Aplikacija za vođenje i savjetovanje pri reciklaži materijala

Ivković, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA VOĐENJE I SAVJETOVANJE PRI RECIKLAŽI MATERIJALA

Diplomski rad

Ivan Ivković

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 30.08.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ivan Ivković
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1058R, 06.10.2019.
OIB studenta:	28570107984
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Izv.prof.dr.sc. Zdravko Krpić
Naslov diplomskog rada:	Aplikacija za vođenje i savjetovanje pri reciklaži materijala
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
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: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	30.08.2022.

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, 14.09.2022.

Ime i prezime studenta:	Ivan Ivković
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1058R, 06.10.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za vođenje i savjetovanje pri reciklaži materijala**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivica Lukić

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

Zaštita okoliša jedan je od najvećih izazova s kojima se danas suočava čovječanstvo. Stoljeća razvoja i napretka na svim područjima ljudskog života su, osim prednosti, ponajprije u smislu olakšavanja svakodnevnice, donijela i mnogobrojne probleme, u prvom redu one vezane za okoliš, koji mogu poremetiti ravnotežu u biosferi i na taj način otežati ili čak ugroziti budući život na planetu Zemlji. Onečišćenje zraka i voda, globalno zagrijavanje, ozonske rupe i kisele kiše samo su neke od posljedica štetnog ispuštanja raznih spojeva iz industrijskih pogona, vozila i kućanstava. Osim što se pri proizvodnji dobara za svakodnevnu uporabu ispuštaju štetni spojevi, porasla je svijest i da je rasipanje Zemljinih zaliha krajnje neodrživo, budući da su mnoge sirovine već nepotrebno izgubljene, a još neke su na putu iscrpljivanja i mogućeg trajnog nestanka. Suočeni s navedenim problemima i pitanjima, ljudi su s vremenom počeli osjećati odgovornost za budućnost Zemlje te neprestano pokušavaju pronaći načine rješavanja ekoloških problema. Jedan od njih je recikliranje materijala, odnosno postupak obrade otpadnih materijala i iskorištenih proizvoda radi dobivanja sirovina i energije za ponovno iskorištavanje i uporabu [1]. Premda se pojam recikliranja često povezuje isključivo sa suvremenom djelatnosti zaštite okoliša, ono je zapravo postojalo tijekom većeg dijela ljudske prošlosti, ali prije svega kao posljedica nemogućnosti pronalaska izvorišta potrebnih resursa, a samim time i nedostatka novih resursa te njihove veće cijene. U suvremenom dobu recikliranje se provodi i u jednu i u drugu svrhu, odnosno u svrhu uštede prirodnih resursa i u svrhu financijskih ušteda pri proizvodnim procesima. Primjerice, recikliranje aluminija koristi 95% manje energije nego proizvodnja aluminija iz novih sirovina, odnosno jednom tonom recikliranog aluminija štedi se do 8 tona boksita, 14.000 kWh električne energije, 6.300 litara nafte i 7,6 kubnih metara prostora za odlaganje otpada [2]. Recikliranje papira, promatrano samo s ekološkog gledišta, vrlo je važno jer se recikliranjem jedne tone starog papira umjesto proizvodnje novog pridonosi očuvanju prosječno 17 stabala i uštedi približno 32.000 litre vode, a kad bi se tona papira zapalila umjesto preradila za ponovnu uporabu, u Zemljinu atmosferu otišlo bi oko 750 kilograma ugljikovog dioksida [3]. Osim metala i papira moguće je recikliranje plastike, stakla, organskog otpada i mnogih drugih materijala.

1.1. Zadatak diplomskog rada

Zadatak diplomskog rada je napraviti aplikaciju koja će korisnike voditi i savjetovati pri reciklaži različitih materijala. Unutar aplikacije korisnik treba za različite materijale saznati kako se i gdje se mogu reciklirati. Nadalje, svaki korisnik i svako reciklažno dvorište trebaju imati evidenciju koji su materijali reciklirani.

2. PREGLED PODRUČJA TEME

Postojeće aplikacije iz područja recikliranja materijala, barem one javno dostupne, uglavnom su usmjerene na korisnike komunalnih usluga, i to prije svega kao pomoć pri razvrstavanju otpada, koje prethodi samom recikliranju. Tako je za potrebe hrvatskih korisnika razvijena aplikacija Razvrstaj.me, čije je sučelje vrlo jednostavno za korištenje. Odmah po otvaranju na početnoj je stranici vidljiva tražilica u koju korisnici mogu upisati naziv predmeta za kojeg žele saznati u koji se spremnik odlaže. Već pri upisivanju počinju se pojavljivati podudarni pojmovi te se klikom na neki od njih otvara stranica s opisom otpada i uputom za odlaganje. Naprednija inačica aplikacije pruža dodatno i informacije specifične za različite davatelje komunalnih usluga, kojima je i namijenjena, kao što su opći podatci o pripadajućim reciklažnim dvorištima te raspored odvoza osnovnih skupina otpada, primjerice miješanog komunalnog otpada, papira i stakla [4].

Residus je aplikacija namijenjena korisnicima u španjolskoj pokrajini Kataloniji. Prvo korištenje započinje unosom i potvrdom imena odgovarajuće općine nakon čega se sadržaj najvećim dijelom počinje odnositi upravo na tu općinu, i to u smislu prikaza spremnika za razvrstavanje, dostupnih na odabranom području, kao i lokacija tamošnjih reciklažnih dvorišta [5]. Odabir općine može se promijeniti u bilo kojem trenutku, nakon čega se i informacije automatski prilagođavaju sukladno novom odabiru. Pomoć pri razvrstavanju otpadnih predmeta realizirana je kao tražilica u koju se unosi ime otpadnog predmeta te je, ako se isti nalazi među rezultatima filtriranja, moguć pristup pojedinostima o cjelokupnom procesu recikliranja, počevši od prikaza spremnika u koji je potrebno odložiti predmet, preko tehničkog opisa samog postupka, pa sve do krajnjeg rezultata. Vrlo slične funkcionalnosti ima i RecycleSmart, aplikacija napravljena za australsko tržište, kojem je potpuno prilagođena s obzirom da su obuhvaćene sve tamošnje jedinice lokalne samouprave [6]. Iz tog razloga je prije početka korištenja obvezno registrirati se te navesti adresu stanovanja i broj telefona.

Preostala dostupna rješenja imaju manje i jednostavnije funkcionalnosti. Na području Sjedinjenih Američkih Država koristi se iRecycle, na čijem se početnom korisničkom sučelju nalazi izbornik sa skupinama predmeta iz svakodnevnice. Ulaskom u neku od skupina prikazuje se pripadajući popis predmeta, a klikom na željeni predmet otvara se prikaz obližnjih reciklažnih dvorišta koja ga mogu prihvatiti [7]. Postoje također i aplikacije koje recikliranju pristupaju kroz prodavanje ili darivanje nepotrebnih predmeta, čime se omogućuje njihovo daljnje korištenje. Takav je primjer Telodoygratis, na raspolaganju prije svega korisnicima u Španjolskoj, kojima je omogućeno postavljanje oglasa za darivanje suvišnih predmeta [8].

