Web aplikacija za kućne popravke

Damjanović-Maglica, Juraj

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

Sveučilišni studij računarstva

Web aplikacija za kućne popravke

Završni rad

Juraj Damjanović-Maglica

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Juraj Damjanović-Maglica
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR4657, 25.07.2018.
JMBAG:	0165078563
Mentor:	Marina Peko, dipl. ing. el.
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	Marina Peko, dipl. ing. el.
Član Povjerenstva 2:	doc. dr. sc. Ivana Hartmann Tolić
Naslov završnog rada:	Web aplikacija za kućne popravke
Znanstvena grana završnog rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Web aplikacija na kojoj se mogu prijaviti korisnici koji znaju obaviti neke popravke u domaćinstvu, te korisnici koji traže majstore za popravke - 2 role korisnika te administratorska rola. Detalji o aplikaciji će biti dani studentu na prvim konzultacijama.

Datum ocjene pismenog dijela završnog rada od strane mentora:	23.09.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Dobar (3)
Datum obrane završnog rada:	27.09.2024.
Ocjena usmenog dijela završnog rada (obrane):	Vrlo dobar (4)
Ukupna ocjena završnog rada:	Vrlo dobar (4)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	03.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 03.10.2024.

Ime i prezime Pristupnika:	Juraj Damjanović-Maglica
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4657, 25.07.2018.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za kućne popravke**

izrađen pod vodstvom mentora Marina Peko, dipl. ing. el.

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 pristupnika:

SADRŽAJ

1. UVOD

Web aplikacija je primjenjivi program koji se pohranjuje na serveru i isporučuje preko interneta putem sučelja preglednika. Svaka komponenta web stranice koja obavlja neku funkciju za korisnika kvalificira se kao web aplikacija. Web aplikacije mogu biti dizajnirane za širok raspon namjena i mogu ih koristiti svi; od organizacije do pojedinca iz brojnih razloga. Često korištene web aplikacije mogu uključivati web-poštu, online kalkulatore ili e-trgovine. Nekim web aplikacijama može pristupiti samo određeni preglednik; međutim, većina je dostupna bez obzira na preglednik.

Platforma za traženje usluga kućnih servisa je cilj ovog završnog rada. Za korištenje web aplikacije potrebno je kreirati račun koji odgovara željama korisnika ovisno što želi od web aplikacije, bilo to traženje usluga ili davanje usluga. Sa korisničke strane moguće je kreirati karte s kojima korisnici traže pomoć za neki njihov kućni problem, prijaviti se na servise radnika i napraviti zahtjev za kućnu uslugu, te onda ocjeniti tu uslugu. Sa radničke strane moguće je prijaviti se na karte od korisnika i napraviti ponudu za rad, kreirati servis koje radnik nudi kao vještine. Cilj ove web aplikacije je olakšati stvaranje zahtjeva i potražnju usluga za kućne servise.

Završni rad je opisan u pet poglavlja, prvo poglavlje je uvod u web aplikaciju, šta predstavlja i koji je cilj web aplikacije, drugo poglavlje je pregled sličnih aplikativnih riješenja koja postoje danas i usporedba tih riješenja koje su prednosti i mane naspram ove web aplikacije. Treće poglavlje opisuje tehnologije koje su se koristile u izradi aplikacije i s tim objašnjava zašto su se koristile. Četvrto poglavlje sadrži srž web aplikacije, to jest komponente s kojima web aplikacija nudi svoje značajke, unutar poglavlja se objašnjavaju procesi te komunikacija unutar aplikacije. Peto poglavlje predstavlja sami izgled web aplikacije i njezine djelove i funkcionalnosti, poglavlje objašnjava čemu služi pojedini dio web aplikacije.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je napraviti web aplikaciju za kućne servise preko koje će korisnici moći izabrati i naručiti određene usluge za njihov dom. Potrebno je implementirati korisnični račun te funkcionalnost nuđenja usluga za čišćenje ili održavanje doma. Uz navedeno, potrebno je održati jednostavno korisničko sučelje kako bi korisničko iskustvo bilo poboljšano i olakšano svim dobnim skupinama. Također, potrebno je prikazati koje su mogućnosti programskih jezika koji se koriste za izradu web aplikacije.

2. PREGLED PODRUČJA TEME

Danas na internetu postoji velika količina web aplikacija i većina dijeli slične koncepte kod izgradnje aplikacije. Kod web aplikacija za kućni servis 2 ključne stvari su: pristupačnost svim korisnicima, odnosno preglednost kod izabiranja i potražnje za servisom, te prosječni izračun cijene izabranih servisa. Pristupačnost je bitna stavka jer je prvi aspekt s kojim se korisnik susreće prilikom posjete web aplikacije te u pravilu uvelike pridonosi tome hoće li korisnik nastaviti koristiti web aplikaciju. Pomoću prosječnog izračuna korisniku pomažemo stvoriti sliku kako, što i koliko je potrebno za njegovo rješavanje određenih problema kod kuće.

2.1. Pregled sličnih rješenja

Platforma na kojoj se mogu nuditi usluge nije novitet u današnjem svijetu, web aplikacije poput Fiverr, Upwork ili Toptal su sve platforme pomoću kojih pojedinici sa određenim vještinama mogu te iste vještine ponuditi kao uslugu korisniku koji ih treba za riješavanje svog problema [1] [2] [3]. Fiverr je online platforma u obliku tržišta gdje korisnici mogu prodavati svoje vještine ili zatražiti tuđe vještine za svoje probleme. Problem Fiverr-a je u tome što vještine koje možeš ponuditi su ograničene po potrebama tržišta samog Fiverr-a. Upwork je platforma za potražnju posla na projektima, korisnici se natječu za mjestu na projektu u načinu nadmetanja, korisnik isto može sam postaviti neku ideju projekta na koju se drugi mogu prijaviti. Mana Upwork-a je u tom što uzimau proviziju na zaradu u zamjenu korištenja njihove platforme. Toptal je web stranica koja služi kao mreža slobodnih agenata gdje korisnik može predstaviti svoje vještine i onda razne firme korisnika mogu pregledati njegov profil, te poslati mu zahtjev za rad.

3. TEHNOLOGIJE WEB APLIKACIJE

3.1. HTML

HTML (engl. HyperText Markup Language) je jedan od najosnovnijih gradbenih elemenata interneta. Definira značenje i strukturu te poboljšava, olakšava organizaciju i prikazivanje web sadržaja na WWW (engl. World Wide Web). Koristi takozvano označavanje (engl. markup) teksta, slika i drugog sadržaja za prikaz na web pregledniku koristeći oznake (engl. tag). Tim načinom je omogućen prikaz velikog broja elemenata, poput odlomaka, slika, poveznica i drugog multimedijskog sadržaja.

Elementi su odvojeni od drugog teksta u dokumetu pomoću „<“ i „>“. Ime elemenata unutar oznaka ne razlikuje velika i mala slova međutim, konvencija i preporučena praksa je pisanja oznaka malim slovima.

Jedna od najvažnijih značajki HTML-a je njegova semantička priroda, što znači da oznake ne služe samo za prezentaciju sadržaja, već također daju značenje različitim dijelovima dokumenta. Oznake ukazuju pregledniku kako da protumači svaku oznaku, primjerice oznaka za naslov ukazuje da je određeni tekst ključni naslov stranice. Ova semantička struktura omogućuje pretraživačima i alatima za pristupačnost da bolje razumiju sadržaj stranice, što poboljšava korisničko iskustvo i optimizira rezultate pretraživanja. [4]

Također, HTML je platformski neovisan, što znači da se može koristiti na bilo kojem operativnom sustavu i web pregledniku, zbog čega je ključni alat za stvaranje univerzalno dostupnog web sadržaja. Web preglednici interpretiraju HTML kod i prikazuju sadržaj korisnicima u skladu s pravilima koja su definirana standardima.

3.2. Javascript

JavaScript je interpretirani programski jezik s prvoklasnim funkcijama. Najpoznatiji je kao skriptni jezik za web stranice, no koriste ga i mnoga okruženja koja nisu preglednici, kao što su Node.js i Adobe Acrobat. To je prototipski dinamički jezik koji podržava objektno-orijetirane, imperativne i deklarativne stilove.

Jezik Javascript, koji radi u paru s povezanim značajkama preglednika, tehnologija je za poboljšanje web sadržaja. Kada se koristi na klijentskom računalu, jezik pomaže pretvoriti statičnu stranicu sadržaja u zanimljivo, interaktivno i inteligentno iskustvo. [5] Omogućuje programerima stvaranje bogatih korisničkih sučelja, obrade podataka u stvarnom vremenu i interakcije s korisnicima na način koji nije moguć samo uz statične jezike poput HTML-a i CSS-a.

Jedna od ključnih karakteristika JavaScripta je direktno izvršavanje koda u pregledniku, bez potrebe za prethodnom interpretacijom. To omogućava brže izvršavanje skripti i stvaranje interaktivnog sadržaja u stvarnom vremenu. Također, JavaScript je dinamički tipiziran jezik, što znači da se vrste podataka mogu mijenjati tijekom izvođenja programa, što olakšava rad s različitim vrstama podataka.

U kombinaciji s tehnologijama kao što su AJAX (engl. Asynchronous JavaScript and XML), JavaScript omogućava slanje i primanje podataka s poslužitelja u stvarnom vremenu bez potrebe za ponovnim učitavanjem stranice, što značajno poboljšava korisničko iskustvo.

Također, JavaScript omogućava rad s DOM-om (engl. Document Object Model), što omogućava programerima manipulaciju HTML i CSS elementima na stranici, promjenu stilova, dodavanje ili uklanjanje elemenata te stvaranje potpuno dinamičkih korisničkih sučelja.

