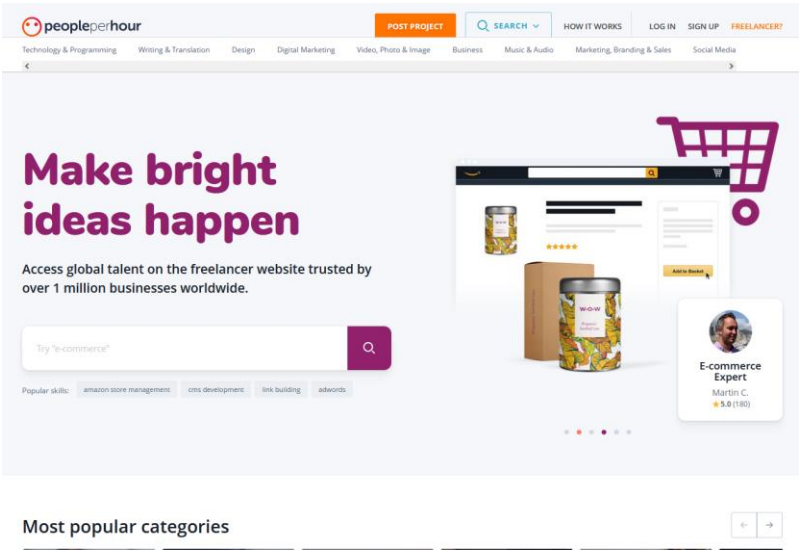
Freelancer također kroz svoju platformu nudi pretragu usluga te funkcionalnosti postavljanja, uređivanja i brisanja usluga kao i ostale aplikacije navedene u ovom poglavlju. Aplikacija se razlikuje od ostalih po tome što uz široku ponudu kategorija, nudi sustav nadmetanja za poslove. Sličnost Freelancer aplikacije i izrađene web aplikacije je funkcionalnost postavljanja usluga, dok je razlika u pretrazi *freelancer* usluga pomoću izbora predodređenih kategorija poput vještina i lokacije *freelancera*. Slikom 2.4. prikazana je naslovna stranica Freelancer aplikacije [4].



Slika 2.4. *Prikaz naslovne stranice Freelancer*

2.5. Peopleperhour

Peopleperhour slično kao i ostale aplikacije nudi funkcionalnosti pretrage, postavljanja i preuzimanja poslova kroz jednostavno korisničko sučelje. Također sadrži širok izbor kategorija rada i brzu pretragu usluga korištenjem predefiniranih filtera poput vremena izrade usluge, cijene i lokacije *freelancera*. Sličnost Peopleperhour i izrađene web aplikacije je jednostavno korisničko sučelje i pretraga objavljenih usluga po ključnoj riječi, dok je razlika u korištenju predefiniranih filtera prilikom pretrage. Na početnoj stranici, uz formu za pretragu nalaze se popularne kategorije. Slikom 2.5. prikazan je izgled naslovne stranice [5].



Slika 2.5. *Prikaz naslovne stranice Peopleperhour*

3. OPIS IZRADE APLIKACIJE

U poglavlju su objašnjeni poslužitelj i klijent izrađene web aplikacije slijedno prateći način korištenja upotrijebljenih tehnologija i postupke izrade aplikacije kroz detaljnu razradu funkcionalnosti. Svaka izrađena komponenta sadrži svoj model i kontroler na poslužitelju te pogled (engl. *view*) na klijentu aplikacije. Primijenjene tehnologije su React, Node.js, Express.js i MongoDB. Navedene tehnologije poznatije su pod kraticom MERN (engl. *MongoDB, Express.js, React, Node.js*). Node.js pruža skalabilnost, kompatibilnost s različitim platformama i veliku korisničku podršku s obzirom na popularnost [6]. Express.js se koristi uz Node.js radi ostvarivanja jasne komunikacije između poslužitelja i klijenta. To se postiže primjenom posrednika (engl. *middleware*) i definiranjem ruta (engl. *routes*) [7]. MongoDB pruža lagano integriranje u aplikaciju i brzu pohranu podataka [8]. React sadrži razne načine razvoja sučelja, no važno je istaknuti način razlaganja koda na komponente. Razlaganje na komponente nudi fleksibilan pristup razvoju aplikacija i sprječava dupliciranje koda. Time se olakšava i održavanje samih aplikacija [9].

3.1. Poslužiteljski dio aplikacije

Za izradu poslužiteljskog dijela aplikacije korištene su Node.js, Express.js i MongoDB tehnologije. Node.js i Express.js upotrijebljeni su za izradu modela, kontrolera i potrebnih ruta, dok je MongoDB korišten za pohranu podataka. S obzirom na navedeno, važno je istaknuti izrađene modele, kontrolere i rute. U modelu korisnika definirane su uloge *freelancer* i *user*. Odabirom uloge *freelancer*, omogućen je pregled te upravljanje zahtjeva za preuzimanjem poslova. Korisnici s ulogom *user* imaju mogućnost slanja zahtjeva za zapošljavanje davatelja usluge odnosno *freelancera.* Neovisno o ulozi svim registriranim korisnicima dostupno je pretraživanje usluga. Na slici 3.1. prikazan je model dokumenta korisnika koji sadrži atribute poput putanje do profilne slike, zaporke, imena, prezimena, uloge i slično. S obzirom na veličinu slike, važno je napomenuti kako se sprema samo putanja do profilne slike, dok je stvarna slika spremljena na vanjsku uslugu *cloudinary*.

```
const Schema = mongoose.Schema;

const userSchema = new Schema(
  {
    firstName: {
      type: String,
      required: true,
    },
    lastName: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      required: true,
      enum: ["Freelancer", "User"],
      default: "User",
    },
    occupation: {
      type: String,
      default: "Basic Knowledge",
    },
    profilePicture: {
      public_id: {
        type: String,
        required: true,
      },
      url: {
        type: String,
        required: true,
      },
    },
  },
  { timestamps: true }
);
```

Slika 3.1. *Prikaz modela korisnika*

Nakon korisničkog modela, važno je istaknuti izrađene korisničke rute pomoću Express.js tehnologije. Rute usmjeravaju na točno određene zahtjeve nad kojima se pozivaju različite HTTP (engl. *HyperText Transfer Protocol*) metode za obradu podataka.

```
userRouter.post("/login", UserController.loginUser);

userRouter.post("/register", UserController.registerUser);
```

Slika 3.2. *Prikaz ruta za prijavu i registraciju korisnika*

Na slici 3.2. prikazane su korisničke rute za prijavu i registraciju koje usmjeravaju zahtjeve prema korisničkom kontroleru s funkcijama za registraciju i prijavu korisnika. Ruta za prijavu korisnika preusmjerava zahtjev prema funkciji *loginUser* unutar kontrolera korisnika, dok ruta za registraciju korisnika preusmjerava prema funkciji *registerUser* korisničkog kontrolera. Za ostvarivanje korisničkih ruta za prijavu i registraciju korištena je POST metoda. Osnovna funkcionalnost POST metode je mogućnost slanja zahtjeva za kreiranje sadržaja.

```
static async loginUser(req, res) {
  const { email, password } = req.body;
  try {
    const user = await User.login(email, password);
    const token = Helpers.createToken(user._id, user.email);
    req.user = user._id;
    const responseData = Helpers.getUserDataForResponse(user);
    res.status(200).json({ userData: responseData, token });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}

static async registerUser(req, res) {
  const { firstName, lastName, email, password, role, profilePic, occupation } = req.body;

  try {
    const user = await User.register(
      firstName,
      lastName,
      email,
      password,
      role,
      profilePic,
      occupation,
    );

    const token = Helpers.createToken(user._id, user.email);
    const responseData = Helpers.getUserDataForResponse(user);

    res.status(200).json({ userData: responseData, token });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}
```

Slika 3.3. *Prikaz funkcija korisničkog kontrolera za prijavu i registraciju*

Na slici 3.3. prikazane su navedene funkcije korisničkog kontrolera. Funkcija korisničkog kontrolera *loginUser* obrađuje poslani zahtjev za prijavu korisnika, dok funkcija *registerUser* obrađuje poslani zahtjev za registraciju korisnika. Unutar navedenih funkcija korisničkog kontrolera za prijavu i registraciju korištene su funkcije za dodatnu provjeru poslanih podataka poput ispravnosti elektroničke pošte i zaporke. Nakon dodatnih provjera, prilikom registracije korisnika kreira se korisnik kao dokument u kolekciji baze podataka. Slikama 3.4. i 3.5.

prikazane su opisane dodatne funkcije za provjeru poslanih podataka prilikom registracije i prijave korisnika. Na slici 3.6. prikazan je dokument korisnika u kolekciji baze podataka sa svim opisanim atributima prema izrađenom modelu korisnika.

```
userSchema.statics.register = async function (
  firstName,
  lastName,
  email,
  password,
  role,
  profilePic,
  occupation
) {
  if (
    !firstName ||
    !lastName ||
    !email ||
    !password ||
    !role ||
    !profilePic ||
    !occupation
  ) {
    throw Error("Please fill all fields to continue");
  }

  if (!validator.isEmail(email)) {
    throw Error("Email is not valid");
  }

  if (!validator.isStrongPassword(password)) {
    throw Error("Password not strong enough");
  }

  const userExists = await this.findOne({ email });

  if (userExists) {
    throw Error("Email already in use");
  }

  const salt = await bcrypt.genSalt(10);
  const hashPassword = await bcrypt.hash(password, salt);
  const uploadPicture = await cloudinary.uploader.upload(profilePic, {
    folder: "profiles",
  });
  const user = await this.create({
    firstName,
    lastName,
    email,
    password: hashPassword,
    role,
    profilePicture: {
      public_id: uploadPicture.public_id,
      url: uploadPicture.secure_url,
    },
    occupation,
  });

  return user;
};
```

Slika 3.4. *Prikaz funkcije za dodatnu provjeru prilikom registracije*

```
userSchema.statics.login = async function (email, password) {
  if (!email || !password) {
    throw Error("Please fill all fields to continue");
  }

  const user = await this.findOne({ email });

  if (!user) {
    throw Error("User does not exist");
  }

  const comparedPass = await bcrypt.compare(password, user.password);

  if (!comparedPass) {
    throw Error("Invalid password");
  }

  return user;
};

module.exports = mongoose.model("user", userSchema);
```

Slika 3.5. *Prikaz funkcije za dodatnu provjeru prilikom prijave*

```
_id: ObjectId('64933fe7b34beb2e99851697')
firstName: "Leopold"
lastName: "Bičvić"
email: "lbicvic@gmail.com"
password: "$2b$10$SjmKqdogGPrAXs2vVIV0VuZUY8buMnlDVU36KCG2zVS4gO3SUftC6"
role: "Freelancer"
occupation: "Fullstack Developer"
▸ profilePicture: Object
createdAt: 2023-06-21T18:22:31.892+00:00
updatedAt: 2023-06-21T18:22:31.892+00:00
__v: 0
```

Slika 3.6. *Prikaz korisničkog dokumenta u kolekciji*

Nakon opisanih ruta za prijavu i registraciju korisnika, važno je naglasiti rute za korisnički profil kako bi bilo moguće prikazati spremljene korisničke podatke. Rute prikazane slikom 3.7. odnose se na prosljeđivanju zahtjeva za dohvaćanje podataka trenutnog korisnika pomoću GET metode i korisnika koji je vlasnik objavljenih usluga pomoću prethodno opisane POST metode. GET metoda služi za slanje zahtjeva za dohvaćanje podataka. Ruta s GET metodom usmjerava zahtjev prema funkciji korisničkog kontrolera *getCurrentUser*, dok ruta s POST metodom usmjerava zahtjev prema funkciji *getUserById.*

```
userRouter.get(
  "/getCurrentUser",
  AuthMiddleware.requireAuth,
  UserController.getCurrentUser
);

userRouter.post(
  "/getUser",
  AuthMiddleware.requireAuth,
  UserController.getUserById
);
```

Slika 3.7. *Prikaz ruta korisničkog profila*

```
static async getCurrentUser(req, res) {
  const { authorization } = req.headers;

  if (!authorization) {
    return res.status(401).json({ error: "Authorization required" });
  }
  const token = authorization.split(" ")[1];
  try {
    const { _id } = jwt.decode(token);
    const { firstName, lastName, email, occupation, role, profilePicture } =
      await UserRepository.getUserById(_id);
    res.status(200).json({ _id, firstName, lastName, email, occupation, role, profilePicture  });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}

static async getUserById(req, res) {
  const { user_id } = req.body;
  try {
    const { _id, firstName, lastName, email, occupation, role, profilePicture } =
      await UserRepository.getUserById(user_id);
    res.status(200).json({ _id, firstName, lastName, email, occupation, role, profilePicture });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}
```

Slika 3.8. *Prikaz funkcija korisničkog kontrolera za dohvaćanje i provjeru podataka*