3. PREGLED KORIŠTENIH TEHNOLOGIJA I ALATA

U ovom su poglavlju navedene i po potpoglavljima teoretski opisane tehnologije korištene tijekom izrade aplikacije. Za izradu klijentske strane korišten je Angular, za poslužiteljsku stranu Node.js, a baza podataka je PostgreSQL. Cjelokupna aplikacija izrađena je u razvojnom okruženju Visual Studio Code, pri čemu je tijekom razvoja poslužiteljske strane za testiranje korišten i Postman.

3.1. Angular

Angular je besplatni okvir otvorenog koda za razvoj dinamičkih jednostraničnih web aplikacija. Njegova prva inačica bila je pisana izravno u JavaScriptu i službeno se zvala AngularJS, a objavio ju je Google 2010. godine. Nakon četiri godine objavljena je druga inačica, značajno drukčija u odnosu na prvu, prije svega zbog korištenja TypeScripta umjesto JavaScripta, pa je iz naziva uklonjen dometak JS, a sve nadolazeće inačice počele su se označavati brojevima, pri čemu razlike među njima nisu bile izražene kao što je bio slučaj između prve i druge inačice.

TypeScript su razvili i 2012. godine prvi put objavili Microsoftovi stručnjaci kao alat za razvijanje velikih aplikacija, uključujući i JavaScript aplikacije za izvršavanje na klijentskoj i poslužiteljskoj strani. On je zapravo nadskup JavaScripta, što znači da su postojeći JavaScript programi ujedno i važeći TypeScript programi, a glavna razlika između dva programska jezika su opcionalna statička tipizacija i objektna orijentiranost TypeScripta. Naime, izostanak spomenutih dviju značajki kod JavaScripta rezultira slabom mogućnosti ponovnog korištenja dijelova programskog koda, odnosno otežanim održavanjem, što posebno dolazi do izražaja u slučaju velikih programa [9]. Kao objektno orijentiran programski jezik, TypeScript podrazumijeva mogućnost definiranja klasa i sučelja te svojstvo nasljeđivanja. Varijable unutar klasa definiraju se navođenjem željenog imena, dvotočke te željenog tipa podatka, što posebno olakšava razvoj aplikacije jer se na taj način mogu trenutačno otkriti i ispraviti sve pogrješke, uključujući i one najmanje.

Osnovna gradivna jedinica svake aplikacije je komponenta, koja se sastoji od klase, predloška i stilova [10]. Klasom se definiraju ponašanje i funkcionalnosti komponente, a neposredno iznad njene deklaracije obvezno se navodi dekorater @Component koji sadrži selektor (za definiranje načina korištenja komponente u predlošku), predložak (za određivanje prikaza sadržaja komponente) i stilove specifične za komponentu. Svaka aplikacija obvezno sadrži barem jednu komponentu, koja se obično naziva korijenska komponenta, a u koju se dodaju nove, odnosno kasnije stvorene komponente aplikacije. Predložak se u osnovi sastoji od HTML koda, kojem se po potrebi mogu dodati dodatne funkcionalnosti, što se postiže uporabom direktiva, odnosno klasa

koje proširuju ponašanja elemenata u aplikacijama, poput obrazaca, lista, stilova i dr. Osim komponenti, koje su same jedna vrsta direktiva, postoje još i atributne te strukturne direktive. Atributne direktive utječu na izgled i ponašanje nekog elementa ili komponente, a neke od najpoznatijih ugrađenih atributnih direktiva su NgClass, NgStyle i NgModel [11]. S druge strane, strukturne direktive, kao što primjerice NgIf, NgFor i NgSwitch, mijenjaju strukturu DOM-a dodavanjem, uklanjanjem ili mijenjanjem njegovih elemenata. U praktičnom dijelu rada neke od direktiva bit će korištene, pa će pritom biti i detaljnije objašnjene. Ondje će biti predstavljeni i preostali koncepti Angulara, kao što su vezanje podataka, moduli, servisi, ubrizgavanje ovisnosti i sl.

3.2. Node.js

Node.js je višeplatformsko izvršno okruženje otvorenog koda koje koristi virtualni stroj V8 za izvođenje programskog koda JavaScripta na poslužiteljskoj strani. Napisao ga je 2009. godine Ryan Dahl kao alternativu PHP-u, ASP-u i sličnim tehnologijama koje su uglavnom koristile sinkronu, odnosno blokirajuću paradigmu, zbog koje je izvođenje većih dijelova programskog koda bilo usporeno. Za razliku od njih, Node.js upravljan je događajima te koristi asinkronu paradigmu, zbog čega nema blokiranja, a zahvaljujući svojoj jednonitnosti sposoban je na učinkovit način rukovati tisućama istovremenim spojeva na jedan poslužitelj, što bi u višenitnom okruženju bilo podložno pogrješkama i također opterećujuće po performanse [12]. Osim navedenih prednosti u vezi s učinkovitosti izvođenja, Node.js je već samim svojim pojavljivanjem znatno u pozitivnom smislu utjecao i na cjelokupni razvoj web aplikacija jer je razvojnim programerima omogućio korištenje JavaScripta i za razvoj poslužiteljskih strana aplikacija, a ne samo klijentskih, kako je to prethodno bilo. Na taj način olakšano je i učenje i usavršavanje, budući da je programerima dovoljno nadograditi znanja iz JavaScripta i potom ih primijeniti na razvoju cjelokupne aplikacije, umjesto učiti potpuno drugi programski jezik. Također, rad s Node.js-om olakšavaju i mnogobrojni moduli, odnosno biblioteke, grupirani u pakete, koje je s jedne strane moguće objaviti, a s druge strane i instalirati te ažurirati kao ovisnosti u projektima, i to sve pomoću Node Package Managera, koji predstavlja standardnu podršku za upravljanje paketima, a poznatiji je pod kraticom npm. Korištenjem navedene kratice u naredbenom retku moguće je naredbom install odjednom u projekt instalirati sve ovisnosti ili samo određene pakete, kasnije ih ažurirati naredbom update i u konačnici pokretati zadatke naredbom run.

3.3. PostgreSQL

PostgreSQL je besplatan sustav za upravljanje relacijskim bazama podataka. Njegovi početci sežu u osamdesete godine 20. stoljeća kada je Michael Stonebraker s Kalifornijskog sveučilišta u Berkeleyju odlučio razviti svoju inačicu Ingresa, relacijske baze podataka na čijem se razvoju aktivno radilo prethodnih godina. Nova inačica nazvana je Postgres, a njena glavna odlika su objektno-relacijske značajke koje su dodane dotadašnjoj jezgri Ingresa. Do 1994. godine je kao interpreter korišten upitni jezik POSTQUEL koji je tada zamijenjen za SQL, pa je 1996. dotadašnji Postgres preimenovan u PostgreSQL. Iste godine postao je javno dostupan, a s obzirom da je otvorenog koda, od tada se neprestano razvija zahvaljujući velikom broju doprinositelja. Za razliku od MySQL-a koji je čisto relacijska, PostgreSQL je, kako je već spomenuto, objektno-relacijska baza podataka te kao takav omogućuje primjerice nasljeđivanje tablica, preopterećivanje funkcija, korisnički definirane tipove podataka i sl.