3.3. Angular

Angular je *framework*[1] za dizajn aplikacija i razvojna platforma za stvaranje učinkovitih i sofisticiranih aplikacija na jednoj stranici. Izgrađen je na TypeScript-u te kao platforma uključuje komponentni *framework* za izgradnju skalabilnih web aplikacija, zbirku dobro integriranih knjižnica koje pokrivaju širok raspon značajki, uključujući usmjeravanje, upravljanje obrascima, komunikaciju klijent-poslužitelj i još mnogo toga. Angular je skup alata za koji pomažu pri razvijanju, izgradnji i ažuriranju koda web aplikacije. [6]

Glavna značajka ovog *frameworka* je komponentalni pristup organizaciji strukture web aplikacije. To su neovisni dijelovi web aplikacije koji mogu biti ponovno upotrebljeni, što je velika prednost prilikom razvijanja kompleksne web aplikacije s puno različitih funkcionalnosti i stranica

[1] stvarna ili idejna konstrukcija koja služi kao potpora ili vodič za razvijanje.

Komponente su građevni blokovi koji uključuje klasu TypeScript, HTML predložak i CSS stilove. HTML predožak opisuje samu strukturu komponente, dok TypeScript klasa opisuje logiku i ponašanje komponente. CSS stilovi dodaju izgled samoj stranici, poput boje fontova ili udaljenosti između HTML elemenata.

Jedna od najčešćih primjena Angulara je razvoj jednostranih aplikacije (engl. Single Page Applications - SPA), gdje cijela aplikacija radi unutar jedne stranice, a promjene u sadržaju vrše se dinamički bez ponovnog učitavanja stranice. Ovaj pristup omogućava brže i tečnije korisničko iskustvo. Angular je također izuzetno popularan u okruženjima poduzećima zbog svoje sposobnosti skalabilnosti velikih projekata te pruža robusne alate za testiranje, upravljanje stanjem i modularizaciju.

3.4. Bootstrap

Bootstrap je besplatni open source[2] *framework* za razvoj front-end[3] web stranica i web aplikacija. *Framework* Bootstrap izgrađen je na HTML-u, CSS-u i JavaScriptu kako bi se olakšao razvoj responzivnih web-lokacija i aplikacija namijenjenih mobilnim uređajima. Njegova glavna svrha je olakšati i ubrzati proces razvoja web stranica te osigurati dosljedan dizajn kroz različite platforme i uređaje. [7]

Glavni cilj Bootstrapa je ubrzati razvoj web stranica, omogućujući programerima da se fokusiraju na funkcionalnost i sadržaj, a ne na detalje dizajna i stilizacije. Zahvaljujući unaprijed definiranim komponentama, Bootstrap omogućuje brzo postavljanje profesionalno dizajniranih stranica i aplikacija bez potrebe za opsežnim CSS znanjem.

3.5. Firebase

Firebase je skup alata za izgradnju, poboljšanje i razvoj aplikacije. Firebase se definira kao platforma kao usluga (engl. Platform as a Service – PaaS) čiji alati pokrivaju velik dio usluga kao

[2] odnosi se na softver čiji je izvorni kod dostupan unutar "open source" licence svim korisnicima koji ga mogu mijenjati, prepravljati i poboljašvati njegov sadržaj.
[3] Grafički dio korisničkog sučelja web stranice

što su analitika, autentifikacija, baze podataka, konfiguracija, pohrana datoteka, push poruka i mnoge ostale. Usluge su smještene na oblaku i skaliraju se uz malo truda od strane developera. [8]

Neke od najvažnijih značajki uključuju:

- Firebase Realtime Database je baza podataka koja pohranjuje podatke u JSON formatu i omogućava sinkronizaciju podataka u stvarnom vremenu. To znači da se svi podaci automatski ažuriraju na svim povezanim klijentima, bez potrebe za ručnim osvježavanjem.
- Cloud Firestore je naprednija baza podataka koja podržava strukturirane upite, transakcije i bolje upravljanje skaliranjem. Firestore omogućava rad s većim i složenijim setovima podataka, podržava *offline* mod i fleksibilno indeksiranje.
- Autentikacija: Firebase pruža jednostavne alate za autentifikaciju korisnika putem različitih metoda, uključujući email i lozinku, telefonski broj, te prijave putem društvenih mreža. Firebase Authentication omogućava brzu integraciju ovih metoda bez potrebe za vlastitom implementacijom sigurnosnih protokola.
- Firebase Hosting: Ova usluga omogućava brzo i sigurno posluživanje za web aplikacije. Firebase Hosting je optimiziran za jednostavnu implementaciju SPA aplikacija i pruža mogućnost postavljanja dinamičkih sadržaja putem Cloud Functions. [9]

4. POSTAVLJANJE RAZVOJNOG OKRUŽENJA I KOMPONENTE WEB APLIKACIJE

4.1. Postavljanje razvojnog okruženja web aplikacije

Razvojna okolina za Angular uključuje instalaciju potrebnih alata i konfiguraciju projekta. Koraci potrebni za postavljanje Angular razvojne okoline su idući:

Korak 1: Instalacija Node.js

Angular zahtijeva Node.js i njegov paketni upravitelj *npm*. Node.js se preuzima preko interneta, te nakon instalacije potrebno je provjeriti je li instalacija uspješno završila pokretanjem sljedećih naredbi u naredbenom retku (engl. command prompt) prikazanim u programskom kodu 4.1:

```
Node –v
Npm –v
```

Programski kod 4.1. *Provjera verzije Node.js*

Korak 2: Instalacija Angular CLI

Angular CLI (engl. Command Line Interface) koristi se za generiranje, izgradnju i upravljanje Angular projektima. Instalira se koristeći *npm* paket putem naredbe prikazane u programskom kodu 4.2:

```
npm install -g @angular/cli
```

Programski kod 4.2*. Instalacija Angular CLI*

Korak 3: Kreiranje novog Angular projekta

Nakon instalacije potrebnih alata, kreira se novi Angular projekt naredbom prikazanom u programskom kodu 4.3:

```
ng new naziv-projekta
```

Programski kod 4.3*. Kreiranje novog Angular projekta*

Korak 4: Pokretanje razvojnog poslužitelja

Kako bi se pokrenio Angular projekt lokalno, potrebno je koristiti naredbu prikazanu u programskom kodu 4.4:

```
ng serve
```

Programski kod 4.4. *Pokretanje Angular projekta*

Aplikacija se pokreće na lokalnom poslužitelju na adresi *http://localhost:4200/* te je Angular aplikacije spremna za daljnje razvijanje i izgradnju.

Iduća poglavlja opisuju sve korištene komponente koje su gradivni dio Angular aplikacije razvijene u završnom radu.

4.2. Login komponenta

Programski kod 4.5. implementira komponentu za prijavu korisnika u Angular aplikaciji koristeći Firebase autentifikaciju. Komponenta sadrži formu s poljima za email i lozinku. Nakon unosa, korisnik se može prijaviti pomoću Firebase funkcije *signInWithEmailAndPassword()* koja se pokreće u metodi *onSubmit()*.

Ako je prijava uspješna, korisnik se preusmjerava na stranicu */dashboard*, a prikazuje se poruka putem *MatSnackBar* komponente. U slučaju greške, koristi se *FirebaseError* za prikaz relevantnih poruka, npr. "Korisnik nije pronađen" ili "Neispravna lozinka".

Također, koristi se *Router* za navigaciju i *FormBuilder* za izradu reaktivne forme.

```
export class LoginComponent implements OnInit {
  (...)

  async onSubmit(): Promise<void> {
    if (this.loginForm.valid) {
      const { email, password } = this.loginForm.value;

      try {
        const userCredential = await signInWithEmailAndPassword(this.auth,
email, password);
        console.log('User signed in:', userCredential);

        this.snackBar.open('Logging in!', 'Close', {
          duration: 5000,
          panelClass: ['success-snackbar']
        });
        this.router.navigate(['/dashboard']);
      } catch (error: unknown) {
        console.error('Login error:', error);

        if (error instanceof FirebaseError) {
          switch (error.code) {
            case 'auth/user-not-found':
              this.errorMessage = 'No user found with this email address.';
              break;
            case 'auth/wrong-password':
              this.errorMessage = 'Incorrect password. Please try again.';
              break;
            default:
              this.errorMessage = 'Login failed. Please check your
credentials and try again.';
```

```
              break;
            }
          } else {
            this.errorMessage = 'An unexpected error occurred. Please try
again later.';
          }
        }
      }
    }
}
```

Programski kod 4.5. *Login komponenta*

4.3. Register komponenta

Programski kod 4.6. implementira komponentu za registraciju korisnika u Angular aplikaciji koristeći Firebase Authentication i Firestore. Komponenta omogućava korisnicima da se registriraju kao radnici ili obični korisnici, ovisno o tipu korisnika koji odaberu.

Forma sadrži polja za korisničko ime, ime, prezime, email, lozinku i dodatne informacije kao što su usluga, ocjena i biografija za radnike. Validacija unutar forme osigurava da su lozinka i potvrđena lozinka iste.

Prije kreiranja novog korisnika, provjerava se postoji li već korisničko ime u odgovarajućoj kolekciji Firestorea. Ako je korisničko ime zauzeto, korisnik dobiva poruku o grešci. Ako je registracija uspješna, novi korisnik se kreira pomoću funkcije *createUserWithEmailAndPassword()*, a podaci korisnika, uključujući korisničko ime i dodatne informacije, spremaju se u Firestore.