Na slici 3.8. prikazane su funkcije korisničkog kontrolera korištene u opisanim rutama korisničkog profila. Unutar funkcije korisničkog kontrolera *getCurrentUser* obavlja se provjera i dohvaćanje podataka trenutnog korisnika, dok se unutar funkcije *getUserById* izvršava dohvaćanje podataka korisnika koji je vlasnik objavljene usluge. Dohvaćanje podataka korisnika temelji se na pronalaženju dokumenta unutar kolekcije pomoću jedinstvenog identifikacijskog broja. Nakon opisanih modela, kontrolera i ruta korisnika, važno je istaknuti slijedeću grupu modela, kontrolera i ruta koja se odnosi na objavu usluga. CRUD (engl. *Create, Read, Update and Delete*) funkcije primijenjene su za stvaranje, uređivanje, dohvaćanje i brisanje korisničkih

objava poslova te zahtjeva za preuzimanje poslova. Operacije nad poslovima dozvoljene su korisničkoj ulozi *freelancer*, dok su operacije nad zahtjevima za preuzimanje poslova dostupne svim ulogama zbog međusobne interakcije.

```
const Schema = mongoose.Schema;

const serviceSchema = new Schema(
  {
    title: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
    cost: {
      type: Number,
      required: true,
      min: 0,
      default: 0,
    },
    category: {
      type: String,
      required: true,
      lowercase: true,
      enum: ["design", "development", "marketing", "business"],
    },
    user_id: {
      type: String,
      required: true,
    },
    user_name: {
      type: String,
      required: true,
    },
    user_email: {
      type: String,
      required: true,
    },
    picture: {
      public_id: {
        type: String,
        required: true,
      },
      url: {
        type: String,
        required: true,
      },
    },
  },
  { timestamps: true }
);
```

Slika 3.9. *Prikaz modela objave*

Na slici 3.9. prikazan je model objave s raznim atributima poput naziva, opisa, kategorije i cijene usluge. Uz navedene atribute korišteni su korisnički podaci o vlasniku objave poput jedinstvenog identifikacijskog broja, imena i elektroničke pošte.

```
serviceRouter.post("/", ServiceController.addService);
```

Slika 3.10. *Prikaz rute za dodavanje objave*

Ruta prikazana slikom 3.10. koristi se za usmjeravanje zahtjeva za dodavanje objave prema funkciji kontrolera objave *addService* pomoću ranije opisane POST metode. Nakon usmjeravanja zahtjeva, izvršava se obrada zahtjeva unutar navedene funkcije *addService* prikazane slikom 3.11. na način spremanja slike na vanjsku uslugu *cloudinary* i objave sa svim potrebnim podacima prethodno navedenih u opisu modela objave u MongoDB bazu podataka. Nakon uspješnog dodavanja, omogućeno je daljnje dohvaćanje, uređivanje i brisanje objave.

```
static async addService(req, res) {
  const { title, description, cost, category, picture } = req.body;
  const user = req.user;
  try {
    const uploadPicture = await cloudinary.uploader.upload(picture, {
      folder: "services",
    });
    const serviceData = {
      title,
      description,
      cost,
      category,
      picture: {
        public_id: uploadPicture.public_id,
        url: uploadPicture.secure_url,
      },
      user_id: user._id,
      user_name: user.firstName.concat(" ", user.lastName),
      user_email: user.email,
    };
    const service = await ServiceRepository.save(serviceData);
    res.status(200).json(service);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}
```

Slika 3.11. *Prikaz funkcije za dodavanje objava*

```
serviceRouter.get("/", ServiceController.getAllServices);
serviceRouter.get("/:id", ServiceController.getService);
serviceRouter.post("/myServices", ServiceController.getServicesByUserID);
serviceRouter.post("/categories", ServiceController.getServicesByCategory);
```

Slika 3.12. *Prikaz ruta za dohvaćanje objava*

Važno je istaknuti rute za dohvaćanje objavljenih usluga prikazane slikom 3.12. Za ostvarivanje navedenih ruta korištene su GET i POST metode koje su opisane prethodno u ovom poglavlju. Slijedno prateći rute na slici 3.12., ruta za dohvaćanje svih objavljenih usluga pomoću GET metode preusmjerava zahtjev za dohvaćanje objavljenih usluga prema funkciji *getAllServices* unutar kontrolera objave. Funkcija *getAllServices* prikazana na slici 3.13. obavlja dohvaćanje svih objavljenih usluga preko komunikacije s bazom podataka pomoću istoimene funkcije *getAllServices* unutar repozitorija objava.

```
static async getAllServices(req, res) {
  try {
    const { services } = await ServiceRepository.getAllServices();
    res.status(200).json(services);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}
```

Slika 3.13. *Prikaz funkcije za dohvaćanje svih objava*

Ruta za dohvaćanje pojedinačnih objavljenih usluga usmjerava zahtjev prema funkciji *getService* unutar kontrolera objava. Ovakva vrsta zahtjeva obavlja se pomoću GET metode te se pri slanju zahtjeva prosljeđuje jedinstveni identifikacijski broj. Navedena funkcija *getService* prikazana na slici 3.14. provodi daljnju obradu zahtjeva pomoću poslanog identifikacijskog broja na temelju provjere ispravnosti identifikacijskog broja. Nakon uspješne provjere, obavlja se dohvaćanje objavljene usluge s istim identifikacijskim brojem i prosljeđuje na klijentsku stranu u JSON (engl. *JavaScript Object Notation*) formatu.

Slika 3.14. *Prikaz funkcije za dohvaćanje pojedinačne objave*

Ruta za dohvaćanje vlastitih korisničkih objavljenih usluga je ostvarena pomoću POST metode i služi za preusmjeravanje poslanog zahtjeva prema funkciji *getServicesByUserID* unutar kontrolera objave. Poslani zahtjev za dohvaćanje vlastitih korisničkih objavljenih usluga sadržava parametar jedinstveni identifikacijski broj. Funkcija *getServicesByUserID* prikazana na slici 3.15. provodi obradu zahtjeva na način dohvaćanja svih korisničkih objavljenih usluga s istim jedinstvenim identifikacijskim brojem i prosljeđivanju istih na klijentsku stranu u JSON formatu.

Slika 3.15. *Prikaz funkcije za dohvaćanje vlastitih korisničkih objava*

Posljednja ruta za dohvaćanje objavljenih usluga prema kategoriji ostvarena je također pomoću POST metode. Služi za preusmjeravanje zahtjeva prema funkciji *getServicesByCategory* unutar kontrolera objave. Zahtjev sadržava jedan parametar koji se odnosi na naziv kategorije objave. Na temelju parametra zahtjeva, unutar funkcije *getServicesByCategory* prikazane slikom 3.16. provodi se obrada zahtjeva na način pronalaženja i dohvaćanja svih objavljenih usluga istog naziva kategorije. Nakon uspješnog pronalaženja i dohvaćanja, prosljeđuju se sve objavljene usluge s istom kategorijom prema klijentskoj strani.

```
static async getServicesByCategory(req, res) {
  const { category } = req.body;
  try {
    const { services } = await ServiceRepository.getServicesByCategory(
      category
    );
    res.status(200).json(services);
  } catch (error) {
    res.status(404).json({ error: "Service not found" });
  }
}
```

Slika 3.17. *Prikaz funkcije za dohvaćanje objava usluga prema kategoriji*

Nakon ruta i funkcija kontrolera za dohvaćanje, važno je istaknuti rute i funkcije kontrolera za uređivanje i brisanje objavljenih usluga. Ruta za uređivanje objavljenih usluga ostvarena je pomoću PATCH metode. PATCH metoda omogućava slanje zahtjeva za uređivanje sadržaja. Moguće je uređivati dio ili cijeli sadržaj. Ruta za uređivanje objavljenih usluga prikazana slikom 3.18. preusmjerava zahtjev za uređivanje sadržaja prema funkciji *updateService* unutar kontrolera objava. Poslani zahtjev sadržava jedinstveni identifikacijski broj kao parametar pomoću kojeg se provodi pronalaženje objavljene usluge i omogućava uređivanje postojećeg sadržaja objavljenih usluga. Funkcija *updateService* prikazana na slici 3.19. obavlja provjeru ispravnosti poslanog identifikacijskog broja te nakon pronalaska objavljene usluge pomoću ispravnog identifikacijskog broja mijenja poslani sadržaj s postojećim sadržajem usluge. Također, vraća promijenjenu objavljenu uslugu na klijentsku stranu u JSON formatu.

```
serviceRouter.patch("/:id", ServiceController.updateService);
```

Slika 3.18. *Prikaz rute za uređivanje objave*

```
static async updateService(req, res) {
  const { id } = req.params;
  const data = req.body;
  if (!mongoose.Types.ObjectId.isValid(id)) {
    return res.status(404).json({ error: "Service not found" });
  }
  try {
    const service = await ServiceRepository.updateService(data, id);
    res.status(200).json(service);
  } catch (error) {
    res.status(404).json({ error: "Service not found" });
  }
}
```

Slika 3.19. *Prikaz funkcije za uređivanje objave*

Ruta za brisanje objavljenih usluga izrađena je pomoću DELETE metode. DELETE metoda omogućava slanje zahtjeva za brisanje sadržaja. Na slici 3.20. prikazana je ruta za brisanje objavljenih usluga koja preusmjerava zahtjev za brisanje sadržaja prema funkciji *deleteService*. Zahtjev za brisanje sadržaja sadrži jedinstveni identifikacijski broj kao parametar po kojem se provodi daljna obrada. Brisanje sadržaja unutar navedene funkcije *deleteService* prikazane slikom 3.21. obavlja se na temelju poslanog jedinstvenog identifikacijskog broja na način provođenja provjere postojeće objave s istim identifikacijskim brojem, brisanjem pripadajuće slike objavljene usluge s vanjske usluge *cloudinary* te brisanjem objave u cijelosti.

```
serviceRouter.delete("/:id", ServiceController.deleteService);
```

Slika 3.20. *Prikaz rute za brisanje objava*

```
static async deleteService(req, res) {
  const { id } = req.params;
  if (!mongoose.Types.ObjectId.isValid(id)) {
    return res.status(404).json({ error: "Service not found" });
  }
  try {
    const service = await ServiceRepository.deleteService(id);
    await cloudinary.uploader.destroy(service.picture.public_id);
    res.status(200).json(service);
  } catch (error) {
    res.status(404).json({ error: "Service not found" });
  }
}
```

Slika 3.21. *Prikaz funkcije za brisanje objava*

```
serviceRouter.post("/searchByTitle", ServiceController.getServicesByTitle);
```

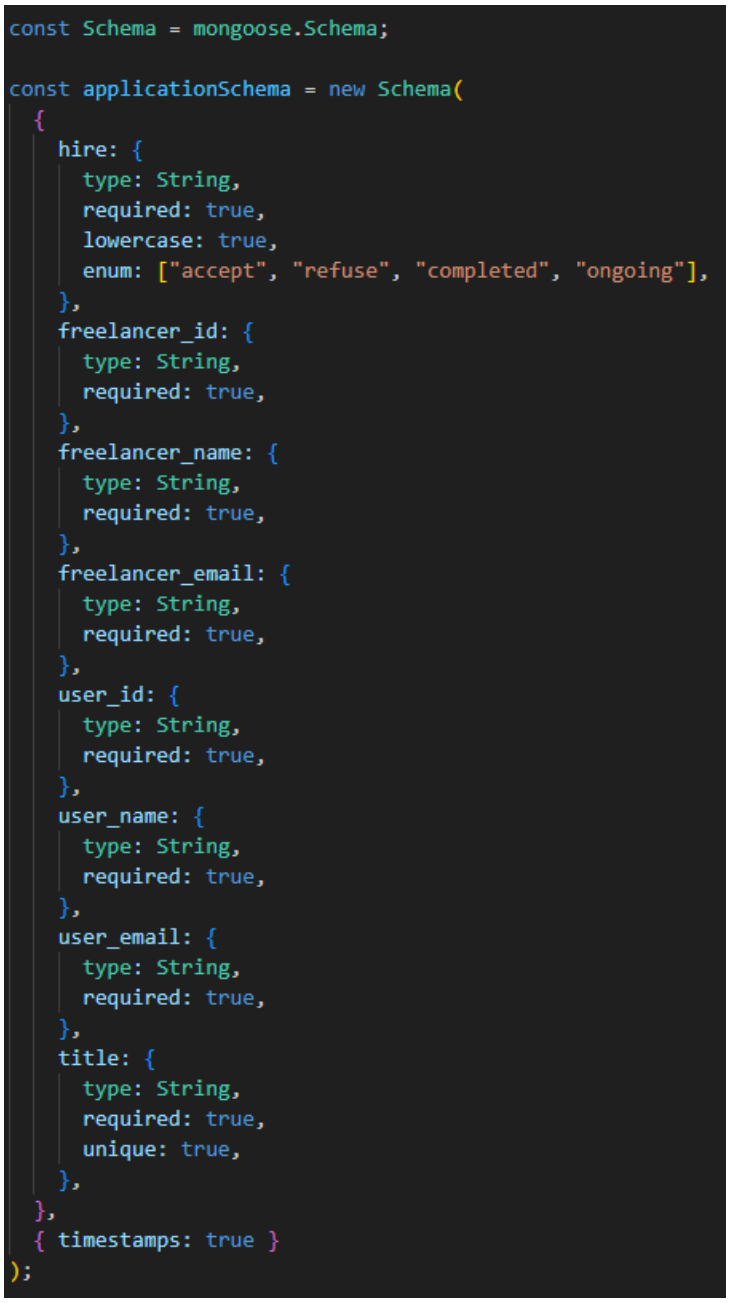
Slika 3.22. *Prikaz rute za pretraživanje*