3.4. Visual Studio Code

Visual Studio Code je uređivač izvornog koda. Razvijen je u Microsoftu i predstavljen 2015. godine kao razvojno okruženje za operacijske sustave Windows, Linux i macOS. Zbog besplatnosti, jednostavnosti korištenja i brojnih drugih značajki postao je najpopularnijim razvojnim okruženjem. Osim korištenja različitih boja teksta i općenito naglašavanja sintakse, kojima je cilj povećanje preglednosti i čitljivosti programskog koda, sadrži i inteligentno dovršavanje koda, refaktoriranje, uklanjanje pogrješaka te mogućnost dodavanja različitih proširenja kojima se dodatno olakšava pisanje programskog koda za razvoj različitih aplikacija, kao i mnoge druge prednosti.

3.5. Postman

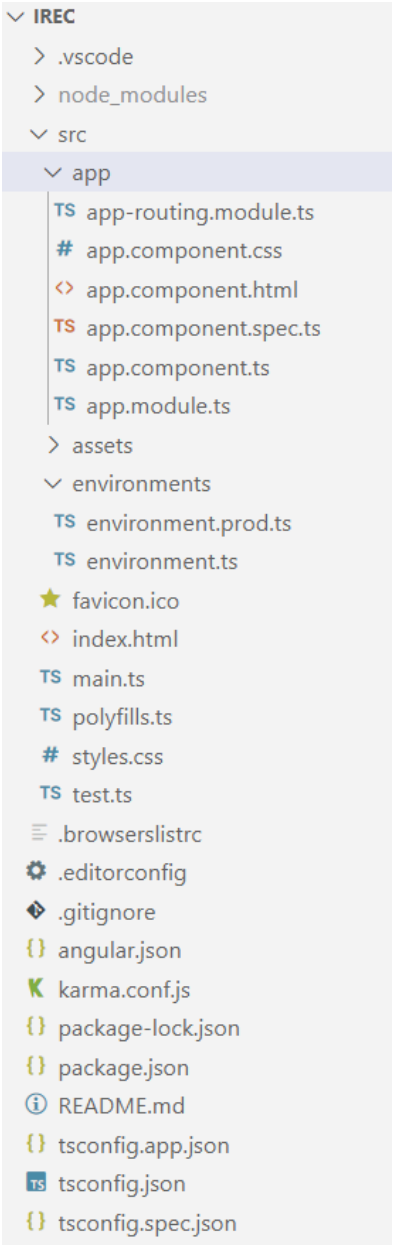
Postman je platforma za dizajniranje, izradu i testiranje API-ja. Pokrenuta je 2012. godine, prije svega s ciljem lakšeg testiranja API-ja, koje se obavlja postavljanjem adrese poslužitelja i odabirom odgovarajuće metode te unosom podataka nad kojima se izvodi metoda, poslije čega se, u formatu JSON, vraća uspješan ili neuspješan odgovor. Pritom je moguće stvaranje čitavih kolekcija poziva API-ja, čime se štedi vrijeme i dodatno olakšava testiranje.

4. IZRADA APLIKACIJE

Postupak izrade aplikacije sastoji se od nekoliko koraka koji su, svaki zasebno, opisani i objašnjeni u sljedećim potpoglavljima.

4.1. Inicijalizacija projekta i datotečna struktura

Izrada aplikacije na klijentskoj strani počinje stvaranjem projekta u Angularu, odnosno unosom naredbe ng new Irec, gdje Irec predstavlja željeno ime projekta. Njome se u novostvoreni radni prostor instaliraju najnužniji paketi i ovisnosti, sadržani u datotečnoj strukturi prikazanoj slikom 4.1.



Slika 4.1. *Datotečna struktura projekta u Angularu*

Prethodno spomenuti npm paketi nalaze se u direktoriju node_modules i na taj su način dostupni u cijelom radnom prostoru Irec. Direktorij src osobito je važan jer se u njemu nalaze konfiguracijske datoteke te sve izvorne datoteke aplikacije, djelomično raspoređene po poddirektorijima app, assets i environments. Sva logika projekta sadržana je u poddirektoriju app, koji u početku sadrži samo datoteku app-routing.module.ts te ostale datoteke vezane za korijensku komponentu AppComponent: app.component.css, app.component.html, app.component.spec.ts, app.component.ts i app.module.ts. Datoteka app-routing.module.ts predstavlja modul za usmjeravanje, koje u jednostraničnim aplikacijama omogućuje promjenu pogleda. Pogledi se izmjenjuju pomoću ruta koje je u datoteci potrebno definirati i navesti. Što se tiče same korijenske komponente, datoteka app.component.css definira njen stil, a datoteka app.component.html definira HTML predložak, odnosno raspored elemenata za pogled korijenske komponente. Nadalje, u app.component.ts nalazi se logika za AppComponent, dok app.component.spec.ts sadrži definiciju unit testa za komponentu. Za aplikacije izrađene u Angularu kaže se da su modularne, odnosno da se sastoje od modula, koji ujedinjavaju komponente aplikacije tvoreći tako funkcionalne cjeline, pa se datotekom app.module.ts definira korijenski modul AppModule unutar kojeg se obvezno deklariraju sve komponente aplikacije [13]. Također, kako se u nastavku izrade aplikacije budu stvarale nove komponente i drugi potrebni dijelovi, sve će se dodavati u poddirektorij app i stoga prethodno navedeno nije njegova konačna struktura. U poddirektorij assets spremaju se dodatci poput slika i sl., a poddirektorij environments sadrži konfiguracijske postavke za određena okruženja, kojima se po potrebi mogu dodati i nova. Od preostalih datoteka direktorija src potrebno je istaknuti index.html, odnosno glavnu HTML stranicu aplikacije, zatim main.ts, glavnu ulaznu točku aplikacije, koja prevodi aplikaciju i korijenskom modulu omogućuje izvođenje u pregledniku, i na kraju styles.css, kojom se mogu postaviti globalne stilske značajke, primjenjive na razini cijele aplikacije.