Ovisno o tipu korisnika, podaci se spremaju u kolekciju *workers* za radnike ili *users* za obične korisnike. Nakon uspješne registracije, korisnik se preusmjerava na stranicu za prijavu, uz prikaz poruke o uspjehu.

```
export class RegisterComponent implements OnInit {
  registerForm: FormGroup;

  constructor(
    private fb: FormBuilder,
    private auth: Auth,
    private firestore: Firestore,
    private router: Router,
    private snackBar: MatSnackBar
  ) {
    this.registerForm = this.fb.group({
      userType: ['user', Validators.required],
      username: ['', [Validators.required]],
      name: ['', [Validators.required]],
      surname: ['', [Validators.required]],
      email: ['', [Validators.required, Validators.email]],
```

```
      password: ['', [Validators.required]],
      confirmPassword: ['', [Validators.required]],
      service: [''],
      score: [''],
      requests: [''],
      bio: ['']
    }, { validators: this.passwordMatchValidator });
  }

  (...)

  passwordMatchValidator(g: FormGroup) {
    return g.get('password')?.value === g.get('confirmPassword')?.value
      ? null : { notSame: true };
  }

  isFieldInvalid(fieldName: string): boolean {
    const control = this.registerForm.get(fieldName);
    return control?.invalid && (control?.touched || control?.dirty) ? true :
false;
  }


  async onSubmit() {
    if (this.registerForm.invalid) {
      return;
    }

    const { userType, username, name, surname, email, password,
confirmPassword, service, score, requests, bio } = this.registerForm.value;

    try {
      const userCollection = userType === 'worker' ? 'workers' : 'users';
      const userQuery = query(collection(this.firestore, userCollection),
where('username', '==', username));
      const querySnapshot = await getDocs(userQuery);

      if (!querySnapshot.empty) {
        this.registerForm.get('username')?.setErrors({ usernameTaken: true
});
        this.snackBar.open('Username already taken. Please choose another
one.', 'Close', {
          duration: 5000,
          panelClass: ['error-snackbar']
        });
        return;
      }

      const userCredential = await createUserWithEmailAndPassword(this.auth,
email, password);
      const user = userCredential.user;

      await updateProfile(user, { displayName: username });

      const collectionName = userType === 'worker' ? 'workers' : 'users';

      const userRef = doc(this.firestore, `${collectionName}/${user.uid}`);
```

```
  (...)

      if (userType === 'worker') {
        userData.service = service ?? null;
        userData.score = score ?? null;
        userData.requests = requests ?? [];
        userData.bio = bio ?? '';
      }

      if (userType === 'user') {
        userData.bio = bio ?? '';
      }

      await setDoc(userRef, userData);

      this.snackBar.open('Register complete!', 'Close', {
        duration: 5000,
        panelClass: ['success-snackbar']
      });
      this.router.navigate(['/login']);
    } catch (error) {
      console.error('Error during registration:', error);
      this.snackBar.open('Error during registration. Please try again.',
'Close', {
        duration: 5000,
        panelClass: ['error-snackbar']
      });
    }
  }
}
```

Programski kod 4.6. *Register komponenta*

4.4. Password reset komponenta

Programski kod 4.7. implementira komponentu za ponovno postavljanje lozinke u Angular aplikaciji. Komponenta omogućava korisnicima da zatraže e-mail za ponovno postavljanje lozinke.

Forma se sastoji od jednog polja za unos email adrese koja mora biti ispravno unesena. Kada se forma pošalje, koristi se *AuthService* za slanje e-maila za ponovno postavljanje lozinke. Nakon uspješnog slanja e-maila, korisnik dobiva obavijest i preusmjerava se na stranicu za prijavu. Ako dođe do greške, prikazuje se odgovarajuća poruka.

```
export class PasswordResetComponent {
  resetForm: FormGroup;

  (...)

  onSubmit(): void {
```

```
    if (this.resetForm.valid) {
      const email = this.resetForm.get('email')?.value;
      this.authService.sendPasswordResetEmail(email).subscribe({
        next: () => {
          alert('Password reset email sent!');
          this.router.navigate(['/login']);
        },
        error: (error) => {
          alert('Error sending password reset email: ' + error.message);
        }
      });
    }
  }
}
```

Programski kod 4.7. *PasswordReset komponenta*

4.5. Main komponenta

Programski kod 4.8. implementira glavnu komponentu u Angular aplikaciji koja omogućava upravljanje kartama i uslugama, te prijavu za iste. Komponenta učitava karte i usluge iz Firestore baze podataka. Karte i usluge mogu se filtrirati prema kategorijama, rasponu datuma i cijeni. Postoji mogućnost filtriranja karata prema odabranim kategorijama i datumima, te usluga prema kategoriji i cijeni. Korisnici mogu poslati svoje prijave za karte i usluge, a obrazac za prijavu uključuje unos teksta zahtjeva, datuma početka i završetka te ostalih relevantnih informacija.

Kada se komponenta inicijalizira, provjerava je li korisnik prijavljen. Na temelju korisničkog ID-a, komponenta određuje vrstu korisnika, kao što su radnik ili korisnik, i prema tome učitava odgovarajuće karte ili usluge. Detalji o korisnicima i radnicima povezani s kartama i uslugama se učitavaju i pohranjuju u lokalne varijable kako bi se mogli prikazati u korisničkom sučelju.

Komponenta omogućava unos i provjeru datuma prijave, osiguravajući da su uneseni datumi ispravni. Provjerava se da li je datum završetka kasniji od datuma početka i da li datum početka nije prošli datum. Koristi modalne prozore za prijavu za karte i usluge, omogućavajući korisnicima da unesu svoje podatke i podnesu zahtjeve. Obavijesti o uspješnim ili neuspješnim prijavama prikazuju se putem snackbar poruka. Ovaj kod pruža složenu funkcionalnost za rad s kartama i uslugama u aplikaciji, uključujući filtriranje, prijavu i upravljanje korisničkim informacijama.

```
export class MainComponent implements OnInit {

 (...)

  async loadTickets() {
    try {
      const ticketsRef = collection(this.firestore, 'tickets');
      const querySnapshot = await getDocs(ticketsRef);
```

```
      const tickets: Ticket[] = [];
      for (const docSnap of querySnapshot.docs) {
        const data = docSnap.data();
        const ticket: Ticket = {
          id: docSnap.id,
          name: data['name'],
          userName: data['userName'],
          description: data['description'],
          category: data['category'],
          startDate: data['startDate'],
          endDate: data['endDate'],
          userId: data['userId'],
          status: data['status'],
          applications: data['applications'] || [],
          selectedWorker: data['selectedWorker'] || []
        };

        const userDocRef = doc(this.firestore, 'users', ticket.userId);
        const userDocSnap = await getDoc(userDocRef);
        const userData = userDocSnap.data();
        if (userData) {
          this.userProfiles[ticket.userId] = {
            profilePicture: userData['profilePicture'],
            username: userData['username']
          } as UserProfile;
        }

        tickets.push(ticket);
      }

      this.tickets = tickets;
      this.filterTickets();
    } catch (error) {
      console.error('Error loading tickets or user profiles: ', error);
    }
  }

  async loadServices() {
    try {
      const servicesRef = collection(this.firestore, 'services');
      const querySnapshot = await getDocs(servicesRef);

      const services: Service[] = [];
      for (const docSnap of querySnapshot.docs) {
        const data = docSnap.data();
        const service: Service = {
          id: docSnap.id,
          name: data['name'],
          description: data['description'],
          category: data['category'],
          startTime: data['startTime'],
          endTime: data['endTime'],
          userId: data['userId'],
          cost: data['cost']
        };

        const workerDocRef = doc(this.firestore, 'workers', service.userId);
```

```
        const workerDocSnap = await getDoc(workerDocRef);
        const workerData = workerDocSnap.data();
        if (workerData) {
          this.userProfiles[service.userId] = {
            profilePicture: workerData['profilePicture'],
            username: workerData['username'],
            score: workerData['score'] || 0,
            requestsDone: workerData['requestsDone'] || 0
          } as UserProfile;
        }

        services.push(service);
      }

      this.services = services;
      this.filterServices();
    } catch (error) {
      console.error('Error loading services or worker profiles: ', error);
    }
  }

  filterTickets() {
    this.filteredTickets = this.tickets.filter(ticket => {
      const ticketDate = new Date(ticket.startDate);
      const fromDate = new Date(this.dateRange.from);
      const toDate = new Date(this.dateRange.to);

      return (!this.selectedCategory || ticket.category ===
this.selectedCategory) &&
             (!this.dateRange.from || ticketDate >= fromDate) &&
             (!this.dateRange.to || ticketDate <= toDate);
    });
  }

(...)

  applyForTicket(ticket: Ticket) {
    this.selectedTicket = ticket;
    const userId = ticket.userId;
    const userRef = doc(this.firestore, `users/${userId}`);
    getDoc(userRef).then((docSnap) => {
      if (docSnap.exists()) {
        this.selectedTicketUser = docSnap.data() as UserProfile;
      }
    });
  }
  closeTicketApplyForm() {
    this.selectedTicket = null;
    this.ticketRequestText = '';
  }

  async applyForTicketRequest() {
    if (!this.selectedTicket) return;

    try {
      const application = {
        ticketId: this.selectedTicket.id,
```

```
        userId: this.selectedTicket.userId,
        userName: this.selectedTicket.userName,
        ticketName: this.selectedTicket.name,
        ticketDesc: this.selectedTicket.description,
        startDate: this.selectedTicket.startDate,
        endDate: this.selectedTicket.endDate,
        workerId: this.auth.currentUser?.uid,
        requestText: this.ticketRequestText,
        cost: this.ticketRequestCost,
        category: this.selectedTicket.category,
        startTime: this.ticketRequestStartTime,
        endTime: this.ticketRequestEndTime,
        status: 'Pending'
      };

      await addDoc(collection(this.firestore, 'ticketApplications'),
application);
      this.snackBar.open('Application submitted successfully!', 'Close', {
        duration: 5000,
        panelClass: ['success-snackbar']
      });
      this.closeTicketApplyForm();
    } catch (error) {
      console.error('Error applying for ticket: ', error);
    }
  }

(...)
```

Programski kod 4.8. *Main komponenta*

4.6. Profile komponenta

Programski kod 4.9. definira profil komponentu u Angular aplikaciji koja omogućava korisnicima da upravljaju svojim profilom, uključujući ažuriranje biografije, upravljanje kartama i zahtjevima, te ocjenjivanje usluga.