Ruta za pretragu objavljenih usluga prikazana slikom 3.22. služi za preusmjeravanje zahtjeva za pretragu objavljenih usluga prema funkciji *getServicesByTitle* unutar kontrolera objave. Ruta je ostvarena pomoću opisane POST metode. Zahtjev sadržava ključnu riječ, odnosno naslov usluge. Funkcija *getServicesByTitle* prikazana na slici 3.23. provodi pretragu usluga na temelju poslanog parametra odnosno ključne riječi. Nakon pronalaska svih objavljenih usluga koje sadrže ključnu riječ, obavlja se slanje pronađenih usluga na klijentsku stranu u JSON formatu.

```
static async getServicesByTitle(req, res) {
  const { title } = req.body;
  try {
    const { services } = await ServiceRepository.getServicesByTitle(title);
    res.status(200).json(services);
  } catch (error) {
    res.status(404).json({ error: "Service not found" });
  }
}
```

Slika 3.23. *Prikaz kontrolerske funkcije za pretraživanje poslovnih objava*

Nakon opisanih modela, ruta i kontrolera za objavu usluga, važno je napomenuti preuzimanje poslova odnosno usluga. U ovom slučaju, funkcije kontrolera s CRUD operacijama za preuzimanje poslova identične su prethodno opisanim CRUD funkcijama kontrolera za objavu. S obzirom na navedeno, bitno je istaknuti model i rute za preuzimanje poslova.



```
const Schema = mongoose.Schema;

const applicationSchema = new Schema(
  {
    hire: {
      type: String,
      required: true,
      lowercase: true,
      enum: ["accept", "refuse", "completed", "ongoing"],
    },
    freelancer_id: {
      type: String,
      required: true,
    },
    freelancer_name: {
      type: String,
      required: true,
    },
    freelancer_email: {
      type: String,
      required: true,
    },
    user_id: {
      type: String,
      required: true,
    },
    user_name: {
      type: String,
      required: true,
    },
    user_email: {
      type: String,
      required: true,
    },
    title: {
      type: String,
      required: true,
      unique: true,
    },
  },
  { timestamps: true }
);
```

Slika 3.24. *Prikaz modela zahtjeva za preuzimanje poslova*

Na slici 3.24. prikazan je model zahtjeva za preuzimanjem poslova koji sadržava razne atribute poput statusa zahtjeva, korisničkih podataka o objavitelju i izvršitelju posla odnosno *freelanceru* i korisničkih podataka o kupcu usluge odnosno *useru.* Atributi o korisničkim podacima su jedinstveni identifikacijski broj, ime i elektronička pošta. Uz navedene atribute, model sadržava i naslov objavljene usluge koja se obavlja.

```
applicationRouter.post("/", ApplicationController.addApplication);

applicationRouter.get(
  "/userApplications",
  ApplicationController.getApplicationsByUserID
);

applicationRouter.get(
  "/freelancerApplications",
  ApplicationController.getApplicationsByFreelancerID
);
applicationRouter.patch("/:id", ApplicationController.updateApplication);

applicationRouter.delete("/:id", ApplicationController.deleteApplication);
```

Slika 3.25. *Prikaz ruta zahtjeva*

Rute za slanje zahtjeva za preuzimanje poslova prikazane su na slici 3.25. te su slične prethodno opisanim rutama za objave poslova u ovom poglavlju. Jedina je razlika u funkcionalnosti ruta za dohvaćanje zahtjeva koje služe za dohvaćanje zahtjeva za preuzimanje poslova na temelju jedinstvenog identifikacijskog broja korisnika ovisno o ulozi. Korištena je opisana GET metoda. Ukoliko se odnosi na korisnika s ulogom *user*, provodi se ruta za dohvaćanje zahtjeva koja prosljeđuje zahtjev funkciji *getApplicationsByUserID* unutar kontrolera zahtjeva. Funkcija *getApplicationsByUserID* obavlja pronalazak i dohvaćanje zahtjeva za preuzimanje poslova na isti način kao prethodno opisano dohvaćanje objavljene usluge pomoću jedinstvenog identifikacijskog broja. Ako se odnosi na korisnika s ulogom *freelancer*, provodi se ruta za dohvaćanje zahtjeva koja preusmjerava zahtjev za preuzimanje poslova funkciji *getApplicationsByFreelancerID* unutar kontrolera zahtjeva čija je funkcionalnost identična funkciji *getApplicationsByUserID*, ali je razlika u predanom parametru.

3.2. Klijentski dio aplikacije

Za izradu klijentskog dijela aplikacije korištena je React tehnologija. React se koristi zbog brzog i dinamičkog razvoja koji uvelike utječe na izvođenje web aplikacije. S obzirom na način korištenja bitno je napomenuti predložak JSX, odnosno proširenje za pisanje JavaScripta unutar Reacta koje nudi mogućnost pisanja direktnih HTML elemenata kao način strukturiranja povratnih elemenata komponenti [10]. Također, unutar klijenta je važno istaknuti izrađene React komponente poput korisničke prijave, korisničkog profila te komponente za objavljivanje, preuzimanje i pretraživanje usluga odnosno poslova.

```
return (
  <section className="login-form">
    <div className="login-form__wrapper">
      <h2>Login</h2>
      <form onSubmit={handleSubmit}>
        <input
          className="email"
          type="text"
          id="email"
          name="email"
          placeholder="Email"
          ref={emailRef}
          required
        />
        <input
          className="password"
          type="password"
          id="password"
          name="password"
          placeholder="Password"
          autoComplete="on"
          ref={passwordRef}
          required
        />
        <button type="submit">Log in</button>
        {error && <div className="error"> {error} </div>}
        <p>
          Need an account? <Link to="/register" className="form__link">Register</Link>
        </p>
      </form>
    </div>
  </section>
);
```

Slika 3.26. *Prikaz komponente za korisničku prijavu*

Na slici 3.26. prikazana je komponenta za korisničku prijavu koja sadržava formu. Za slanje zahtjeva sa klijenta prema poslužitelju korišten je *axios* paket unutar Reacta. *Axios* poziv sadržava putanju do određene rute na poslužitelju, tijelo u kojem se šalje sadržaj i dodatnih opcija poput određivanja vrste sadržaja. Nakon ispravnog unosa, provodi se *axios* poziv s metodom POST pomoću kojeg se šalje sadržaj prema poslužiteljskoj strani u obliku JSON formata.

```
return (
  <>
    <div className="user">
      <div className="user__wrapper">
        <h2>Account Details</h2>
        <div className="user__details">
          <img
            className="user__picture"
            src={user.profilePicture.url}
            alt="Profile Picture"
          />
          <p>
            First Name: <span>{user.firstName}</span>
          </p>
          <p>
            Last Name: <span>{user.lastName}</span>
          </p>
          {is_profile_details && (
            <p>
              Email:{" "}
              <a href={`mailto:${user.email}`} className="text-italic">
                {user.email}
              </a>
            </p>
          )}
          {!is_profile_details && (
            <p>
              Email: <span>{user.email}</span>
            </p>
          )}
          {user.role === "Freelancer" && (
            <p>
              Occupation: <span>{user.occupation}</span>
            </p>
          )}
          <p>
            Role: <span>{user.role}</span>
          </p>
        </div>
      </div>
    </div>
  </>
);
};
```

Slika 3.27. *Prikaz komponente korisničkog profila*

Na slici 3.27. prikazana je komponenta korisničkog profila čijim se izvršenjem koda obavlja prikaz u korisničke podatke. Za dohvaćanje korisničkih podataka poput *user.firstName* i *user.lastName* korišten je *axios* poziv s metodom GET i pripadajućom putanjom prema poslužitelju za dohvaćanje korisničkih podataka opisanom u ovom poglavlju.

```
return (
  <section className="service-form">
    <div className="service-form__wrapper">
      <h2>Post New Service</h2>
      <form onSubmit={handleSubmit}>
        <label htmlFor="picture">Choose Picture:</label>
        <input className="picture" type="file" name="picture"
          accept="image/*"
          onChange={async (e) => {
            await setFileToBase64(e.target.files[0]);
          }}
          required
        />
        <input className="title" type="text" name="title" placeholder="Title" ref={titleRef}
          required
        />
        <input
          className="description" type="text" name="description" placeholder="Description" ref={descriptionRef}
          required
        />
        <input className="cost" type="text" name="cost" placeholder="Cost" ref={costRef}
          required
        />
        <div
          className="service-form__radio-buttons"
          onChange={(e) => {
            setCategory(e.target.value);
          }}
        >
          <input className="category" type="radio" id="categoryDesign" name="category" value={"Design"}
            required
          />
          <label htmlFor="categoryDesign">Design</label>

          <input className="category" type="radio" id="categoryDevelopment" name="category" value={"Development"}
            required
          />
          <label htmlFor="categoryDevelopment">Development</label>

          <input className="category" type="radio" id="categoryMarketing" name="category" value={"Marketing"}
            required
          />
          <label htmlFor="categoryMarketing">Marketing</label>

          <input className="category" type="radio" id="categoryBusiness" name="category" value={"Business"}
            required
          />
          <label htmlFor="categoryBusiness">Business</label>
        </div>
        <button type="submit">Post Service</button>
        {error && <div className="error"> {error} </div>}
      </form>
    </div>
  </section>
);
```

Slika 3.28. *Prikaz komponente za objavljivanje*

Izvršenjem koda komponente za objavljivanje usluga prikazanog slikom 3.28. obavlja se prikaz forme za objavljivanje usluga. Nakon ispravnog unosa, izvršava se *axios* poziv s metodom POST i sadržajem u obliku JSON formata prema poslužitelju. Kako bi komunikacija preko zahtjeva za preuzimanje poslova bila što jasnija, važno je napomenuti različita stanja unutar dijelova komponente zahtjeva za korisničke uloge. Dio komponente zahtjeva za preuzimanjem poslova za korisničku ulogu *user* prikazan je slikom 3.29. čijim se izvršenjem koda obavlja komunikacija u ovisnosti o statusu zahtjeva. Status zahtjeva može biti *ongoing*, *accept*, *refuse* i *completed*.

Nakon svake promjene statusa, provodi se *axios* poziv s metodom POST i sadržajem. U ovom slučaju je sadržaj promijenjeni status zahtjeva.

```
{currentUser.role == "User" && (
  <div className="application__details">
    <h3>{application.title}</h3>
    {application.hire == "ongoing" && (
      <>
        <p>
          waiting on{" "}
          <span className="text-italic">
            {application.freelancer_name}
          </span>{" "}
          to accept or decline application
        </p>
        <button
          onClick={() => deleteApplication(application._id)}
        >
          Delete
        </button>
      </>
    )}
    {application.hire == "accept" && (
      <>
        <p>
          <span className="text-italic">
            {application.freelancer_name}
          </span>{" "}
          has accepted your application, please contact
          freelancer on this email{" "}
          <a
            href={`mailto:${application.freelancer_email}`}
            className="text-italic"
          >
            {application.freelancer_email}
          </a>
        </p>
      </>
    )}
    {application.hire == "refuse" && (
      <>
        <p>
          <span className="text-italic">
            {application.freelancer_name}
          </span>{" "}
          has refused your application, please delete this
          application
        </p>
        <div className="application__buttons">
          <button
            onClick={() => deleteApplication(application._id)}
          >
            Delete
          </button>
        </div>
      </>
    )}
    {application.hire == "completed" && (
      <>
        <p>
          <span className="text-italic">
            {application.freelancer_name}
          </span>{" "}
          has completed this service, if something went wrong,
          please contact him again on{" "}
          <a
            href={`mailto:${application.freelancer_email}`}
            className="text-italic"
          >
            {application.freelancer_email}
          </a>
        </p>
```

Slika 3.29. *Prikaz dijela komponente zahtjeva za usera*

```
{currentUser.role == "Freelancer" && (
  <div className="application__details">
    <h3>{application.title}</h3>
    {application.hire == "ongoing" && (
      <>
        <p>
          <span className="text-italic">
            {application.user_name}
          </span>{" "}
          wants to hire you for this service
        </p>
        <div className="application__buttons">
          <button onClick={() => acceptApplication(application)}> Accept</button>
          <button onClick={() => declineApplication(application)}> Decline</button>
        </div>
      </>
    )}
    {application.hire == "accept" && (
      <>
        <p>
          You have accepted{" "}
          <span className="text-italic">
            {application.user_name}
          </span>{" "}
          application, please contact user on this email{" "}
          <a
            href={`mailto:${application.user_email}`}
            className="text-italic"
          >
            {application.user_email}
          </a>
          . After completing this service, if user has paid
          you, please mark this application as Completed
        </p>
        <button onClick={() => completeApplication(application)}> Completed </button>
      </>
    )}
    {application.hire == "refuse" && (
      <>
        <p>
          You have refused{" "}
          <span className="text-italic">
            {application.user_name}
          </span>{" "}
          application, please delete this application
        </p>
        <div className="application__buttons">
          <button onClick={() => deleteApplication(application._id)}> Delete </button>
        </div>
      </>
    )}
    {application.hire == "completed" && (
      <>
        <p>
          You have completed this service and{" "}
          <span className="text-italic">
            {application.user_name}
          </span>{" "}
          has paid you in full, if something went wrong,
          please contact user on this email{" "}
          <a
            href={`mailto:${application.user_email}`}
            className="text-italic"
          >
            {application.user_email}
          </a>{" "}
          or undo application status
        </p>
        <button onClick={() => acceptApplication(application)} className="text-center"> Undo </button>
      </>
    )}
  </div>
```

Slika 3.30. *Prikaz dijela komponente zahtjeva za freelancera*

Drugi dio komponente zahtjeva za preuzimanjem poslova odnosi se na korisničku ulogu *freelancer* prikazano slikom 3.30. čijim se izvršenjem koda obavlja komunikacija identična prethodno opisanoj komunikaciji korisnika s ulogom *user*. Korisniku s ulogom freelancer dodatno se omogućuje prihvaćanje i odbijanje zahtjeva za zapošljavanjem odnosno preuzimanjem posla. Nakon opisanih komponenti zahtjeva za preuzimanje posla, bitno je istaknuti komponentu za pretraživanje koja je dostupna svim prijavljenim korisnicima radi lakšeg pronalaženja potrebnih *freelancer* usluga. Pretraživanje objavljenih poslova izvršava se preko ključne riječi.