4.2. Korisničko sučelje za pomoć pri razvrstavanju otpada

Odvajanje i razvrstavanje otpada preduvjet je i zapravo prvi korak u recikliranju materijala. Radi se o relativno jednostavnom postupku koji je u Hrvatskoj obvezan od 2013. godine, a najučinkovitiji je kada se provodi već od najosnovnijih razina društva, odnosno od kućanstava. S obzirom da je zakonska obveza razvrstavanja otpada relativno nova pojava, ono među stanovništvom još uvijek nije zaživjelo u potpunosti i na odgovarajući način. S jedne strane, razlog je zastoj u opremanju kućanstava spremnicima za odvojeno prikupljanje otpada koji se dogodio u nekim jedinicama lokalne samouprave. S druge strane, korisnici koji su već dobili navedene spremnike nerijetko ne znaju u koji spremnik trebaju odložiti neke otpadne predmete. U pravilu se

koriste spremnici plave, žute, smeđe i zelene boje, te je načelno poznato da se u plave spremnike odlažu papir i karton, u žute spremnike plastika, u smeđe spremnike biootpad, a u zelene spremnike miješani komunalni otpad. No, često nastaju poteškoće, ponajprije s razvrstavanjem otpada za koje se ne može sa sigurnošću odrediti od kojeg su materijala načinjeni i mogu li se uopće reciklirati.

Kao odgovor na prethodan problem i pitanje ispravnog razvrstavanja otpada u kućanstvima, aplikacija svim svojim korisnicima, uključujući i neregistrirane, nudi korisničko sučelje u kojem oni jednostavnim unošenjem imena željenog otpadnog predmeta mogu dobiti informaciju u koji ga je spremnik potrebno odložiti. Ono je prije svega zamišljeno kao tablični prikaz u kojem svaki redak sadrži naziv predmeta, kategoriju otpada kojoj predmet pripada i najvažnije, spremnik u koji se odlaže promatrani predmet. S obzirom na nepraktičnost samostalnog pretraživanja velikog broja predmeta u tablici, uz nju je poželjno dodati i traku za filtriranje unosa, u koju korisnik može početi unositi naziv željenog predmeta kako bi se potom prikazao samo odgovarajući redak tablice. Za početak, unosom naredbe ng generate component components/wasteItems stvorena je komponenta wasteItems, a kako bi se održala preglednost datotečne strukture, stvoren je i poddirektorij components, koji će sadržavati sve naknadno stvorene komponente aplikacije. Slikom 4.2. prikazan je sadržaj HTML predloška komponente wasteItems.

```
<div class="col-md-6">
    <h4>Kako razvrstavati?</h4>
    <input name="searchTerm" class="form-control" type="text" placeholder="Unesite naziv otpadnog predmeta" [(ngModel)]="searchTerm" (input)="search(searchTerm)" />
</div>

<div>
    <table class="table">
      <thead>
        <tr>
          <th>predmet</th>
          <th>kategorija</th>
          <th>spremnik</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let wasteItem of wasteItems | searchFilter: searchTerm">
          <td>{{ wasteItem.name }}</td>
          <td>{{ wasteItem.category }}</td>
          <td>{{ wasteItem.container }}</td>
        </tr>
      </tbody>
    </table>
</div>
```

Slika 4.2. *HTML predložak komponente wasteItems*

Kao što je već spomenuto, predložak se sastoji od dva dijela: obrasca za filtriranje i tablice. Primjetno je da se u HTML oznakama, osim uobičajenih atributa, nalaze i neki novi atributi, odnosno značajke karakteristične za Angular. U četvrtom retku to su [(ngModel)] i (input). Prva od navedenih je značajka poznata kao dvosmjerno vezanje podataka (engl. *two-way data binding*) koje, kako ime kaže, omogućuje protok podataka u dva smjera. U ovom slučaju protok je, s jedne strane, moguć od HTML oznake <input>, točnije tekstnog okvira za unos koji je označen sa searchTerm, do svojstva searchTerm koje je definirano u TypeScript datoteci komponente, a s

druge strane i u obrnutom smjeru, od svojstva natrag prema tekstnom okviru. Navedenom značajkom svaka promjena na jednom mjestu automatski se primjenjuje i na drugom mjestu, pa će se promjenom vrijednosti u tekstnom okviru u predlošku odmah promijeniti i vrijednost samog svojstva komponente, a isto tako će se s promjenom svojstva komponente promijeniti i vrijednost u tekstnom okviru [14]. Potrebno je naglasiti kako se direktiva ngModel u projektu može koristiti tek kada se u datoteku app.module.ts uveze FormsModule jer će u suprotnom biti javljena pogrješka. Također u četvrtom retku, neposredno iza dvosmjernog vezanja podataka nalazi se primjer vezanja događaja (engl. *event binding*) koje, kad se u DOM-u dogodi neki događaj, omogućuje pozivanje odgovarajuće metode iz komponente, odnosno TypeScript datoteke. Ovdje se za događaj input, odnosno unos nekog podatka, poziva metoda search() koja kao parametar prima searchTerm. U drugom, tabličnom dijelu predloška, s gledišta proučavanja Angulara značajni su redci 17 te 18, 19 i 20. Sedamnaesti redak sadrži strukturnu direktivu NgFor koja u ovom slučaju iterira kroz polje wasteItems stvarajući po jedan redak tablice za svaki element polja, označen varijablom wasteItem. U sljedeća tri retka koristi se interpolacija (engl. *interpolation binding*), koja nije ništa drugo nego jednosmjerno vezanje podataka, odnosno prijenos neke vrijednosti od komponente prema predlošku, a označava se vitičastim zagradama {{ }} unutar kojih se navodi svojstvo čija se vrijednost prikazuje, primjerice wasteItem.name na primjeru osamnaestog retka. Kao što je slučaj sa svojstvom searchTerm, polje wasteItems, čija je instanca wasteItem, također je definirano u TypeScript datoteci komponente, i to na način prikazan slikom 4.3.

```
1  import { Component, OnInit } from '@angular/core';
2  import { WasteItem } from 'src/app/interfaces/waste-item';
3
4  @Component({
5    selector: 'app-waste-items',
6    templateUrl: './waste-items.component.html',
7    styleUrls: ['./waste-items.component.css']
8  })
9  export class WasteItemsComponent implements OnInit {
10
11   searchTerm = '';
12   wasteItems: WasteItem[] = [];
13
14   constructor() { }
15
16   ngOnInit(): void {
17   }
18
19 }
```

Slika 4.3. *Definiranje svojstava u datoteci waste-item.component.ts*

Svojstvo wasteItems zapravo je polje tipa WasteItem, koji je unosom naredbe ng generate interface interfaces/wasteItem stvoren kao sučelje, a čiji je sadržaj prikazan slikom 4.4. Sučelja se u Angularu koriste kao sredstva opisivanja objekata i njima se, najjednostavnije rečeno, propisuju svojstva koja odgovarajući objekti moraju imati.

```
export interface WasteItem {
    name: string;
    category: string;
    container: string;
}
```