Kada se komponenta inicijalizira, provjerava je li korisnik prijavljen i učitava podatke o korisniku, kartama, potvrđenim i odbijenim zahtjevima. Omogućuje korisnicima da prikazuju i skrivaju različite sekcije kao što su karte, potvrđeni zahtjevi, odbijeni zahtjevi i povijest zahtjeva.

Komponenta omogućava učitavanje i ažuriranje podataka korisnika iz Firestore-a, uključujući biografiju i profilnu sliku. Ako korisnik odabere datoteku za profilnu sliku, ta slika se učitava na Firebase Storage i ažurira se URL slike u Firestore-u.

Koristi modalne prozore za prikaz detalja karte i zahtjeva, te omogućava korisnicima da ažuriraju ili izbrišu karte. Također pruža mogućnost potvrde brisanja računa, što uključuje brisanje korisničkih podataka iz Firestore-a i Firebase Authentication-a, kao i svih povezanih karata i zahtjeva.

Kada se korisnik prijavi za uslugu, može ostaviti ocjenu koju se pohranjuje u Firestore, a status zahtjeva se ažurira na "ocijenjeno". Odbijeni zahtjevi se mogu obrisati, a podaci se učitavaju i ažuriraju kako bi korisničko sučelje odražavalo najnovije promjene.

Kod također uključuje provjere valjanosti za unos ocjena i drugih podataka, kao i prikaz obavijesti putem *snackbar* poruka za uspješne ili neuspješne akcije.

```
export class ProfileUserComponent implements OnInit {
(...)

  onFileSelected(event: Event) {
    const input = event.target as HTMLInputElement;
    if (input.files && input.files.length > 0) {
      const file = input.files[0];
      this.uploadProfilePicture(file);
    }
  }

  async uploadProfilePicture(file: File) {
    if (!this.userId) return;

    const fileRef = ref(this.storage,
`profilePictures/${this.userId}/${file.name}`);
    await uploadBytes(fileRef, file);
    const photoURL = await getDownloadURL(fileRef);

    const userRef = doc(this.firestore, `users/${this.userId}`);
    await updateDoc(userRef, { profilePicture: photoURL });
    this.loadUserData();
  }

  async updateTicket() {
    if (this.selectedTicket) {
      const ticketRef = doc(this.firestore,
`tickets/${this.selectedTicket.id}`);
      await updateDoc(ticketRef, {
        name: this.selectedTicket.name,
        description: this.selectedTicket.description,
        startDate: this.selectedTicket.startDate,
        endDate: this.selectedTicket.endDate
      });
      this.snackBar.open('Ticket updated!', 'Close', {
        duration: 5000,
        panelClass: ['success-snackbar']
      });
      this.closeTicketDetail();
      this.loadUserTickets();
    }
  }

  async deleteTicket(ticketId: string) {
    const ticketRef = doc(this.firestore, `tickets/${ticketId}`);
    this.snackBar.open('Ticket deleted!', 'Close', {
      duration: 5000,
```

```
      panelClass: ['success-snackbar']
    });
    await deleteDoc(ticketRef);
    this.closeTicketDetail();
    this.loadUserTickets();
  }

(...)

  async submitRating(): Promise<void> {
    if (this.ratingForm.invalid || !this.selectedRequest) {
      return;
    }
    console.log(this.selectedRequest);
    const { description, rating } = this.ratingForm.value;
    const newRating = {
      userId : this.selectedRequest.userId,
      userName: this.selectedRequest.userName,
      workerId : this.selectedRequest.workerId,
      description,
      requestStatus:'done',
      rating: Number(rating),
      requestId: this.selectedRequest.id,
      timestamp: new Date()
    };
    try {
      const ratingsCollection = collection(this.firestore,
'workersRatedRequests');
      await addDoc(ratingsCollection, newRating);
      console.log('Rating submitted successfully');

      this.snackBar.open('Request rated!', 'Close', {
        duration: 5000,
        panelClass: ['success-snackbar']
      });

      const requestDocsRef = doc(this.firestore,
`requests/${this.selectedRequest.requestId}`);
      await updateDoc(requestDocsRef, { status: 'rated' });
      console.log('Request status updated to "rated" successfully');

      const requestDocRef = doc(this.firestore,
`confirmedRequests/${this.selectedRequest.id}`);
      await deleteDoc(requestDocRef);
      console.log('Confirmed request deleted successfully');

    } catch (error) {
      console.error('Error submitting rating:', error);
    }

    this.loadDeniedRequests();
    this.loadConfirmedRequests();
    this.closeRatingModal();

  }
}
```

Programski kod 4.9. *Profile komponenta*

4.7. Kalendar komponenta

Programski kod 4.10. prikazuje kalendar komponentu i omogućuje upravljanje zahtjevima i kartama povezanima s korisnikom. Kalendar se generira za trenutni mjesec, uključujući zamjenske dane za početak i kraj mjeseca. Korisnik može navigirati između mjeseci, a odabrani datum i povezani zahtjevi prikazuju se na kalendaru.

Komponenta koristi Firebase za pohranu podataka i biblioteku *date-fns* za manipulaciju datumima. Učitava zahtjeve iz Firestore baze podataka, organizira ih po datumima i prikazuje samo zahtjeve sa statusom "Prihvaćeno". Funkcionalnosti uključuju odabir datuma, prikaz karata i zahtjeva te provjeru postoji li podatak za određeni datum.

```
isDateInSelectedRange(date: Date): boolean {
    if (!this.selectedDate) return false;

    const selectedRequest =
this.getTicketsOrRequestsForDate(this.selectedDate)
      .find(item => {
        if (!item.startDate || !item.endDate) return false;
        const startDate = new Date(item.startDate);
        const endDate = new Date(item.endDate);
        return startDate <= date && date <= endDate;
      });

    return !!selectedRequest;
  }

generateCalendar() {
    const start = startOfMonth(new Date(this.currentYear,
this.currentMonth));
    const end = endOfMonth(new Date(this.currentYear, this.currentMonth));
    const startWeekday = start.getDay();
    const endWeekday = end.getDay();

    let date = start;
    this.calendarDays = [];
      for (let i = 0; i < startWeekday; i++) {
      this.calendarDays.push({
        day: null,
        isToday: false,
        isSelected: false,
        isInSelectedRange: this.isDateInSelectedRange(date),
        date: null
      });
    }

    while (date <= end) {
      this.calendarDays.push({
        day: date.getMonth() === this.currentMonth ? date.getDate() : null,
        isToday: isToday(date),
        isSelected: isSameDay(date, this.selectedDate || new Date()),
```

```
        isInSelectedRange: false,
        date: date.getMonth() === this.currentMonth ? date : null
      });
      date = addDays(date, 1);
    }

    for (let i = endWeekday + 1; i < 7; i++) {
      this.calendarDays.push({
        day: null,
        isToday: false,
        isSelected: false,
        isInSelectedRange: false,
        date: null
      });
    }

    this.cdr.detectChanges();
  }
```

Programski kod 4.10. *Kalendar komponenta*

4.8. Service form komponenta

Programski kod 4.11 prikazuje *Service* form komponentu koja omogućuje kreiranje i uređivanje usluga pomoću obrazaca u Angular aplikaciji. Koristi reactive forms za unos podataka o usluzi, uključujući naziv, opis, kategoriju, vrijeme početka i završetka te trošak.

Obrazac se inicijalizira s potrebnim validacijama, uključujući provjeru obveznih polja i minimalnih vrijednosti za trošak. Ako se uređuje postojeća usluga, podaci se učitavaju u obrazac. Nakon što korisnik pošalje obrazac, podaci se pohranjuju u Firebase Firestore. Ako je usluga uspješno kreirana, korisnik dobiva obavijest putem *snackbar-a*, a zatim se preusmjerava na odgovarajuću stranicu.

```
export class ServiceFormComponent implements OnInit {
  @Input() service?: Service;
  serviceForm: FormGroup;
  isEditMode: boolean = false;

  constructor(
    private fb: FormBuilder,
    private serviceService: ServiceService,
    private auth: Auth,
    private router: Router,
    private snackBar: MatSnackBar
  ) {
    this.serviceForm = this.fb.group({
      name: ['', Validators.required],
      description: ['', Validators.required],
      category: ['', Validators.required],
      startTime: ['', Validators.required],
      endTime: ['', Validators.required],
      cost: ['', [Validators.required, Validators.min(0)]],
```

```
    });
  }

  ngOnInit(): void {
    if (this.service) {
      this.isEditMode = true;
      this.serviceForm.patchValue(this.service);
    }
  }

  async onSubmit() {
    const user = await this.auth.currentUser;
    if (user) {
      const serviceData: Service = {
        ...this.serviceForm.value,
        userId: user.uid,
      };

      if (this.isEditMode && this.service?.id) {
        await this.serviceService.updateService(this.service.id,
serviceData);
        this.router.navigate(['/profile-user']);
      } else {
        const serviceRef: DocumentReference = await
this.serviceService.createService(serviceData);
        this.snackBar.open('Service created successfully!', 'Close', {
          duration: 5000,
          panelClass: ['success-snackbar']
        });
        this.router.navigate(['/profile-worker']);
      }
    }
  }
}
```

Programski kod 4.11. *Service form komponenta*

4.9. Ticket form komponenta

Programski kod 4.12. prikazuje *Ticket* form komponentu koja omogućuje kreiranje i uređivanje karata u Angular aplikaciji putem obrazaca. Koristi reactive forms za unos podataka o kartama uključujući naziv, kategoriju, opis, datum početka i datum završetka. Obrazac se inicijalizira s validacijom koja osigurava da su sva polja obavezna te provjerava ispravnost datuma početka i završetka.