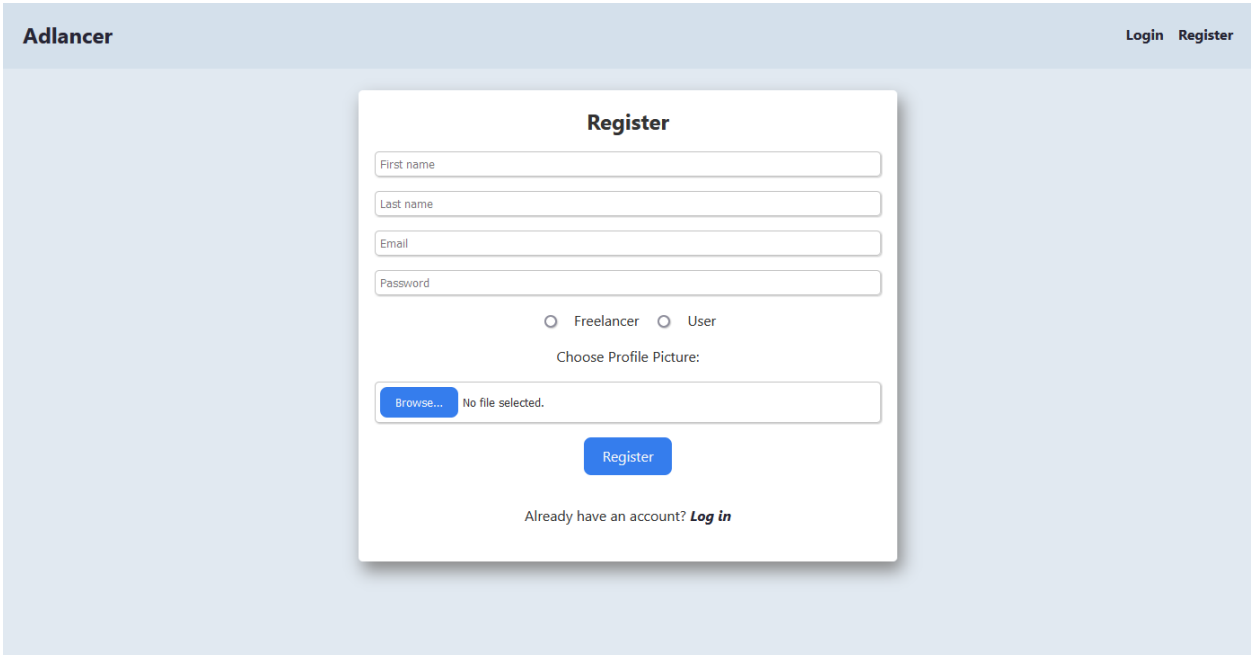
```
return (
  <>
    <section className="search">
      <div className="search__wrapper">
        {currentUser && (
          <div className="search__box">
            <i className="" onClick={() => search()}>
              <img
                width="30px"
                height="30px"
                src={searchLogo}
                alt="Search Icon"
              />
            </i>
            <input
              type="text"
              placeholder="Search"
              id="search"
              onKeyDown={handleKeyDown}
            />
          </div>
        )}
      </div>
    </section>
  </>
);
```

Slika 3.31. *Prikaz komponente za pretragu*

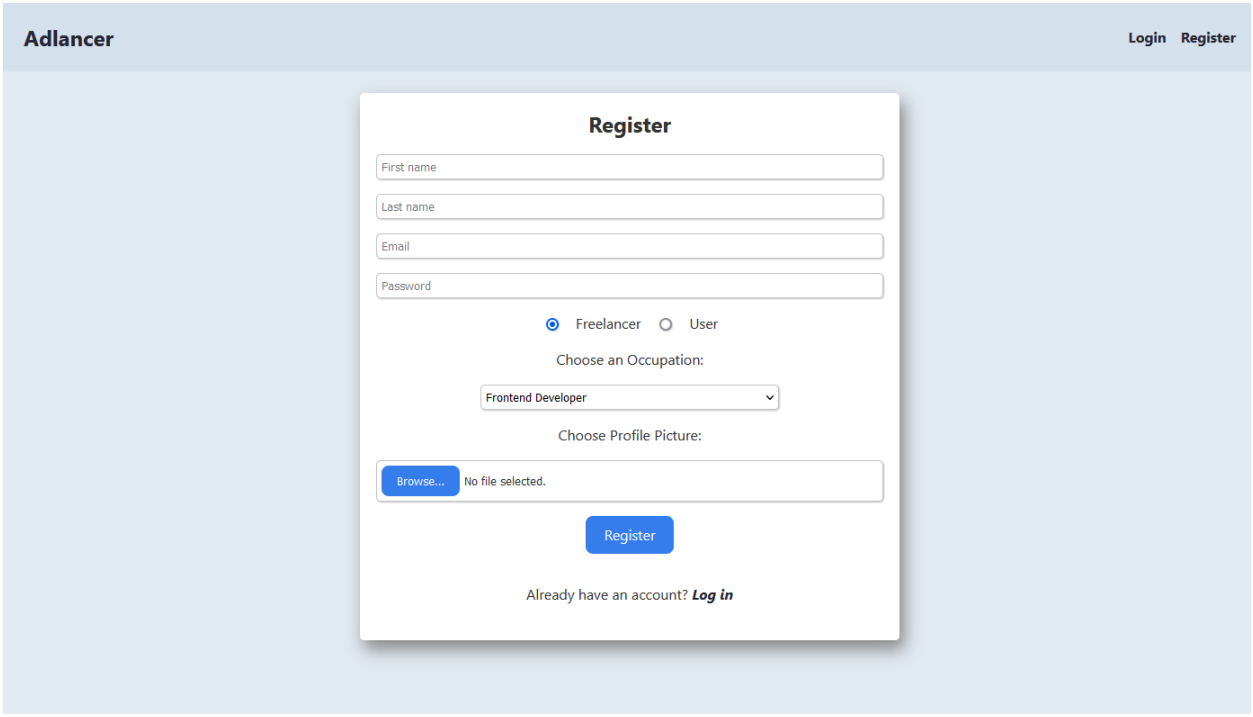
Na slici 4.31. prikazana je komponente za pretragu poslovnih objava. Nakon ispravnog unosa, poziva se funkcija *search* unutar koje se obavlja *axios* poziv s metodom POST i sadržajem odnosno ključnom riječi u JSON formatu. Pretraga se vrši identificiranjem naslova postojećih objavljenih usluga provjerom unesene ključne riječi.

4. PRIKAZ RADA WEB APLIKACIJE

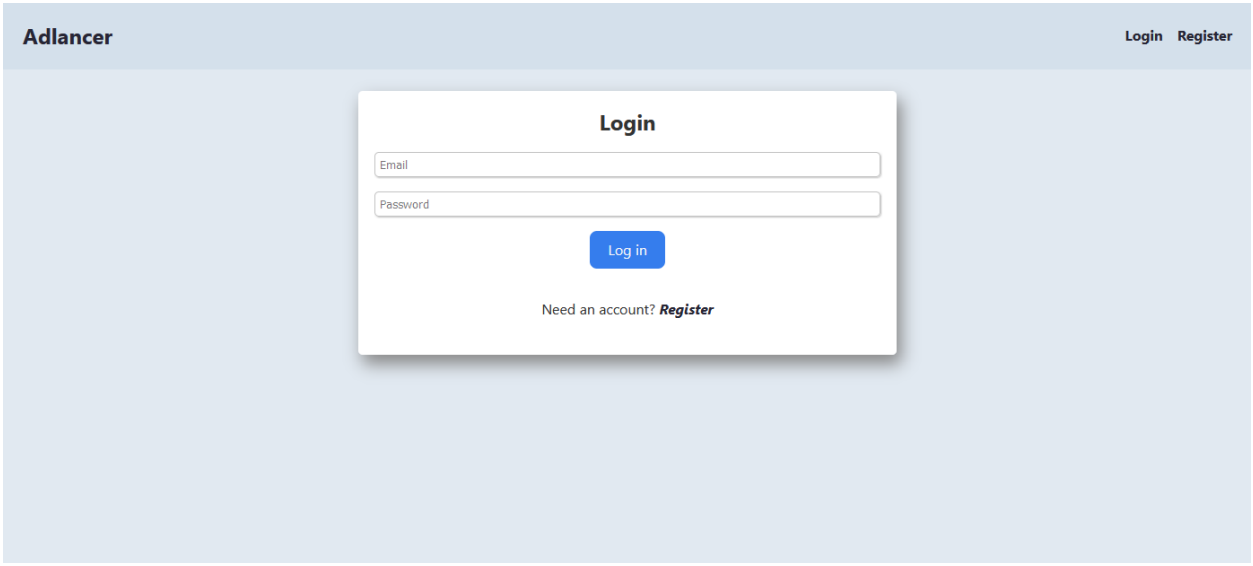
U ovom poglavlju je prikazan način korištenja i funkcioniranja izrađene web aplikacije. Prikaz načina korištenja i funkcioniranja web aplikacije odnosi se na prolazak kroz aplikaciju slijedno prateći prethodno opisane funkcionalnosti unutar aplikacije. Slike unutar poglavlja prikazuju se za obje navedene uloge korisnika. Kako bi korisnik mogao pristupiti web aplikaciji potrebno je napraviti korisnički račun putem ponuđene forme za registraciju. Odabirom uloge *freelancer* prikazuje se dodatna opcija za odabir područja interesa rada poput *Frontend Developer* i *Backend Developer*. Ukoliko je korisnik već registriran postoji forma za prijavu kojoj se može pristupiti klikom na gumb *Log in*. Na slikama 4.1., 4.2. i 4.3. prikazane su navedene forme. U formi za registraciju prikazani su unosi za ime, prezime, elektroničku poštu, zaporku, ulogu i slike, dok su u formi za prijavu prikazani unosi elektroničke pošte i zaporke.