Slika 4.4. *Sučelje WasteItem*

U ovom slučaju sučeljem su definirana tri svojstva znakovnog tipa podataka (string), a koja označavaju ime otpadnog predmeta, kategoriju razvrstavanja i spremnik u koji je potrebno odložiti takav otpadni predmet. Radi jednostavnosti, ovdje se traženi podatci o otpadnim predmetima s navedenim svojstvima ne dohvaćaju s poslužitelja, kako je uobičajeno, nego iz posebno stvorene JSON datoteke, koja je dodana u direktorij assets, točnije njegov naknadno stvoreni poddirektorij data. Slikom 4.5. prikazano je dohvaćanje podataka iz datoteke waste-items.json.

```
export class WasteItemsComponent implements OnInit {

  searchTerm = '';

  wasteItems: WasteItem[] = [];
  allWasteItems: WasteItem[] = [];

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
    this.http.get<WasteItem[]>('./assets/data/waste-items.json')
      .subscribe((data: WasteItem[]) => {
        this.wasteItems = data;
        this.allWasteItems = this.wasteItems;
      });
  }

}
```

Slika 4.5. *Dohvaćanje podataka o otpadnim predmetima iz datoteke waste-items.json*

Prva promjena učinjena je na konstruktoru kojem je kao prvi parametar dodano privatno svojstvo http, a preko kojeg je zapravo ubrizgan servis HttpClient koji sadrži razne metode za izvođenje različitih HTTP zahtjeva. Navedeni oblikovni obrazac, kojim se jednoj klasi omogućuje korištenje metoda druge klase, čime prva klasa postaje ovisna o drugoj, naziva se ubrizgavanje ovisnosti (engl. *dependency injection*). Da bi ono bilo moguće prvo je potrebno uvesti servis HttpClient, a još prije toga je u datoteci app.module.ts potrebno uvesti HttpClientModule. Jedna od metoda za izvođenje HTTP zahtjeva, metoda get(), koja se koristi za dohvaćanje podataka, ovdje je pozvana unutar predefinirane metode ngOnInit(), koja se izvodi pri svakoj inicijalizaciji komponente [15]. Nadalje, nad metodom get() servisa HttpClient poziva se metoda subscribe() klase Observable kojom se, preko strjeličaste funkcije (engl. *arrow function*) s parametrom data, komponenta pretplaćuje na objekt klase Observable, a koji odašilje podatke s poslužitelja i o mogućim promjenama redovito obavještava komponentu, čime je prikaz podataka s poslužitelja uvijek ažuran.

Funkcionalnost filtriranja redaka tablice temeljem unosa, vidljiva na slici 4.2. u sedamnaestom retku nakon operatora |, implementirana je korištenjem cijevi (engl. *pipe*) koje se, najkraće rečeno, koriste za transformaciju podataka. Unosom naredbe ng generate pipe pipes/searchFilter stvorena je cijev searchFilter, čiji je sadržaj prikazan slikom 4.6.

```
1   import { Pipe, PipeTransform } from '@angular/core';
2
3   @Pipe({
4     name: 'searchFilter'
5   })
6   export class SearchFilterPipe implements PipeTransform {
7
8     transform(list: any[], filterText: string): any {
9       return list ? list.filter(item => item.name.search(new RegExp(filterText, 'i')) > -1) : [];
10    }
11
12  }
```

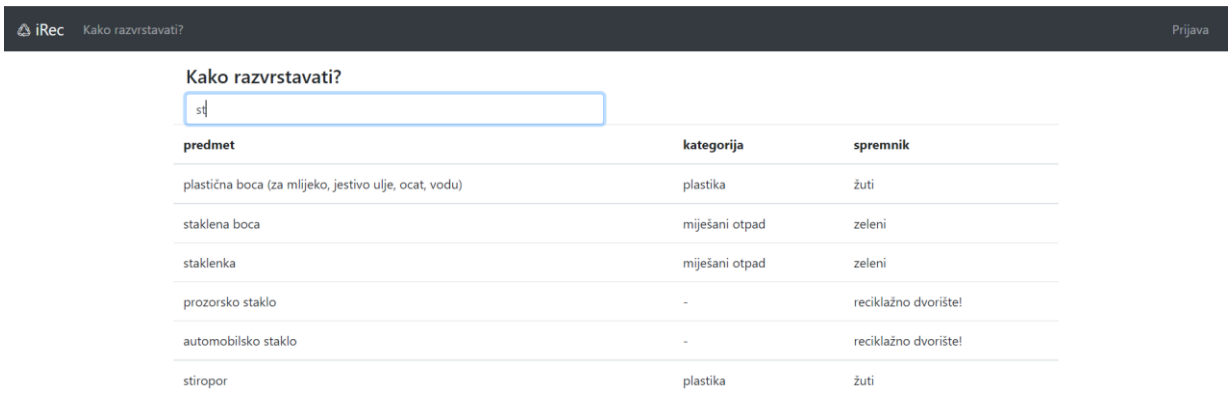
Slika 4.6. *SearchFilterPipe*

Novostvorena cijev implementira sučelje PipeTransform unutar kojeg je deklarirana metoda transform, a koja se koristi upravo za transformaciju podatka nad kojim se obavlja operacija. U ovom slučaju transformacija je obavljena filtriranjem liste temeljem unosa naziva. Slikom 4.7. dana je implementacija već spomenute metode search() u TypeScript datoteci komponente, koja se poziva događajem unosa u tekstnom okviru.

```
27  search(value: string): void {
28    this.wasteItems = this.allWasteItems.filter((val) =>
29      val.name.toLowerCase().includes(value)
30    );
31  }
```

Slika 4.7. *Metoda search() u datoteci waste-item.component.ts*

Konačan izgled komponente wasteItems prikazan je slikom 4.8.



Slika 4.8. *Izgled komponente wasteItems*

4.3. Autentikacija i autorizacija korisnika

Glavni dio aplikacije odnosi se na recikliranje različitih materijala, koje se obavlja tako što korisnici komunalnih usluga donose razne otpadne predmete iz svojih kućanstava u reciklažna dvorišta. Pri tome se uvijek vode određene evidencije, kao što je primjerice evidencija predmeta koji se recikliraju i evidencija korisnika koji donose predmete. Iz tog je razloga potrebno korisnicima omogućiti registraciju i prijavu, nakon kojih trebaju biti u mogućnosti pristupiti navedenom dijelu aplikacije, zbog čega je prvo stvoren projekt u Node.js-u, koji predstavlja poslužiteljsku stranu aplikacije. Nakon stvaranja običnog direktorija, koji je nazvan Irec-backend, unutar njega je unesena naredba npm init, kojom je on postao projektom Node.js-a. Zatim su naredbom npm install instalirani potrebni moduli. Express je razvojni okvir za izradu poslužiteljskih aplikacija koji omogućuje postavljanje poslužitelja i njihovo povezivanje s bazama podataka, što se postiže mnogobrojnim metodama za rukovanje zahtjevima te rutama. CORS je paket koji svojim zaglavljem omogućuje razmjenu zahtjeva i podataka između poslužiteljske i klijentske strane aplikacije, i to na siguran način, bez mogućnosti neovlaštenog pristupa ili krađe podataka koji se razmjenjuju. Modul body-parser pruža međusloj za parsiranje nadolazećih tijela zahtjeva prije samog rukovanja zahtjevima, a u programskom kodu se koristi kroz svojstvo req.body [16]. Sequelize je alat za objektno relacijsko mapiranje (ORM) koji omogućuje povezivanje aplikacije s bazom podataka i izvođenje operacija bez izravnog pisanja SQL upita [17]. Za rad s bazom podataka PostgreSQL važni su moduli pg i pg-hstore, koji serijalizira i deserijalizira podatke u formatu JSON u format hstore. Modul jsonwebtoken implementira JSON Web Token koji se u aplikaciji koristi za autentikaciju umjesto sjednica, a modul bcryptjs omogućuje kriptiranje. U okviru datotečne strukture projekta Node.js-a svi moduli nalaze se u poddirektoriju node_modules, a navedeni su i u datotekama package.json i package-lock.json. Za ulaznu točku poslužiteljske aplikacije umjesto predefinirane datoteke index.js odabrana je datoteka server.js u koju se uvoze svi moduli i rute potrebni za rad aplikacije, kako je prikazano slikom 4.9.