Ako se uređuje postojeća karta, podaci se učitavaju u obrazac. Nakon što korisnik pošalje obrazac, podaci se pohranjuju u Firebase Firestore. Ako je karta uspješno kreirana, korisnik dobiva obavijest putem *snackbar-a*, a zatim se preusmjerava na stranicu profila.

Korisnički definirana validacija osigurava da su datumi ispravno unešeni, provjeravajući da datum završetka ne bude prije datuma početka.

```
export class TicketFormComponent implements OnInit {
  @Input() ticket?: Ticket;
  ticketForm: FormGroup;
  isEditMode: boolean = false;

  constructor(
    private fb: FormBuilder,
    private ticketService: TicketService,
    private auth: Auth,
    private router: Router,
    private snackBar: MatSnackBar
  ) {
    this.ticketForm = this.fb.group({
      name: ['', Validators.required],
      category: ['', Validators.required],
      description: ['', Validators.required],
      startDate: ['', [Validators.required, startDateValidator()]],
      endDate: ['', [Validators.required]]
    }, { validator: this.dateValidator });
  }

  ngOnInit(): void {
    if (this.ticket) {
      this.isEditMode = true;
      this.ticketForm.patchValue(this.ticket);
    }
  }

  dateValidator(formGroup: FormGroup): ValidationErrors | null {
    const startDateCtrl = formGroup.get('startDate');
    const endDateCtrl = formGroup.get('endDate');

    if (!startDateCtrl || !endDateCtrl) {
      return null;
    }

    const startDateValue = startDateCtrl.value;
    const endDateValue = endDateCtrl.value;

    const startDateError = startDateValidator()(startDateCtrl);
    const endDateError = endDateValidator(startDateCtrl)(endDateCtrl);

    const errors = {
      ...startDateError,
      ...endDateError
    };

    return Object.keys(errors).length ? errors : null;
  }

  async onSubmit() {
    const user = await this.auth.currentUser;
    if (user) {
      const ticketData: Ticket = {
        ...this.ticketForm.value,
        userId: user.uid,
        userName: user.displayName
```

```
      };

      if (this.isEditMode && this.ticket?.id) {
        await this.ticketService.updateTicket(this.ticket.id, ticketData);
        this.router.navigate(['/profile-user']);
      } else {
        const ticketRef: DocumentReference = await
this.ticketService.createTicket(ticketData);
        this.snackBar.open('Ticket created successfully!', 'Close', {
          duration: 5000,
          panelClass: ['success-snackbar']
        });
        this.router.navigate(['/profile-user']);
      }
    }
  }
}
```

Programski kod 4.12. *Ticket form komponenta*

4.10. Header komponenta

Programski kod 4.13. prikazuje *header* komponentu koja omogućuje prikazivanje zaglavlja aplikacije s funkcionalnostima za autentifikaciju korisnika i upravljanje sesijama.

Komponenta koristi Angular Firebase za autentifikaciju i Firestore za dohvaćanje profila korisnika. Kada se korisnik prijavi, komponenta provjerava njegove podatke u kolekcijama za korisnike, radnike i administratore. Na temelju rezultata postavlja se odgovarajući tip korisnika i ažurira stanje. Ako korisnik pripada administratorima, preusmjerava se na administratorsku ploču.

Komponenta također omogućuje korisniku da se odjavi, nakon čega se prikazuje obavijest o uspješnoj odjavi.

Svi zahtjevi i navigacija unutar komponente koriste asinkrone metode kako bi se osigurao pravilan prikaz i upravljanje stanjem korisnika.

```
export class HeaderComponent implements OnInit {
  user$ = new BehaviorSubject<UserProfile | null>(null);
  userType: 'user' | 'worker' | 'admin' | undefined;
  isLoading = true;

  ngOnInit(): void {
    onAuthStateChanged(this.auth, (user) => {
      if (user) {
        this.fetchUserProfile(user.uid);
      } else {
        this.user$.next(null);
        this.isLoading = false;
      }
    });
```

```
  }

  fetchUserProfile(userId: string) {
    this.isLoading = true;

    let userRef = doc(this.firestore, `users/${userId}`);

    docData<UserProfile>(userRef).pipe(
      switchMap((userProfile: UserProfile | undefined) => {
        if (userProfile) {
          this.user$.next(userProfile);
          this.userType = 'user';
          return of(userProfile);
        } else {
          userRef = doc(this.firestore, `workers/${userId}`);
          return docData<UserProfile>(userRef).pipe(
            switchMap((workerProfile: UserProfile | undefined) => {
              if (workerProfile) {
                this.user$.next(workerProfile);
                this.userType = 'worker';
                return of(workerProfile);
              } else {
                userRef = doc(this.firestore, `admins/${userId}`);
                return docData<UserProfile>(userRef).pipe(
                  switchMap((adminProfile: UserProfile | undefined) => {
                    if (adminProfile) {
                      this.user$.next(adminProfile);
                      this.userType = 'admin';
                      console.log(this.userType);
                      this.router.navigate(['/admin-board']);
                      return of(adminProfile);
                    } else {
                      this.user$.next(null);
                      return of(null);
                    }
                  })
                );
              }
            })
          );
        }
      }),
      catchError((error) => {
        console.error('Error fetching user profile:', error);
        this.user$.next(null);
        return of(null);
      })
    ).subscribe(() => {
      this.isLoading = false;
    });
  }

  logout(): void {
    signOut(this.auth).then(() => {
      console.log('User logged out');
      window.location.href = '/login';
      this.snackBar.open('Logging out!', 'Close', {
```

```
        duration: 5000,
        panelClass: ['success-snackbar']
      });
    }).catch(error => {
      console.error('Logout error:', error);
    });
  }
}
```

Programski kod 4.13. *Header komponenta*

4.11. Servisi aplikacije

Servisi unutar angular aplikacije služe za odvajanje podataka i funkcija koji se koriste u raznim komponentama aplikacije. Da bi se koristio unutar komponente servis treba biti definiran kao ubrizgavajuć.

4.11.1. Auth service

Programski kod 4.14. definira Angular uslugu za slanje e-maila za resetiranje lozinke koristeći Firebase Authentication. Metoda *sendPasswordResetEmail* prima korisnički e-mail i šalje zahtjev za promjenu lozinke, vraćajući *Observable* za asinkrono rukovanje. Usluga je singleton, što znači da postoji samo jedan instanca usluge u cijeloj aplikaciji dok *Observable* omogućuje asinkroni rad s podacima i obavještavanje pretplatnika kada se podaci promjene.

```
export class AuthService {
  constructor(private auth: Auth) {}
  sendPasswordResetEmail(email: string) {
    return from(sendPasswordResetEmail(this.auth, email));
  }
}
```

Programski kod 4.14. *Authorization servis*

4.11.2. Service service

Programski kod 4.15. prikazuje Angular uslugu za upravljanje podacima usluga u Firebase Firestore-u. Usluga omogućuje kreiranje, ažuriranje i brisanje usluga. Metoda *createService* dodaje novu uslugu u kolekciju, *updateService* ažurira postojeću uslugu prema jedinstvenoj oznaci, a *deleteService* briše uslugu prema jedinstvenoj oznaci. Svaka metoda koristi Firebase Firestore API za interakciju s podacima u bazi.

```
export class ServiceService {

  constructor(private firestore: Firestore) {}

  createService(serviceData: Service) {
    const servicesCollection = collection(this.firestore, `services`);
    // Spread the serviceData into a plain object
```

```
    return addDoc(servicesCollection, { ...serviceData });
  }

  updateService(serviceId: string, serviceData: Service) {
    const serviceDoc = doc(this.firestore, `services/${serviceId}`);
    // Spread the serviceData into a plain object
    return updateDoc(serviceDoc, { ...serviceData });
  }

  deleteService(id: string): Promise<void> {
    const ticketDocRef = doc(this.firestore, `services/${id}`);
    return deleteDoc(ticketDocRef);
  }
}
```

Programski kod 4.15. *Service servis*

4.11.3. Ticket service

Programski kod 4.16. implementira *TicketService* za upravljanje kartama u Firestore bazi podataka. Usluga omogućuje osnovne operacije kreiranja, ažuriranja, brisanja i dohvaćanja karata. Ove operacije koriste Firestore za pohranu i *Observable* za asinkrono dohvaćanje podataka.

```
export class TicketService {
  private ticketsCollection = collection(this.firestore, 'tickets');

  constructor(private firestore: Firestore, private auth: Auth) {}

  createTicket(ticket: Ticket): Promise<DocumentReference> {
    return addDoc(this.ticketsCollection, ticket);
  }

  getTicketsByUser(userId: string): Observable<Ticket[]> {
    const userTicketsQuery = query(this.ticketsCollection, where('userId',
'==', userId));
    return collectionData(userTicketsQuery, { idField: 'id' }) as
Observable<Ticket[]>;
  }

  updateTicket(id: string, ticket: Ticket): Promise<void> {
    const ticketDocRef = doc(this.firestore, `tickets/${id}`);
    return updateDoc(ticketDocRef, { ...ticket });
  }

  deleteTicket(id: string): Promise<void> {
    const ticketDocRef = doc(this.firestore, `tickets/${id}`);
    return deleteDoc(ticketDocRef);
  }

  async getTickets(): Promise<Ticket[]> {
    const ticketsRef = collection(this.firestore, 'tickets');
    const querySnapshot = await getDocs(ticketsRef);
    return querySnapshot.docs.map(doc => {
      const data = doc.data();
      return {
        id: doc.id,
```

```
        name: data['title'],
        description: data['description'],
        category: data['category'],
        startDate: data['startDate'],
        endDate: data['endDate'] // Ensure this is stored as a string or
formatted date
      } as Ticket;
    });
  }
}
```

Programski kod 4.16. *Ticket servis*

4.11.4. User service

Programski kod 4.17. implementira *UserService* koji upravlja korisničkim profilima i njihovim fotografijama na Firebase-u, kao i korisnikovim kartama. Koristi Angularove usluge i Firebase funkcionalnosti za interakciju s podacima.