Slika 4.1. *Prikaz forme za registraciju*

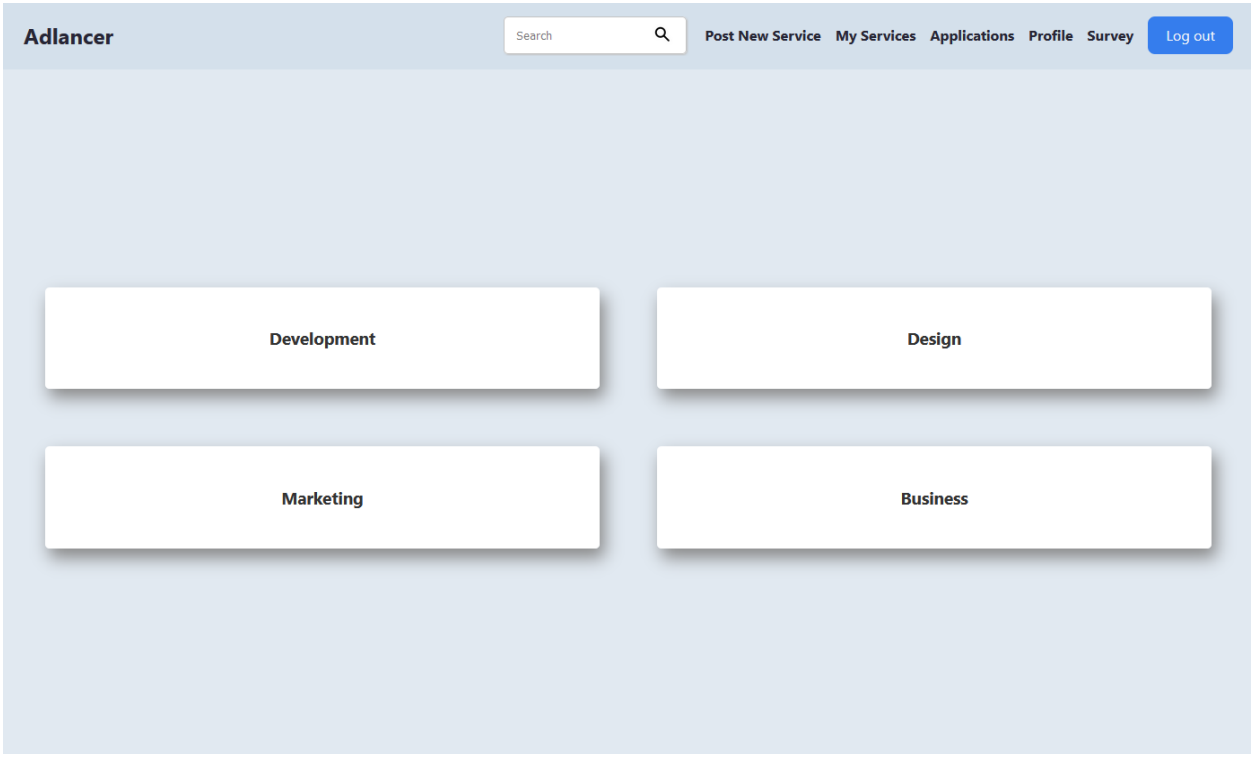


Slika 4.2. *Prikaz dodatne opcije odabirom uloge freelancer*

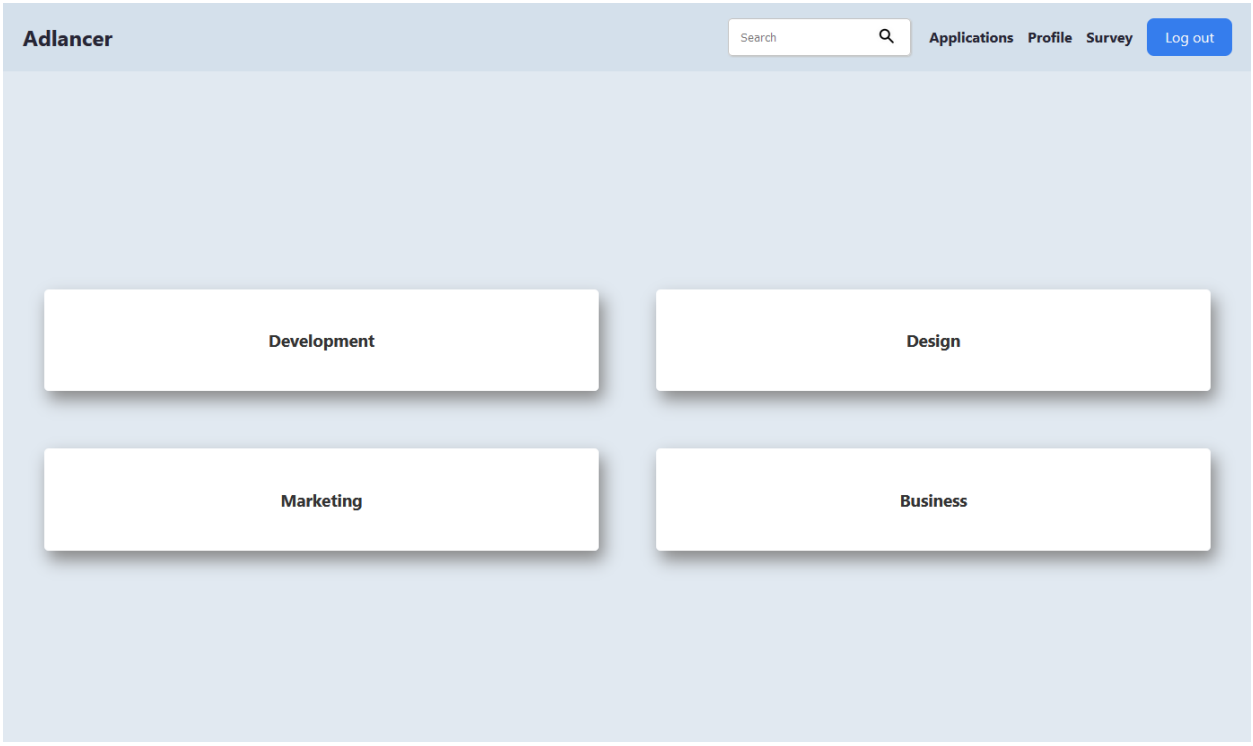


Slika 4.3. *Prikaz forme za prijavu*

Nakon uspješne prijave ili registracije, pojavljuje se naslovna stranica na kojoj se nalaze kategorije objava, tražilica za pretraživanje, uvid u profil, pristup zahtjevima za preuzimanje poslova i upitnik. Slikama 4.4. i 4.5, prikazana je naslovna stranica ovisno o odabranoj ulozi, odnosno korisniku s ulogom *freelancera* uz navedene funkcionalnosti na naslovnoj stranici omogućeno je i stvaranje te pregledavanje vlastitih objava.



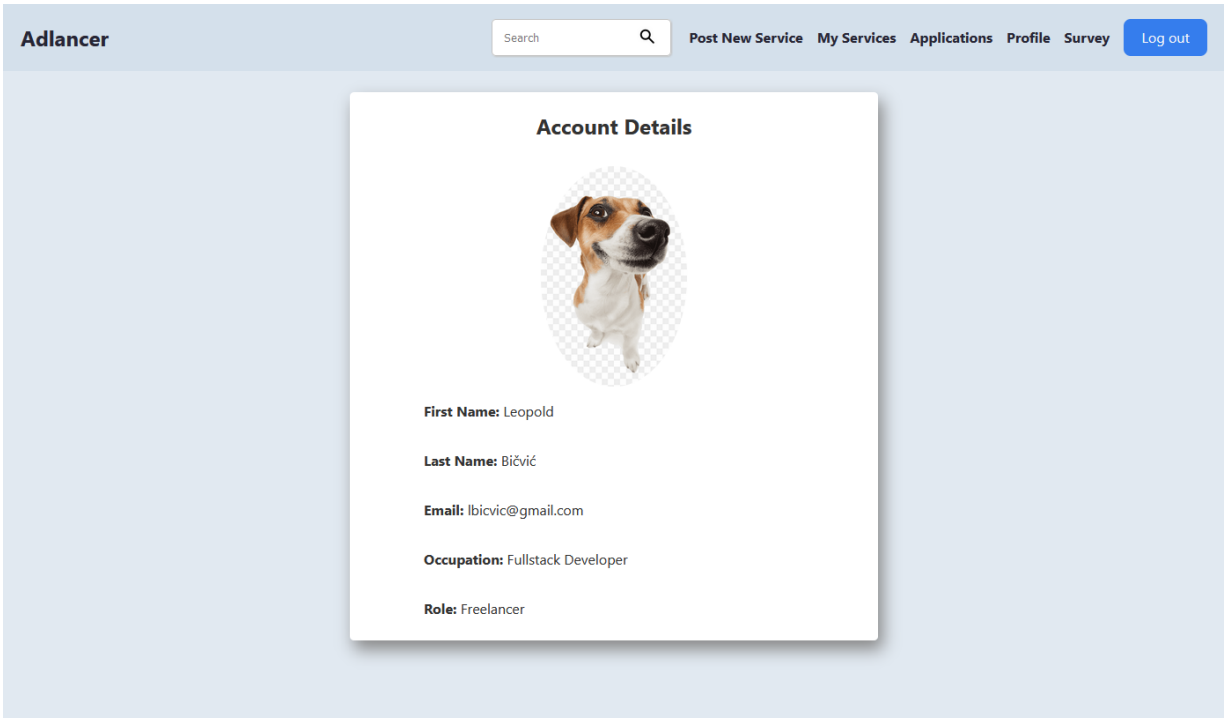
Slika 4.4. *Prikaz naslovne stranice za korisnike s ulogom freelancer*



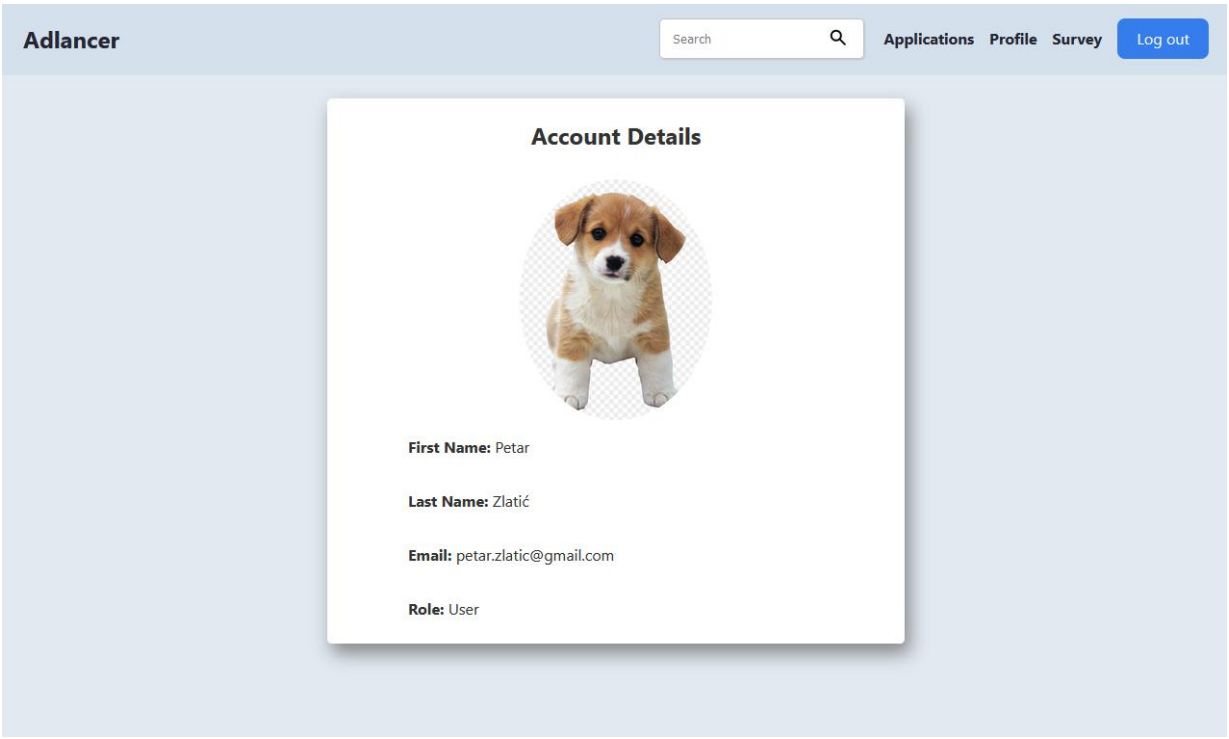
Slika 4.5. *Prikaz naslovne stranice za korisnike s ulogom user*

Na slikama 4.6. i 4.7. prikazan je pogled profila korisnika s ulogama *freelancer* i *user*. Na profilu je vidljivo ime i prezime korisnika, elektronička pošta i vrsta uloge. Jedina razlika u pogledu

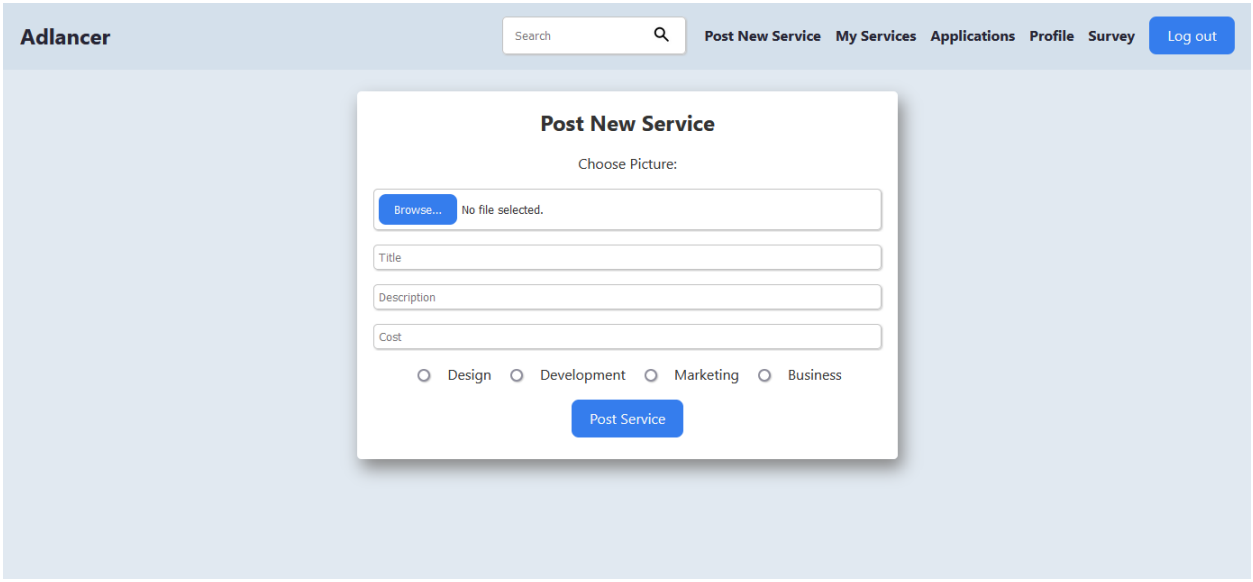
profila je dodatna informacija o poslovnom zanimanju koju sadržava korisnik s ulogom *freelancera.*