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();

app.use(
  cors({
    credentials: true,
    origin: ["http://localhost:8081"],
  })
);
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

const PORT = process.env.PORT || 8079;
app.listen(PORT, () => {
  console.log("Poslužitelj je pokrenut na portu " + PORT +"!");
});

app.get("/", (req, res) => {
  res.json({ message: "Irec-backend radi..." });
});

require("./routes/auth.routes")(app);
require("./routes/user.routes")(app);
```

Slika 4.9. *Datoteka server.js*

Kod uvoza modula CORS potrebno je primijetiti kako je postavljena adresa http://localhost:8081 kao ona s koje će se moći pristupiti poslužitelju, i to uz vjerodajnicu, koju će ovdje predstavljati već spomenuti JSON Web Token. To znači kako će se klijentska strana aplikacije morati izvoditi na portu 8081, odnosno da će njena adresa biti upravo http://localhost:8081. Što se tiče poslužiteljske strane, u nastavku datoteke server.js postavljen je port 8079 kao onaj na kojem se treba izvoditi poslužitelj, odnosno na kojem se osluškuju zahtjevi, te osnovna ruta GET, po čijem se unosu treba ispisati prigodna poruka. Sam poslužitelj pokreće se unosom naredbe node server.js, a uspješnost pokretanja moguće je odmah provjeriti unosom adrese http://localhost:8079, te se, ako je sve u redu, u pregledniku prikazuje prikladna poruka.

Kako bi se svi podatci mogli spremati u bazu podataka i kako bi se dalje njima moglo rukovati, potrebno je poslužitelj povezati s bazom, zbog čega je stvorena datoteka db.config.js, čiji je sadržaj prikazan slikom 4.10.

```
module.exports = {
    HOST: "localhost",
    USER: "postgres",
    PASSWORD: "postgres",
    DB: "irec_db",
    dialect: "postgres",
    pool: {
      max: 5,
      min: 0,
      acquire: 30000,
      idle: 10000
    }
  };
```

Slika 4.10. *Datoteka db.config.js*

Prvih pet parametara naznačuju kako je baza podataka lokalna, za pristup su potrebni korisničko ime i zaporka, ime joj je irec_db te je riječ o PostgreSQL-u. Parametar pool može se i ne mora navoditi, a određuje postavke povezivanja za Sequelize: najveći broj povezivanja, najmanji broj povezivanja, najveće vrijeme tijekom kojeg će biti pokušano spajanje prije javljanja pogrješke te najveće vrijeme za koje spoj može biti u stanju mirovanja prije prekida.

Kako bi se podatci mogli spremati u bazu podataka, potrebno je unutar nje stvoriti tablice, što je učinjeno pomoću modela. Za početak stvoren je poddirektorij models te unutar njega model User, prikazan slikom 4.11., i model Role, prikazan slikom 4.12.

```
module.exports = (sequelize, Sequelize) => {
    const User = sequelize.define("user", {
      name: {
        type: Sequelize.STRING
      },
      surname: {
        type: Sequelize.STRING
      },
      vat: {
        type: Sequelize.STRING
      },
      email: {
        type: Sequelize.STRING
      },
      password: {
        type: Sequelize.STRING
      }
    });
    return User;
  };
```

Slika 4.11. *Model User*

Model User predstavlja korisnike aplikacije koji će se registracijom spremati u tablicu users, a sadrži podatke poput imena, prezimena, OIB-a, adrese e-pošte i zaporke. Svi navedeni podatci su tekstualnog tipa, odnosno tipa string.

```
module.exports = (sequelize, Sequelize) => {
    const Role = sequelize.define("role", {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true
      },
      name: {
        type: Sequelize.STRING
      }
    });
    return Role;
  };
```

Slika 4.12. *Model Role*

Model Role predstavlja ovlasti, odnosno razine pristupa za registrirane korisnike, budući da je predviđeno postojanje dviju vrsta registriranih korisnika. Kako bi se, temeljem sada definiranih modela, omogućilo stvaranje tablica users i roles u samoj bazi podataka, također u poddirektoriju models stvorena je datoteka index.js, prikazana slikom 4.13.

```
const config = require("../config/db.config.js");
const Sequelize = require("sequelize");
const sequelize = new Sequelize(
  config.DB,
  config.USER,
  config.PASSWORD,
  {
    host: config.HOST,
    dialect: config.dialect,
    operatorsAliases: false,
    pool: {
      max: config.pool.max,
      min: config.pool.min,
      acquire: config.pool.acquire,
      idle: config.pool.idle
    }
  }
);
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;
db.user = require("../models/user.model.js")(sequelize, Sequelize);
db.role = require("../models/role.model.js")(sequelize, Sequelize);
db.role.belongsToMany(db.user, {
  through: "users_roles",
  foreignKey: "roleId",
  otherKey: "userId"
});
db.user.belongsToMany(db.role, {
  through: "users_roles",
  foreignKey: "userId",
  otherKey: "roleId"
});

db.ROLES = ["consumer", "recycler"];
module.exports = db;
```

Slika 4.13. *Datoteka index.js*

Prvo su uvezene postavke baze podataka, o kojima je bilo riječi oko slike 4.10., zatim modul Sequelize i konačno novostvoreni modeli User i Role. Također, definirana je veza N:M između dvaju modela, odnosno tablica users i roles, što podrazumijeva stvaranje treće tablice, imena users_roles, a koja sadrži samo primarne ključeve tablica users i roles. Na kraju su definirane dvije vrste registriranih korisnika, od kojih consumer predstavlja potrošače, odnosno korisnike usluga reciklažnog dvorišta, dok recycler predstavlja davatelja usluge recikliranja.

Nakon definiranja modela, potrebno je stvoriti odgovarajući kontroler, koji određuje način odgovora na zahtjeve. Kad je u pitanju autentikacija, radi se o funkciji registracije s jedne, te funkciji prijave korisnika s druge strane. U tu je svrhu stvoren poddirektorij controllers i unutar njega datoteka auth.controller.js za implementaciju navedenih dviju metoda. Na početku su

ponovno uvezene postavke baze podataka te modeli User i Role, a zatim i moduli jsonwebtoken i bcrypt koji su ovdje potrebni. Slikom 4.14. prikazana je metoda za registraciju u kontroleru autentikacije, a slikom 4.15. metoda za prijavu u istoj datoteci.