Servis omogućava dohvaćanje korisničkog profila s Firebase Firestore baze podataka, ažuriranje korisničkog profila s novim podacima, učitavanje profilne slike korisnika u Firebase Storage i vraćanje URL slike, dohvaćanje karata povezanih s korisnikom, brisanje korisničkog profila i svih karata povezanih s korisnikom te brisanje korisničkog računa iz Firebase Authentication-a.

```
export class UserService {
  constructor(
    private firestore: Firestore,
    private storage: Storage,
    private auth: Auth
  ) {}

  getUserProfile(userId: string): Observable<UserProfile> {
    const userRef = doc(this.firestore, `users/${userId}`);
    return docData<UserProfile>(userRef);
  }

  updateUserProfile(userId: string, profileData: Partial<UserProfile>):
Promise<void> {
    const userRef = doc(this.firestore, `users/${userId}`);
    return updateDoc(userRef, profileData);
  }

  uploadProfilePicture(userId: string, file: File): Promise<string> {
    const storageRef = ref(this.storage,
`profile_pictures/${userId}/${file.name}`);
    return uploadBytes(storageRef, file).then(() =>
getDownloadURL(storageRef));
  }

  getUserTickets(userId: string): Promise<Ticket[]> {
    const ticketsRef = collection(this.firestore, 'tickets');
    const q = query(ticketsRef, where('userId', '==', userId));
    return getDocs(q).then(querySnapshot =>
```

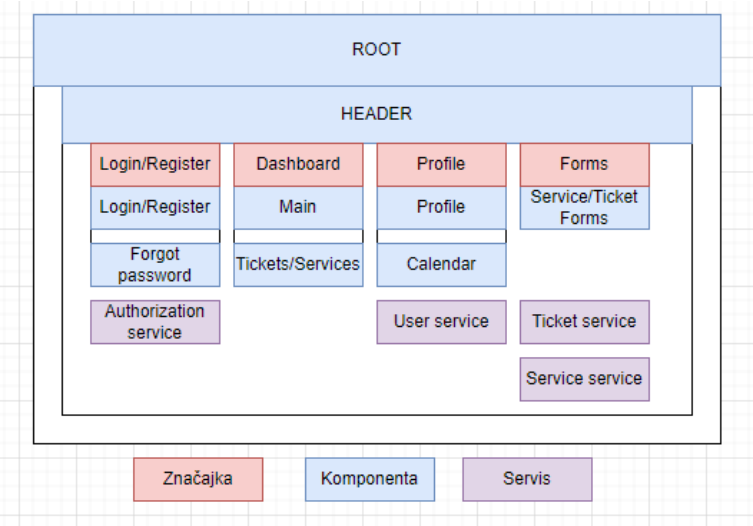
```
      querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() as Ticket
}))
    );
  }

  deleteUser(userId: string): Promise<void> {
    const userRef = doc(this.firestore, `users/${userId}`);
    return deleteDoc(userRef);
  }

  deleteUserTickets(userId: string): Promise<void[]> {
    const ticketsRef = collection(this.firestore, 'tickets');
    const q = query(ticketsRef, where('userId', '==', userId));
    return getDocs(q).then(querySnapshot =>
      Promise.all(querySnapshot.docs.map(doc => deleteDoc(doc.ref)))
    );
  }

  deleteAccount(): Promise<void> {
    return this.auth.currentUser?.delete() ?? Promise.reject('No current
user');
  }
}
```

Programski kod 4.17. *User service*

5. IZGLED WEB APLIKACIJE

U ovom poglavlju detaljno su predstavljeni izgledi i funkcionalnosti web aplikacije, kako za korisnike, tako i za radnike. Aplikacija nudi intuitivno sučelje prilagođeno potrebama različitih korisničkih uloga, čime osigurava jednostavnu i učinkovitu interakciju s platformom.

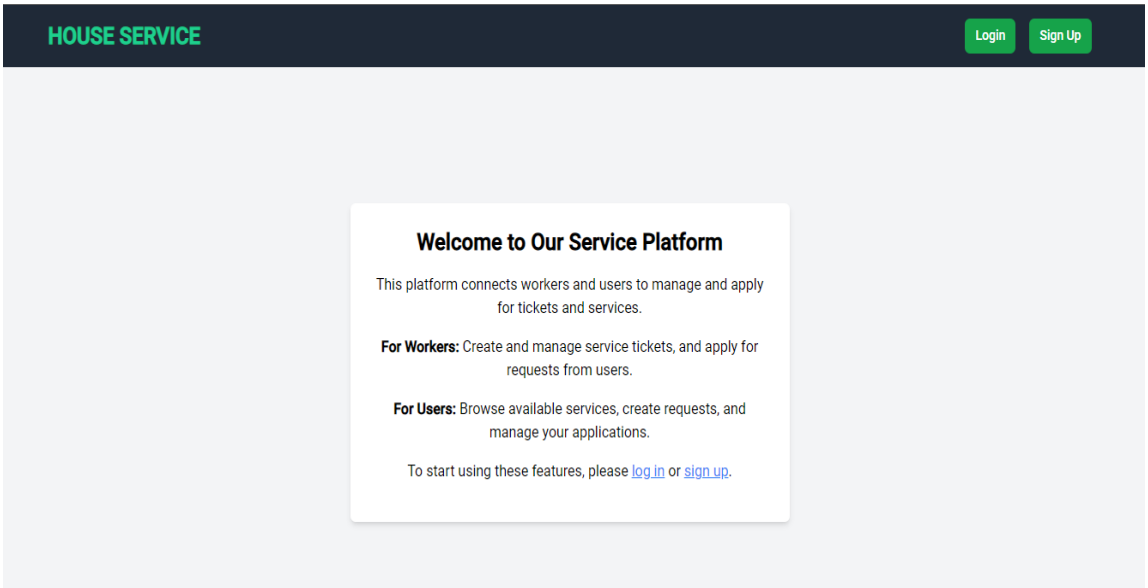
Slika 5.1. prikazuje dijagram toka aplikacije House service te njezine značajke i modele.



Slika 5.1. *Dijagram toka aplikacije House service*

5.1. Izgled dashboard značajke

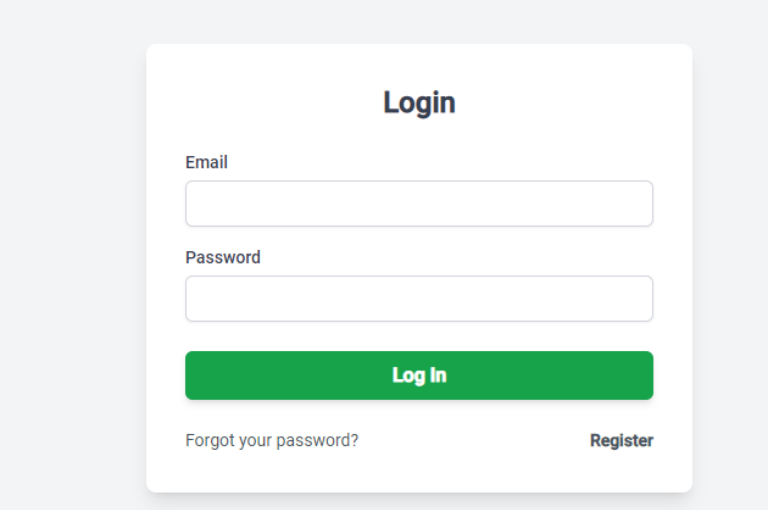
Slika 5.2. prikazuje glavni ekran aplikacije gdje korisnik dobiva infromacije o primarnoj funkciji ove aplikacije. Tu je korisnik upućen sa sljedećim korakom da ako želi koristiti aplikaciju da se prvobitno mora prijaviti ili registrirati se kako bi stvorio novi račun.



Slika 5.2. *Glavni ekran aplikacije „dashboard“*

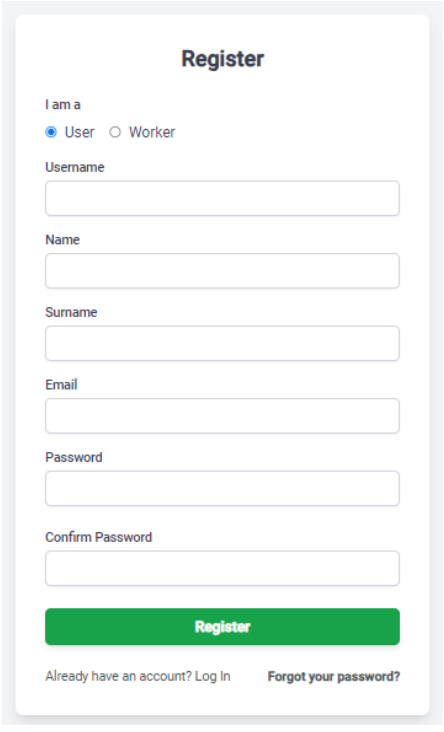
5.2. Izgled prijave i registracije

Slika 5.3. prikazuje komponentu gdje korisnik unosi svoj email i lozinku za prijavu u postojani račun. Također se može stvoriti novi račun ili ponovno postaviti zaboravljenu lozinku.