Slika 4.6. *Prikaz profila korisnika s ulogom freelancera*

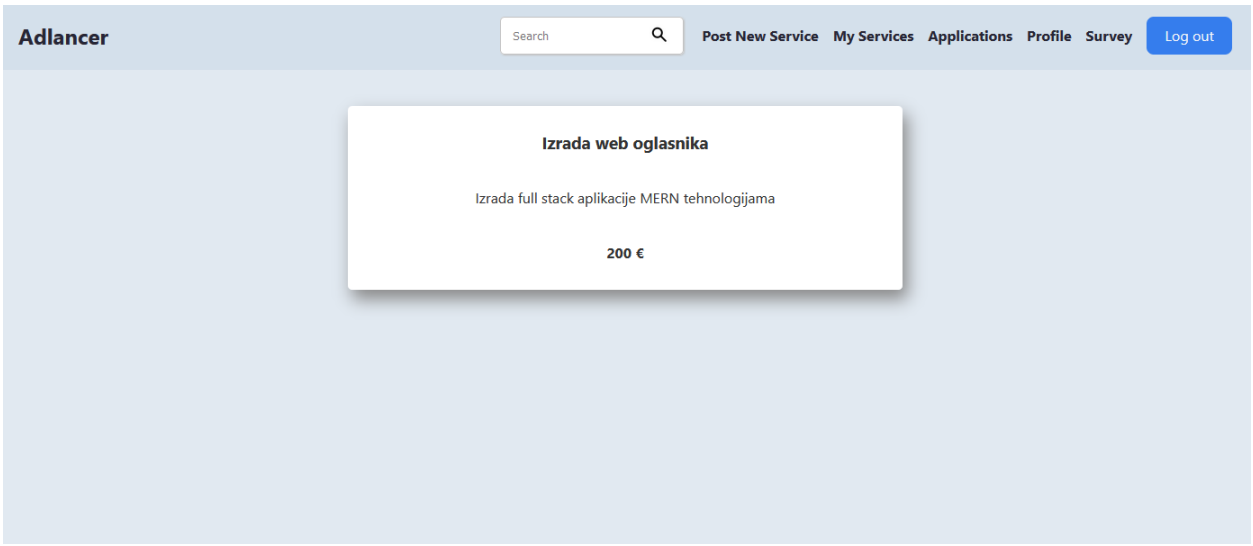


Slika 4.7. *Prikaz profila korisnika s ulogom user*



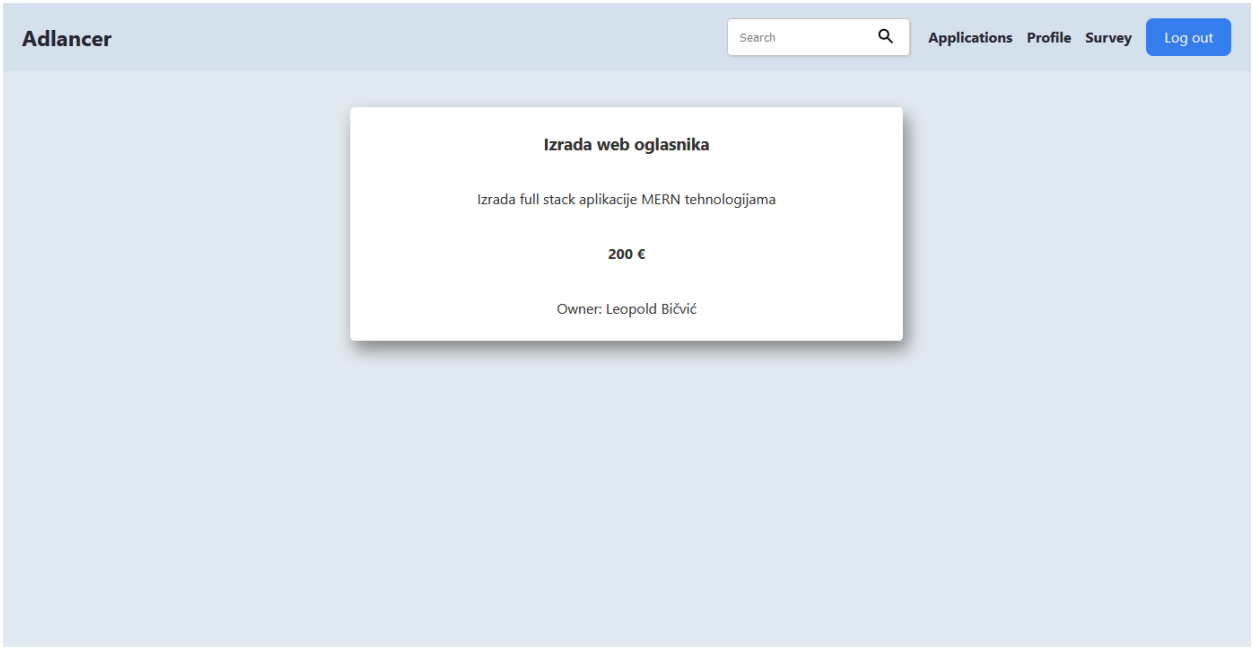
Slika 4.8. *Prikaz forme za postavljanje objava poslova*

Na slici 4.8. prikazana je forma za postavljanje objava poslova. Forma sadrži unose za sliku, naslov, opis, cijenu i kategoriju usluge odnosno posla. Nakon što korisnik s ulogom *freelancer* ispravno popuni formu, novonastala objava nalazi se unutar vlastitih objava korisnika i pripadnoj kategoriji.



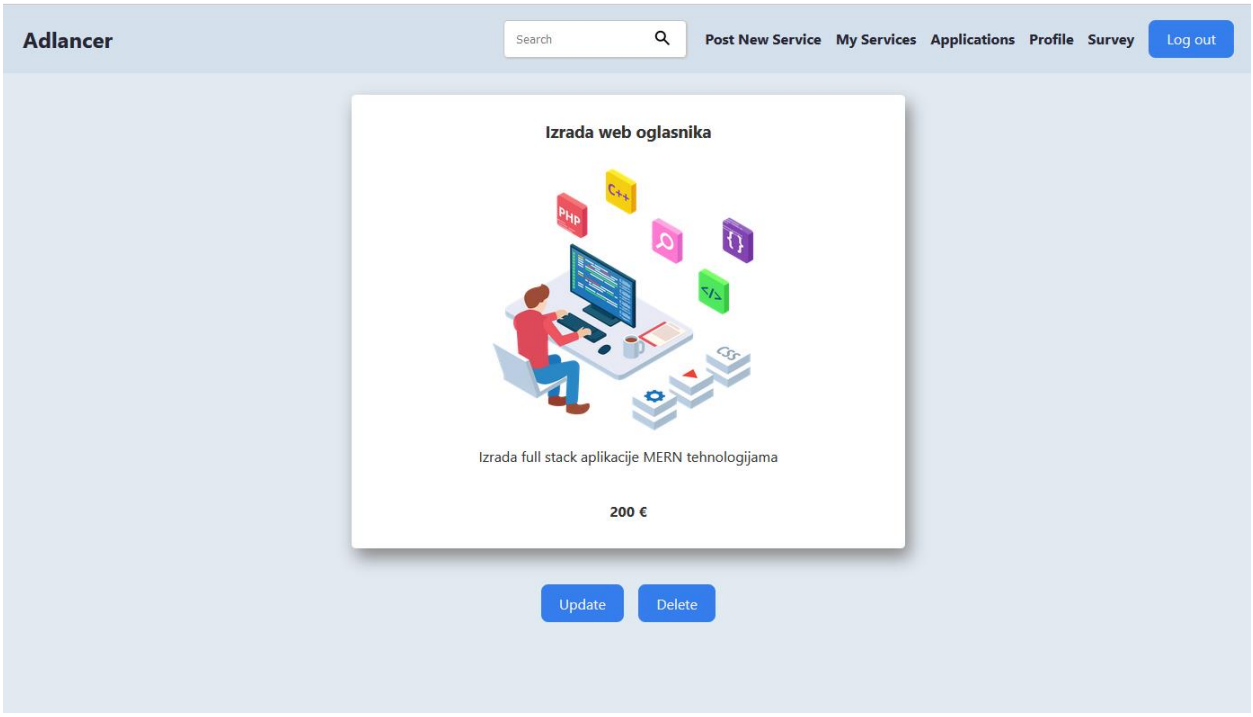
Slika 4.9. *Prikaz vlastitih objava poslova unutar pripadne kategorije*

Na slikama 4.9. i 4.10 prikazana je objavljena usluga na kojoj se nalazi naslov, opis i cijena. Razlika u prikazima je ta što je korisniku s ulogom *user* prikazana informacija o vlasniku objave.

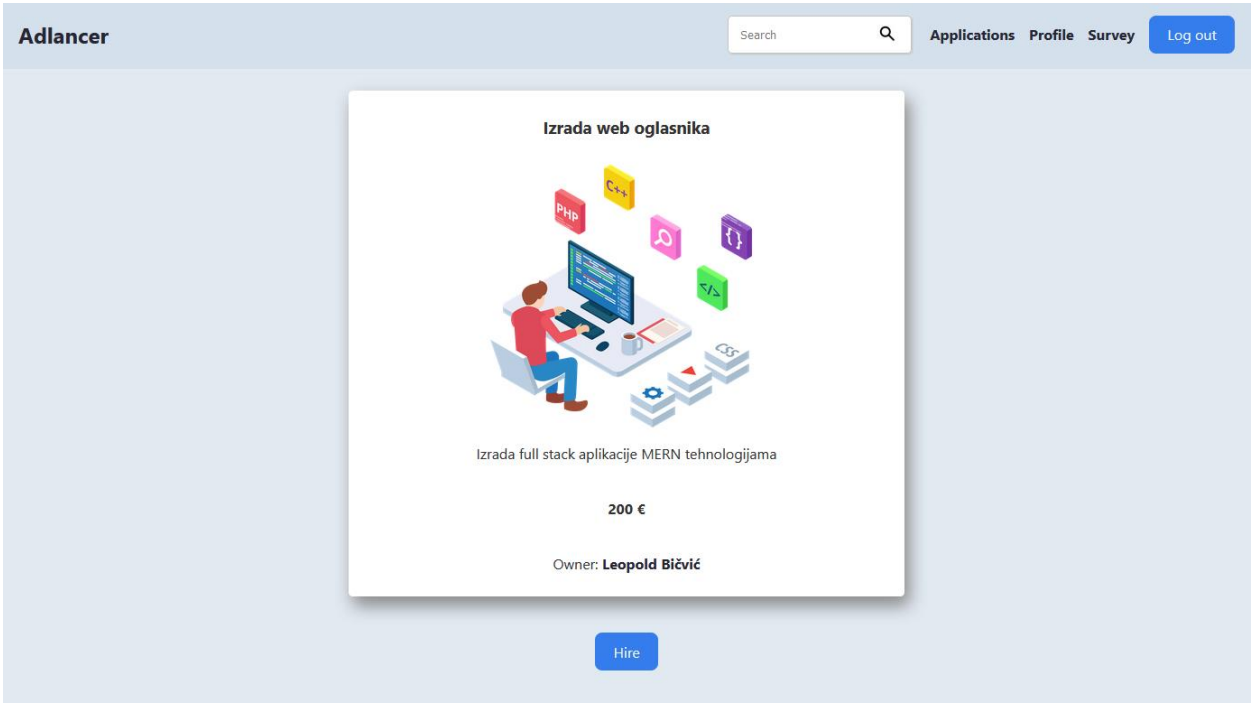


Slika 4.10. *Prikaz freelancer objave*

Na slikama 4.11. i 4.12. prikazani su detalji *freelancer* objave koja sadržava naslov, sliku, opis i cijenu. Korisnik s ulogom *freelancer* kao vlasnik usluge ima dodatne mogućnosti poput uređivanja i brisanja objave, dok je korisniku s ulogom *user* omogućen uvid u profil vlasnika objavljene usluge klikom na ime i prezime koje se nalazi na dnu objave i zapošljavanje vlasnika usluge odnosno freelancera.