```
exports.signup = (req, res) => {
  User.create({
    name: req.body.name,
    surname: req.body.surname,
    vat: req.body.vat,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8)
  })
    .then(user => {
      if (req.body.roles) {
        Role.findAll({
          where: {
            name: {
              [Op.or]: req.body.roles
            }
          }
        }).then(roles => {
          user.setRoles(roles).then(() => {
            res.send({ message: "Novi korisnik uspješno je registriran!" });
          });
        });
      } else {
        user.setRoles([1]).then(() => {
          res.send({ message: "Novi korisnik uspješno je registriran!" });
        });
      }
    })
    .catch(err => {
      res.status(500).send({ message: err.message });
    });
};
```

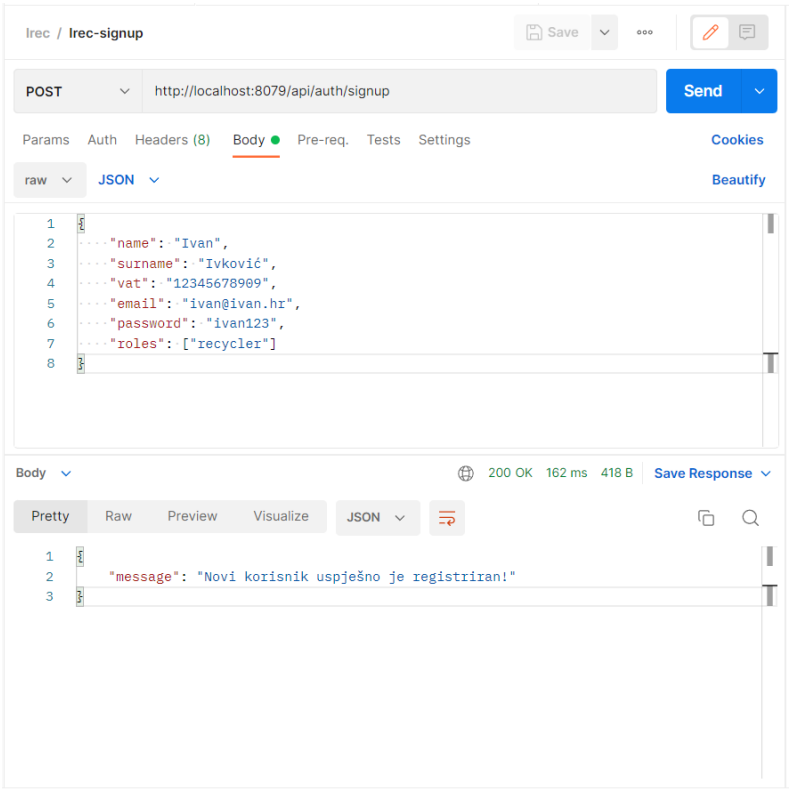
Slika 4.14. *Metoda za registraciju u datoteci auth.controller.js*

Kako je vidljivo, za funkciju registracije, odnosno stvaranje novog korisnika, korištena je metoda create() iz modula Sequalize, a koja je ovdje pozvana nad modelom User. Pri postavljanju razine pristupa, nad modelom Role pozvana je metoda findAll(), također iz modula Sequelize, a korištena je kako bi se provjerilo je li razina koja je navedena u tijelu zahtjeva uopće definirana. Ako nije navedena ni jedna razina, automatski se postavlja razina oznake 1, odnosno consumer. Za samo postavljanje razine pristupa novoregistriranog korisnika korištena je metoda setRoles(). Ako je registracija uspješna, dobiva se povratna poruka s takvom obavijesti.

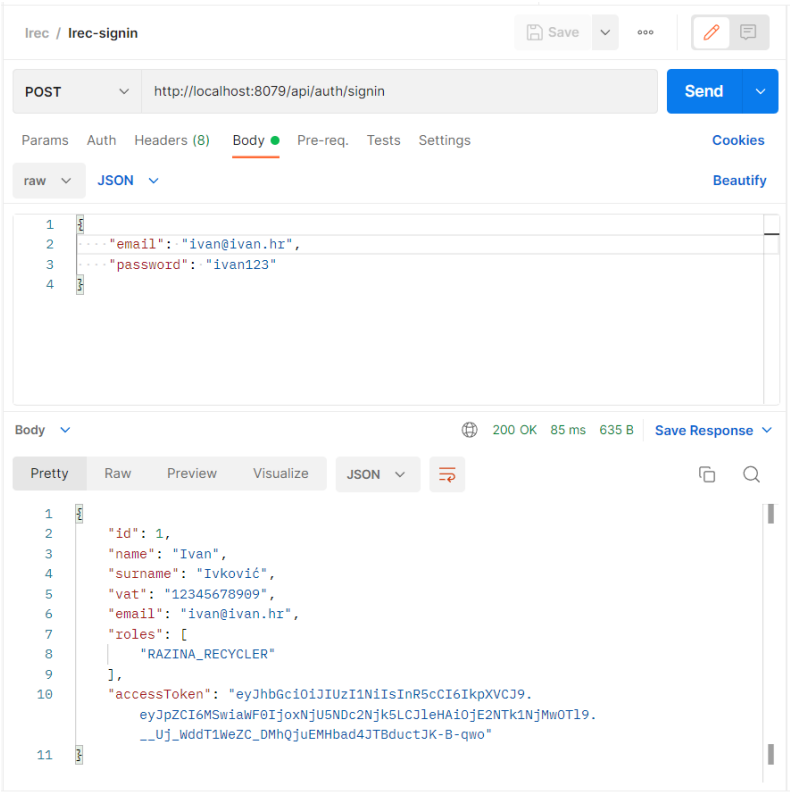
```
exports.signin = (req, res) => {
  User.findOne({
    where: {
      email: req.body.email
    }
  })
    .then(user => {
      if (!user) {
        return res.status(404).send({ message: "Korisnik nije pronađen!" });
      }
      var passwordIsValid = bcrypt.compareSync(
        req.body.password,
        user.password
      );
      if (!passwordIsValid) {
        return res.status(401).send({
          accessToken: null,
          message: "Netočna zaporka!"
        });
      }
      var token = jwt.sign({ id: user.id }, config.secret, {
        expiresIn: 86400
      });
      var authorities = [];
      user.getRoles().then(roles => {
        for (let i = 0; i < roles.length; i++) {
          authorities.push("RAZINA_" + roles[i].name.toUpperCase());
        }
        res.status(200).send({
          id: user.id,
          name: user.name,
          surname: user.surname,
          vat: user.vat,
          email: user.email,
          roles: authorities,
          accessToken: token
        });
      });
    })
    .catch(err => {
      res.status(500).send({ message: err.message });
    });
};
```

Slika 4.15. *Metoda za prijavu u datoteci auth.controller.js*

Kod prijave korisnika provjerava se postoji li takav korisnik u bazi podataka, zbog čega je nad modelom User pozvana metoda findOne(), a kriterij za provjeru je adresa e-pošte koja se koristi za prijavu. Ako korisnik s navedenom adresom e-poštom nije pronađen u bazi podataka, ispisuje se odgovarajuća poruka. Dalje se provjerava ispravnost unesene zaporke te se u slučaju neispravnog unosa također ispisuje odgovor s takvom obavijesti. Ako je sve u redu, generira se jedinstveni token s trajanjem od 86.400 sekundi, točnije 24 sata. Na kraju se dohvaća korisnikova razina pristupa, te se zajedno s tokenom i korisnikovim podatcima dohvaćenima iz baze, vraća kao odgovor. Kako bi se provjerila ispravnost funkcionalnosti registracije i prijave korisnika, u Postmanu su stvoreni zahtjevi čiji su rezultati prikazani slikama 4.16. i 4.17.



Slika 4.16. *Provjera ispravnosti funkcionalnosti registracije*



Slika 4.17. *Provjera ispravnosti funkcionalnosti prijave*

Za izvršenje željenih zahtjeva prethodno je potrebno definirati odgovarajuće rute. Na slikama 4.16. i 4.17. pri vrhu je vidljivo da su one /api/auth/signup (za registraciju) i /api/auth/signin (za prijavu), a definirane su u datoteci auth.routes.js koja se nalazi u poddirektoriju routes i čiji je sadržaj prikazan slikom 4.18.