Slika 5.3. *Login komponenta*

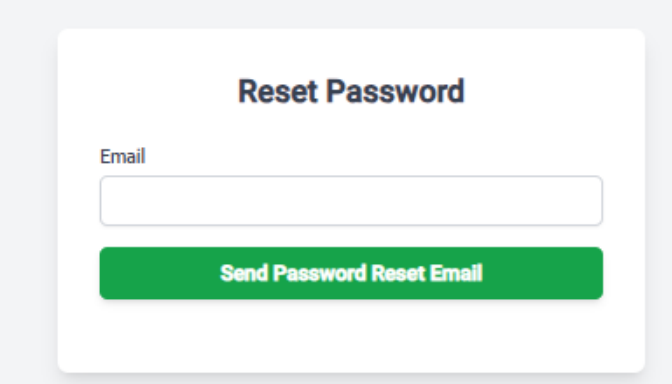
Slika 5.4. prikazuje komponentu gdje korisnik može stvoriti račun za aplikaciju *House Service*. Odabire se korisničko ime s čime se korisnik predstavlja u aplikaciji, te ostale informacije potrebne poput imena, prezime i slično. Korisnik isto može izabrati koji je tip korisnika, je li običan korisnik koji zahtijeva potrebne usluge ili tip radnika koji ispunjava zahtjeve.



Slika 5.4. *Register komponenta*

5.2.1 Izgled forgot password komponente

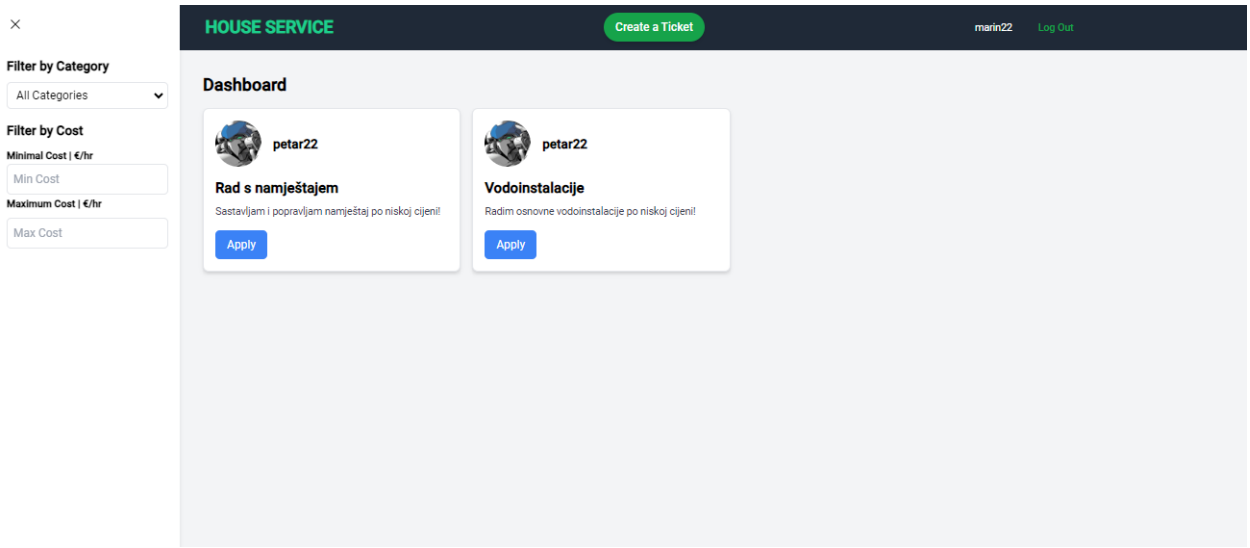
Slika 5.5. prikazuje reset password komponentu gdje korisnik unosi email na koji se šalje zahtjev za promjenu lozinke.



Slika 5.5. Reset password komponenta

5.3. Izgled ekrana za korisnika

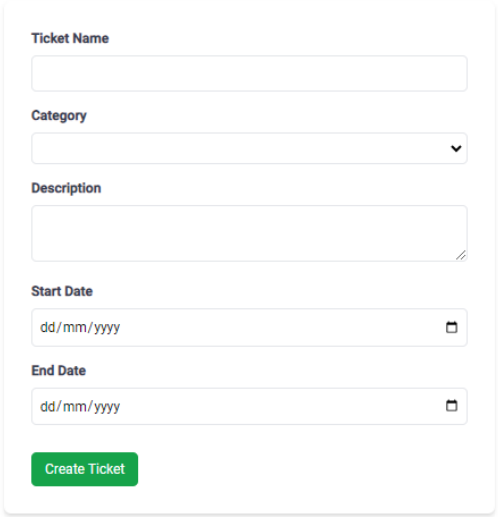
Slika 5.6. prikazuje sučelje kada je korisnik prijavljen. Ponuđene su značajke poput filtriranja servisa, pregled servisa od radnika, prijava na servis klikom na poželjni servis, odlazak na stranicu za kreiranje zahtjeva kojeg korisnik treba od radnika te odlazak na svoj profil.



Slika 5.6. *Izgled komponente dashboard za korisnika*

5.3.1. Izgled komponente za kreiranje zahtjeva

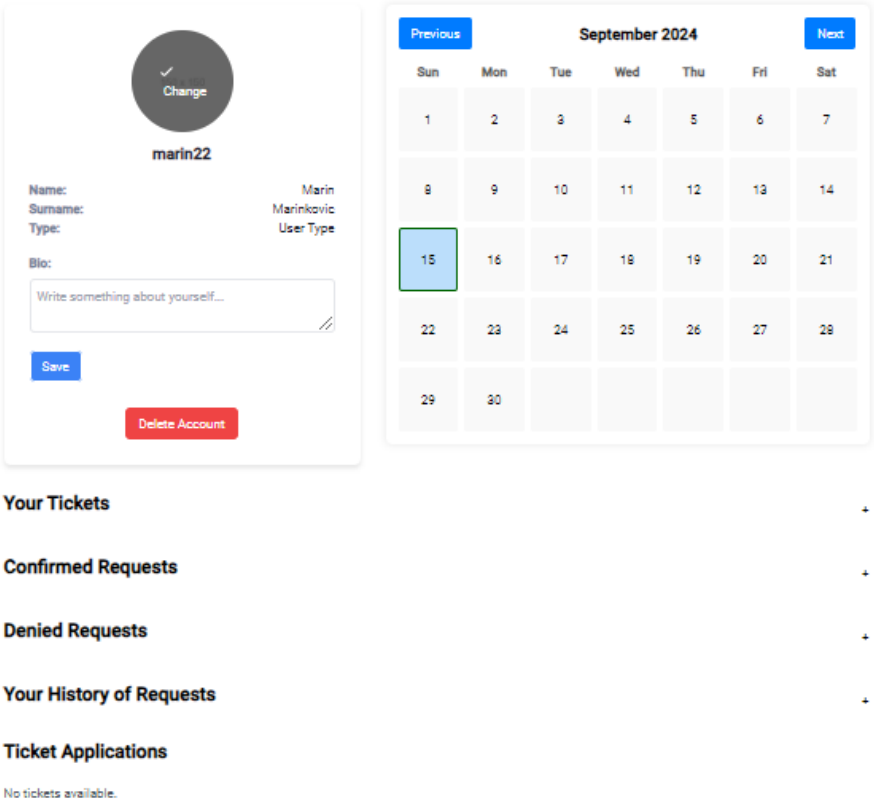
Slika 5.7. prikazuje formu kojom korisnik izrađuje zahtjev pomoću kojeg radnici mogu vidjeti što je potrebno korisniku. Ovdje korisnik definira ime, kategoriju, opis te datum zahtjeva kojeg zahtijeva.



Slika 5.7. *Izgled forme za izradu korisničkog zahtjeva*

5.3.2. Izgled komponente profil za korisnika

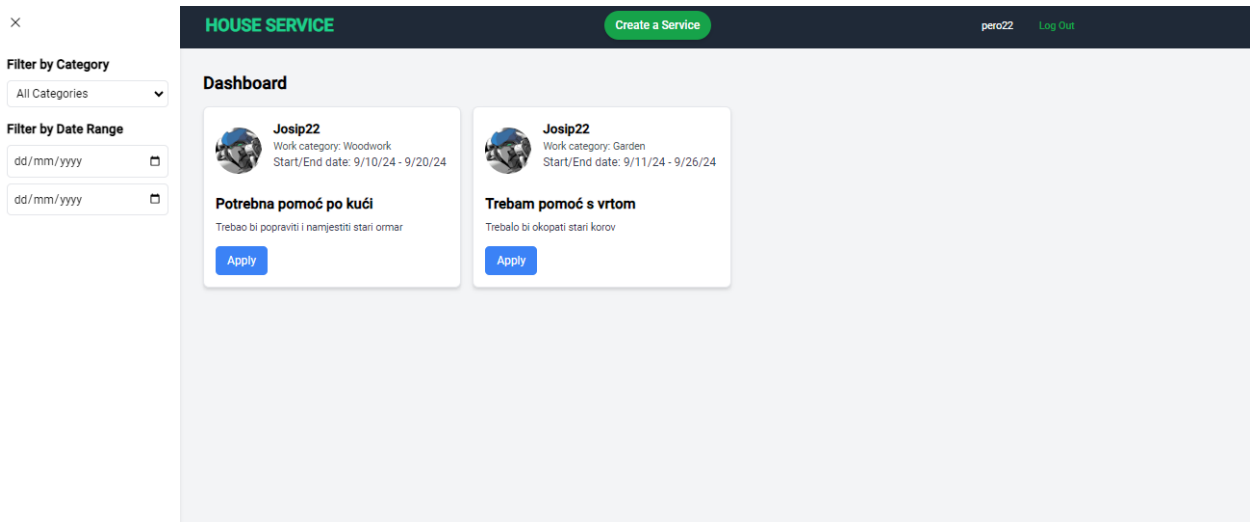
Slika 5.8. prikazuje komponentu profile. Korisnik može vidjeti svoje zahtjeve, prihvaćene i odbijene te povijest zahtjeva. Uz to mogu vidjeti radnike koji su se prijavili na korisnikove zahtjeve i prihvatiti ili odbiti prijave na korisnikov zahtjev. Korisnik također ima i kalendar koji prikazuje prihvaćene zahtjeve za taj datum. Korisnici također mogu promijeniti profilnu sliku i opis profila.