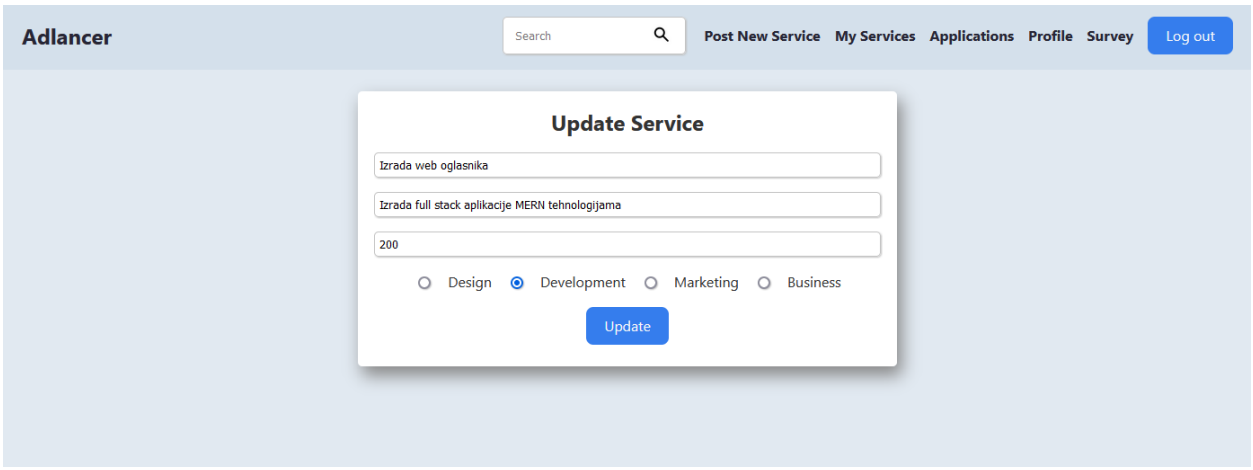


Slika 4.11. *Prikaz detalja freelancer objave*

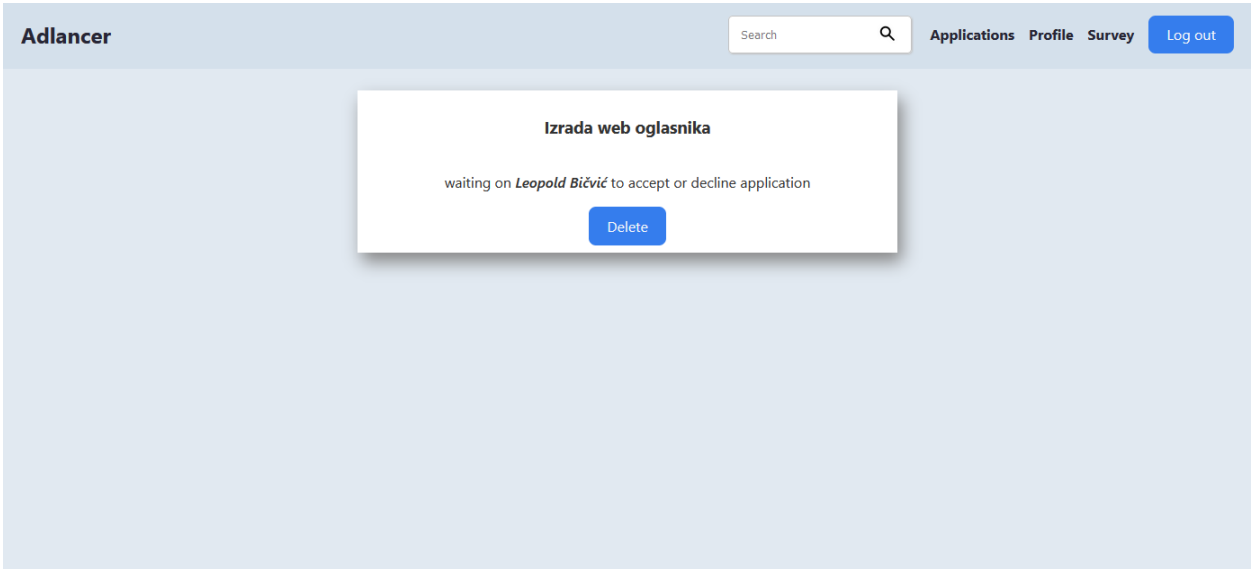


Slika 4.12. *Prikaz detalja korisničke objave*

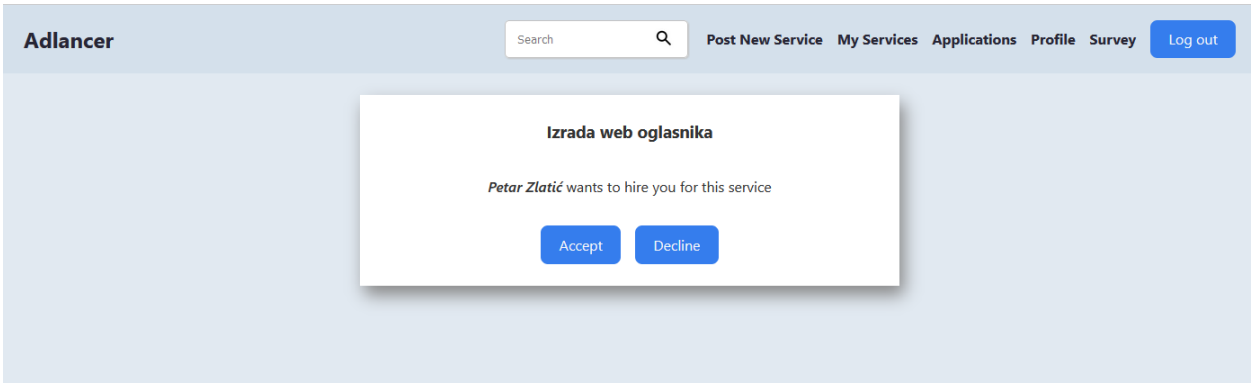
Za mijenjanje sadržaja objave koristi se forma za uređivanje prikazana slikom 4.13. na kojoj se nalazi unos za naslov, opis, cijenu i kategoriju usluge.



Slika 4.13. *Prikaz forme za uređivanje objave*

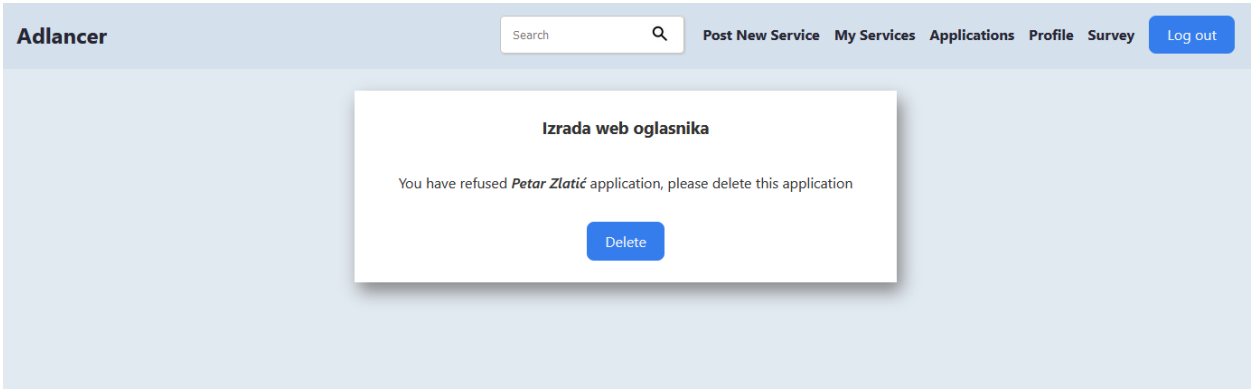


Slika 4.14. *Prikaz zahtjeva za preuzimanje posla na strani korisnika*

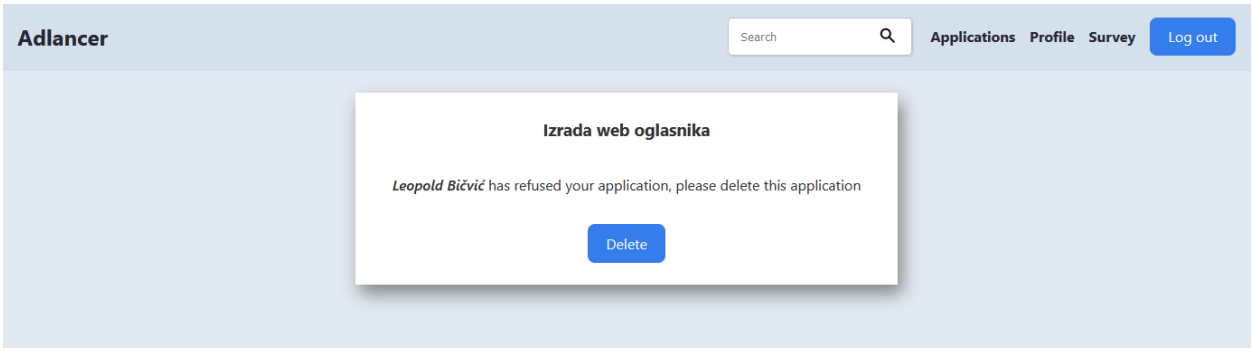


Slika 4.15. *Prikaz zahtjeva za preuzimanje posla na strani freelancera*

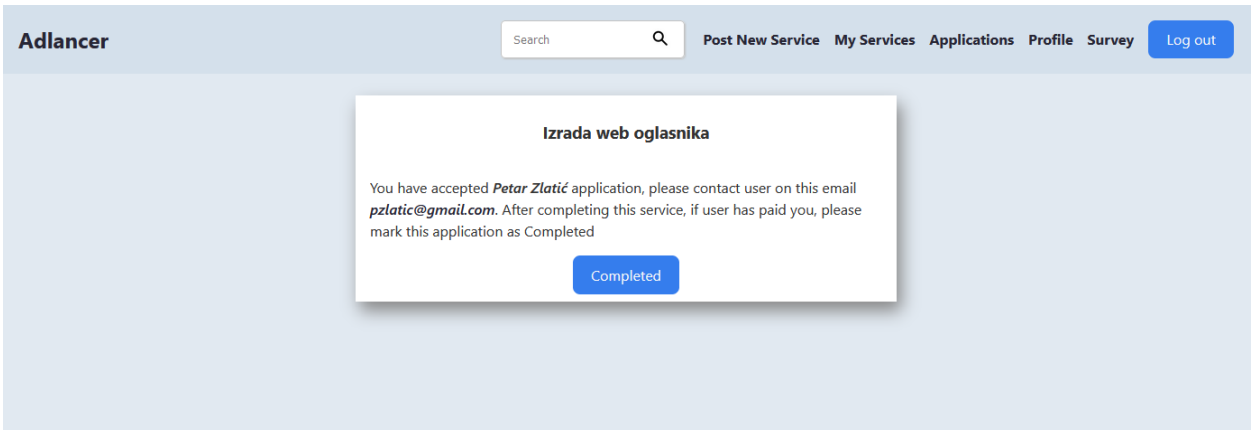
Na slikama 4.14. i 4.15. prikazan je zahtjev za preuzimanje posla iz pogleda korisnika s ulogom *freelancera* i *usera*. Zahtjev sadržava naslov usluge, povratnu informaciju i mogućnosti poput prihvaćanja i odbijanja zahtjeva. Korisniku s ulogom *user* omogućuje se odbijanje zahtjeva, a korisniku s ulogom *freelancer* prihvaćanje ili odbijanje. U slučaju odbijanja zahtjeva prikazan je sadržaj na slikama 4.16. i 4.17. na kojima su vidljive povratne informacije i gumb za brisanje zahtjeva. Ukoliko je zahtjev prihvaćen, korisniku s ulogom *freelancer* prikazan je sadržaj na slici 4.18. koji sadrži povratnu informaciju i gumb za potvrđivanje završenog zahtjeva. Korisniku s ulogom *user* prikazan je sadržaj na slici 4.19. koji sadržava povratnu informaciju s navedenom elektroničkom poštom izvršitelja usluge.



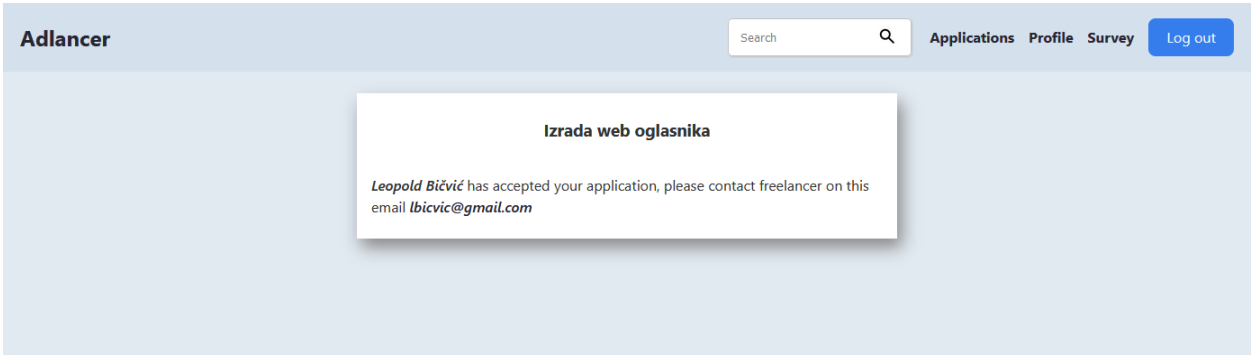
Slika 4.16. *Prikaz odbijenog korisničkog zahtjeva*