Slika 5.8. *Izgled komponente profile*

5.4. Izgled ekrana za radnika

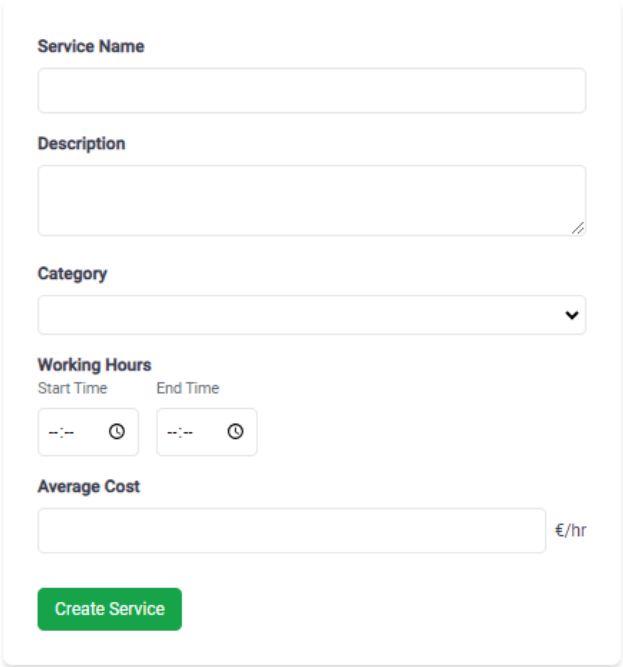
Slika 5.9. prikazuje sučelje prijavljenog radnika. Na sučelju su ponuđene značajke poput filtriranja servisa, pregled korisničkih zahtjeva, mogućnost prijave na zahtjev, odlazak na stranicu za stvaranje servisa kojeg radnik ima u ponudi te odlazak na svoj profil.



***Slika 5.9**. Izgled komponente dashboard za radnike*

5.4.1. Forma za izradu servisa

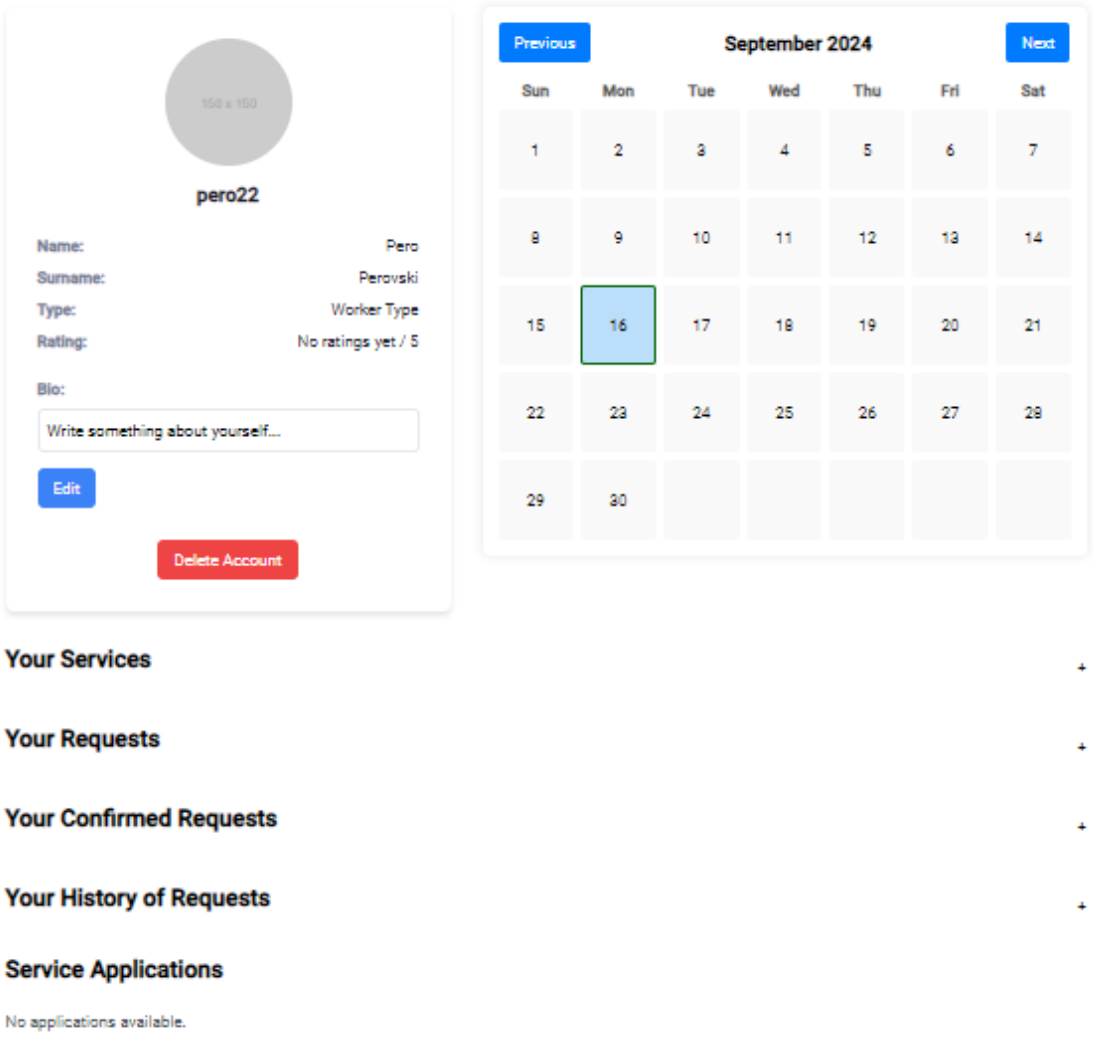
Slika 5.10. prikazuje formu u kojoj radnik unosi ime, opis, kategoriju, radne sate i prosječnu cijenu servisa po satu. Korisnik može napravljeni servis naknadno vidjeti na svom glavnom ekranu aplikacije prikazano na slici 5.6.



***Slika 5.10.** Izgled forme za izradu servisa*

5.4.2. Izgled komponente profile za radnike

Slika 5.11. prikazuje komponentu profile gdje radnik može vidjeti svoje servise, prihvaćene zahtjeve, odbijene zahtjeve, povijest zahtjeva i sve zahtjeve na koje se radnik prijavio. Također može prihvatiti ili odbiti zahtjeve korisnika. Radnik ima i kalendar koji prikazuje prihvaćene zahtjeve za taj datum. Radnici isto mogu promijeniti profilnu sliku i opis profila.



Slika 5.11. *Izgled profilne komponente za radnike*

6. ZAKLJUČAK

U današnjem svijetu, posebno se cijeni pristupačnost i funkcionalnost web aplikacija. Ovaj završni rad izrađen je u Angular okruženju kako bi se prikazale glavne značajke koje bi kvalitetna web platforma za kućne popravke trebala sadržavati, ali i kako bi se istaknule mogućnosti samog Angular okruženja. Angular se pokazao kao vrlo pogodan alat, osobito kada se koristi veći broj web servisa i modula. Njegova glavna prednost je *single-page* pristup, gdje se cjelokupna aplikacija odvija na jednoj stranici, dok se samo mijenja prikaz. Ovo omogućava brže učitavanje stranice te ponovnu uporabu dijelova koda, odnosno komponenti.

Web aplikacija za kućne popravke omogućuje korisnicima pregled ponuda, brz pristup detaljima tih ponuda te olakšava komunikaciju s radnicima. Funkcionalnosti aplikacije su implementirane uz pomoć Angular servisa, što omogućuje prikaz podataka, dinamičko popunjavanje obrazaca te filtriranje prikaza usluga na temelju unesenih kriterija korisnika.

LITERATURA

[1] Fiverr.com, https://www.fiverr.com/ [stranica posjećena 30.05.2023.]

[2] Upwork.com, https://www.upwork.com/ [stranica posjećena 30.05.2023]

[3] Toptal.com, https://www.toptal.com/ [stranica posjećena 30.05.2023]

[4] D. Goodman, **Dynamic HTML: The Definitive Reference, Second Edition,** O'Reilly & Associates, September 2002

[5] D. Goodman, M. Morrison, P. Novitski, T. G. Rayl, **JavaScript Bible, Seventh Edition,** Wiley Publishing, 2010

[6] Angular.io, https://angular.io/docs [stranica posjećena 30.05.2023.]

[7] Bootstrap, https://getbootstrap.com/docs/4.1/getting-started/introduction/ [stranica posjećena 15.09.2024.]

[8] L. Moroney, **The Definitive Guide to Firebase**, Apress, 2017

[9] Firebase Documentation, https://firebase.google.com/docs [stranica posjećena 30.05.2023.]

SAŽETAK

Web aplikacija za kućne popravke

Web aplikacija za kućne popravke koristi tehnologije poput Angular-a, Firebase-a i JavaScript-a za izradu dinamičnog korisničkog sučelja, sigurnu komunikaciju klijent-server te upravljanje podacima. Angular pruža responzivno sučelje, dok Firebase omogućava autentifikaciju korisnika i rad s bazom podataka u stvarnom vremenu. Aplikacija korisnicima nudi pretraživanje, kreiranje, uređivanje i ocjenjivanje servisa, demonstrirajući integraciju ovih tehnologija u funkcionalan sustav za kućne popravke.

Ključne riječi: Angular, JavaScript, Firebase.

ABSTRACT

Web application for home repairs

The home repair web application uses technologies like Angular, Firebase, and JavaScript to create a dynamic user interface, secure client-server communication, and data management. Angular provides a responsive interface, while Firebase enables user authentication and real-time database functionality. The application allows users to search, create, edit, and rate services, demonstrating the integration of these technologies into a functional system for home repairs.

Keywords: Angular, JavaScript, Firebase.