Slika 4.17. *Prikaz povratnog sadržaja odbijenog zahtjeva*

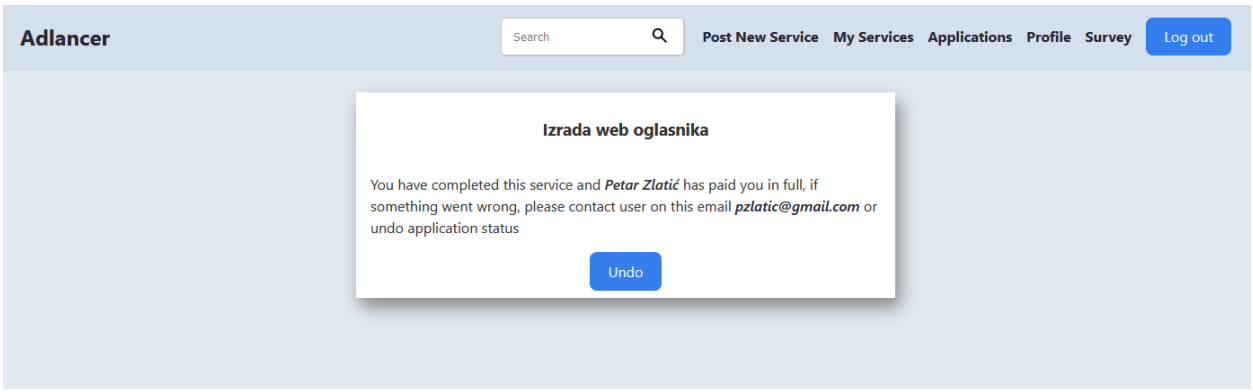


Slika 4.18. *Prikaz prihvaćenog zahtjeva za preuzimanje posla*

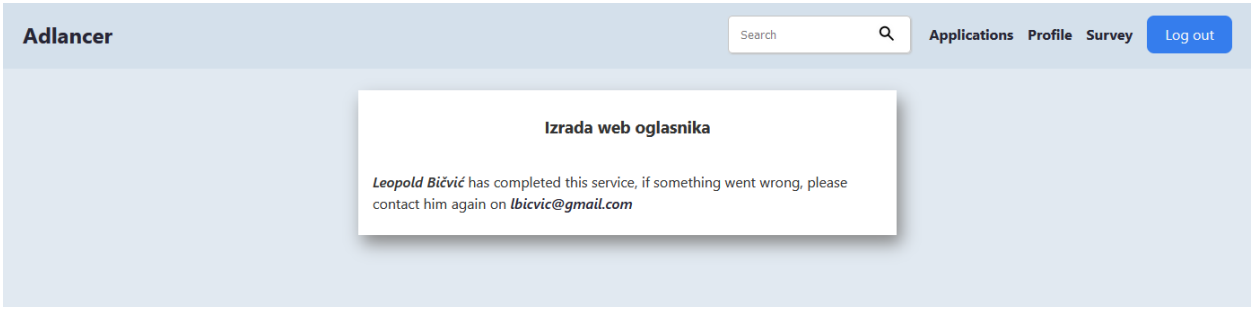


Slika 4.19. *Prikaz povratnog sadržaja prihvaćenog zahtjeva*

Ukoliko je usluga završena, korisniku s ulogom *freelancer* prikazan je sadržaj na slici 4.20. koji sadrži povratnu informaciju i gumb za poništavanje u slučaju kada dogovor nije postignut. Korisniku s ulogom *user* prikazan je sadržaj na slici 4.21. koji sadržava povratnu informaciju s navedenom elektroničkom poštom izvršitelja usluge.



Slika 4.20. *Prikaz sadržaja završenog zahtjeva*

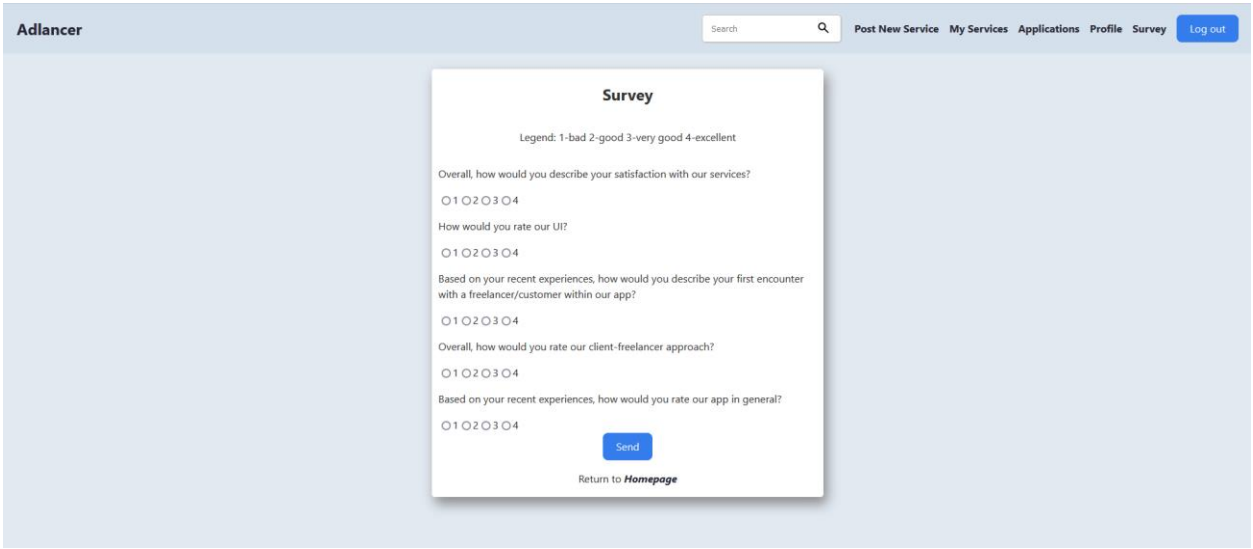


Slika 4.21. *Prikaz povratnog sadržaja završenog zahtjeva*

Pretraživanje postavljenih poslova izvršava se nakon unosa željene riječi. Slikom 4.22. prikazan je primjer korištenja pretrage.

Slika 4.22. *Primjer korištenja pretrage*

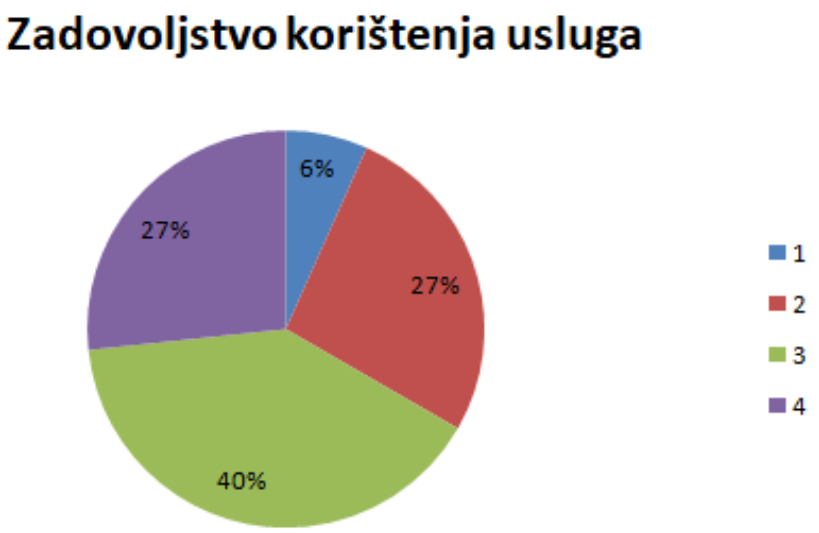
Dodatno, prijavljeni korisnici imaju mogućnost ispunjavanja upitnika kojim mogu ocijeniti iskustvo korištenja aplikacije. Na slici 4.23. prikazan je izrađeni upitnik. Ispod naslova upitnika prikazana je legenda koja opisuje značenje ponuđenih vrijednosti. Upitnik sadržava pitanja, unos za odgovor i gumb za završavanje ankete.



Slika 4.23. *Prikaz upitnika*

5. PRIKAZ REZULTATA ANKETIRANJA

U ovom poglavlju prikazani su rezultati anketiranja grupe korisnika pomoću grafova. Obrada podataka izvršila se unutar programa Microsoft Office Excel 2007. Upitnik sadržava pet kratkih pitanja čiji su odgovori u rangu od 1 do 4 po uzoru na Likertovu skalu. Primjena Likertove skale pruža lakše iskazivanje zadovoljstva korisnika kroz jednostavnost ispunjavanja te jednostavniju obradu analize podataka. Značenje odgovora odnosi se na ocjenu iskustva korištenja od lošeg do odličnog (engl. 1 - *bad*, 2 - *good*, 3 - *very good*, 4 - *excellent*.). Pitanja se zasnivaju na zadovoljstvu korisnika prilikom korištenja usluga te općenitim ocjenjivanjem izgleda i funkcionalnosti unutar web aplikacije. Veličinu testnog uzorka pri anketiranju je činila grupa od 20 korisnika. Na slikama 5.1. i 5.2. izdvojene su zanimljivosti u obrađenim podacima grupe korisnika poput općenitog zadovoljstva korištenja usluga i ocjene web aplikacije.



Slika 5.1. *Prikaz dijagrama zadovoljstva korištenja usluga*

Slika 5.2. *Prikaz dijagrama za ocjenu web aplikacije*

Važno je istaknuti da je najviše korisnika odgovorilo na pitanja brojem 3 (engl. *3 – very good*), vidljivo na prikazanim slikama 5.1. i 5.2, a najmanje brojem 1 (engl. 1 – *bad*). Odgovori na pitanja brojem 2 i 4 (engl. 2 - *good*, 4 - *excellent*.) zastupljeni su u prosjeku podjednako. S obzirom na prethodno navedeno, može se zaključiti kako izrađena web aplikacija teži prema odgovoru broj 3 sa značenjem vrlo dobro te je zadovoljavajuća za korisnike.

6. ZAKLJUČAK

Nagli razvoj tehnologije dovodi do prilagodbe poslodavaca i radnika na digitalni oblik rada. Tom prilagodbom nastaje sve više aplikacija koje pružaju usluge u obliku *freelance* zadataka. Ovim diplomskim radom opisan je postupak izrade web aplikacije za freelancer usluge koja sadržava funkcionalnosti u skladu s očekivanim.

Web aplikacija za freelancer usluge sadržava poslužitelja i klijenta koji su ostvareni pomoću popularnih MERN tehnologija. Unutar web aplikacije implementirane su funkcionalnosti poput stvaranja, uređivanja, pretraživanja i brisanja poslova. Uz navedeno, aplikacija nudi preuzimanje poslova te uvid u korisnički profil i ispunjavanje ankete radi povećanja kvalitete usluga. S obzirom na navedeno, može se zaključiti da aplikacija ispunjava zahtjeve zadatka rada.

Ovom aplikacijom pružen je primjer jedne funkcionalne i zadovoljavajuće web aplikacije za freelancer usluge izrađene pomoću popularnih tehnologija današnjice. Izradom ovakve vrste aplikacije pruža se prilika za učenje trenutno najpopularnijih tehnologija. Također, povećava se kreativnost programera i konkurencija na tržištu rada.

LITERATURA

[1] Fiverr, dostupno na: https://www.fiverr.com/ [29. lipanj 2023.]

[2] Toptal, dostupno na: https://www.toptal.com/ [29. lipanj 2023.]

[3] Upwork, dostupno na: https://www.upwork.com/ [29. lipanj 2023.]

[4] Freelancer, dostupno na: https://www.freelancer.com/ [29. lipanj 2023.]

[5] Peopleperhour, dostupno na: https://www.peopleperhour.com/ [29. lipanj 2023.]

[6] Grupa autora, NodeJS [online], GeeksForGeeks, dostupno na: https://www.geeksforgeeks.org/nodejs/ [29. lipanj 2023.].

[7] Grupa autora, Express.js Introduction [online], MDN,dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction [29. lipanj 2023.]

[8] Grupa autora, MongoDB Introduction [online], GeeksForGeeks, dostupno na: https://www.geeksforgeeks.org/mongodb-an-introduction/ [29. lipanj 2023.]

[9] Grupa autora, Getting started with React [online], MDN, dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started [29. lipanj 2023.]

[10] Grupa autora, JSX In Depth [online], React, dostupno na: https://legacy.reactjs.org/docs/jsx-in-depth.html [29. lipanj 2023.]

SAŽETAK

Osnovni zadatak ovoga diplomskog rada odnosi se na izradu web aplikacije za freelancer usluge koja pruža funkcionalnosti poput stvaranja, uređivanja, brisanja, pretraživanja poslova, zapošljavanja freelancera i dodatnih mogućnosti poput ispunjavanja upitnika. Ovim radom prikazan je postupak izrade web aplikacije za freelancer usluge upotrebom MERN tehnologija. Tehnologije MongoDB, Express.js i Node.js primijenjene su za izradu poslužitelja, a React za izradu klijenta. Uz navedeno, prikazani su rezultati anketiranja grupe registriranih korisnika. Obrada rezultata anketiranja temelji se na Likertovoj skali.

Ključne riječi: Express.js, freelance, Node.js, objava, React

ABSTRACT

Web application for freelancer services

The main task of this paper is developing web application for freelancer services that provides functionalities such as creating, updating, deleting, searching, hiring freelancers and additional options like filling out questionnaire. This paper presents the process of creating a web application for freelancer services using MERN technologies. MongoDB, Express.js and Node.js technologies were used to create the server and React was used to create the client. In addition to the above, the survey results of a group of registered users are presented. The processing of survey results is based on a Likert scale.

Keywords: Express.js, freelance, Node.js, post, React

ŽIVOTOPIS

Leopold Bičvić rođen je datuma 18.5.1998. u Osijeku, Republika Hrvatska. Upisuje se na preddiplomski sveučilišni studij Računarstvo 2017. godine na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku te 2021. godine stječe status prvostupnika. Nastavlja svoje obrazovanje upisujući diplomski studij Računarstvo, izborni blok Programsko inženjerstvo na istom fakultetu.

_____

Potpis autora

PRILOZI

Dokument ovoga diplomskog rada u formatima .docx i .pdf. zajedno s izvornim kodom izrađene web aplikacije nalazi se na CD/DVD-u.