```
1  const { verifySignUp } = require("../middleware");
2  const controller = require("../controllers/auth.controller");
3
4  module.exports = function(app) {
5    app.use(function(req, res, next) {
6      res.header(
7        "Access-Control-Allow-Headers",
8        "x-access-token, Origin, Content-Type, Accept"
9      );
10     next();
11   });
12
13   app.post(
14     "/api/auth/signup",
15     [
16       verifySignUp.checkDuplicateVATOrEmail,
17       verifySignUp.checkRolesExisted
18     ],
19     controller.signup
20   );
21   app.post("/api/auth/signin", controller.signin);
22   app.post("/api/auth/signout", controller.signout);
23 };
```

Slika 4.18. *Datoteka auth.routes.js*

Definirane su rute za registraciju, prijavu i odjavu te su povezane s odgovarajućim metodama u kontroleru koje su već spomenute. Na taj način se pozivanjem određene rute zapravo poziva i izvršava odgovarajuća metoda. Kod rute za registraciju vidljivo je postojanje dviju provjera, implementiranih u datoteci verifySignUp.js koja predstavlja svojevrsni međusloj (engl. *middleware*), zbog čega je stvorena u novom poddirektoriju middleware. Prva metoda pri registraciji provjerava postoji li već u bazi podataka korisnik s navedenim OIB-om ili adresom e-pošte, dok druga metoda provjerava postoji li razina pristupa koja je unesena. Metoda za provjeru OIB-a i adrese e-pošte prikazana je slikom 4.19, a metoda za provjeru postojanja razine pristupa slikom 4.20.

```
checkDuplicateVATOrEmail = (req, res, next) => {
  User.findOne({
    where: {
      vat: req.body.vat
    }
  }).then(user => {
    if (user) {
      res.status(400).send({
        message: "Pogrješka! Već postoji račun s unesenim OIB-om!"
      });
      return;
    }
    User.findOne({
      where: {
        email: req.body.email
      }
    }).then(user => {
      if (user) {
        res.status(400).send({
          message: "Pogrješka! Već postoji račun s unesenom e-poštom!"
        });
        return;
      }
      next();
    });
  });
};
```

Slika 4.19. *Provjera OIB-a i adrese e-pošte*

Ponovno je korištena metoda findOne() koja je pozvana nad modelom User. Ako se provjerom utvrdi da su OIB ili adresa e-pošte već zapisani u bazi podataka, ispisuje se odgovarajuća poruka, a završetak registracije s takvim podatcima nije moguć.

```
checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: "Pogrješka! Razina pristupa " + req.body.roles[i] + " ne postoji!"
        });
        return;
      }
    }
  }
  next();
};
```

Slika 4.20. *Provjera postojanja razine pristupa*

Kako je vidljivo, kod provjere postojanja razine pristupa slučaj je obrnut. Ako se provjerom utvrdi da predana razina pristupa ne postoji u bazi podataka, tada se ispisuje odgovarajuća poruka o pogrješci, a završetak registracije nije moguć sve dok se ne navede jedna od definiranih razina pristupa.

Kako bi se u nastavku rada omogućio uvid u recikliranja svakog korisnika, potrebne su metode za skupno i pojedinačno dohvaćanje svih korisnika. Zbog toga je stvoren novi kontroler, nazvan user.controller.js i prikazan slikom 4.21.

```
const db = require("../models");
const User = db.user;

exports.findAll = (req, res) => {
    User.findAll()
      .then(data => {
        res.send(data);
      })
      .catch(err => {
        res.status(500).send({
          message:
            err.message || "Dogodila se pogrješka pri dohvaćanju korisnika!"
        });
      });
  };

exports.findOne = (req, res) => {
    const id = req.params.id;
    User.findByPk(id)
      .then(data => {
        if (data) {
          res.send(data);
        } else {
          res.status(404).send({
            message: "Korisnik oznake " + id + " nije pronađen!"
          });
        }
      })
      .catch(err => {
        res.status(500).send({
          message: "Dogodila se pogrješka pri dohvaćanju korisnika oznake " + id + "!"
        });
      });
  };
```

Slika 4.21. *Datoteka user.controller.js*

Prva metoda omogućuje dohvaćanje svih registriranih korisnika, pa se nad modelom User poziva metoda findAll(), dok druga dohvaća korisnika određenog oznakom, zbog čega nad modelom User poziva metoda findByPk() kojoj se kao parametar predaje oznaka željenog korisnika. Kako bi se obje novodefinirane metode mogle primijeniti, potrebno je stvoriti rute i povezati ih, što je učinjeno u datoteci user.routes.js, prikazanoj slikom 4.22.

```
const { authJwt } = require("../middleware");

module.exports = app => {
  const users = require("../controllers/user.controller.js");
  var router = require("express").Router();

  router.get("/", [authJwt.verifyToken, authJwt.isRecycler], users.findAll);
  router.get("/:id", [authJwt.verifyToken, authJwt.isRecycler], users.findOne);
  app.use('/api/users', router);
};
```

Slika 4.22. *Datoteka user.routes.js*

I ovdje se kod dohvaćanja ruta obavljaju određene provjere: najprije provjera ispravnosti tokena, a zatim i provjera pripadnosti razini pristupa recycler. U tu je svrhu unutar poddirektorija middleware stvorena datoteka authJwt.js u kojoj su implementirane sljedeće metode: verifyToken(), koja služi za provjeru tokena, te isRecycler() kojom se provjerava ima li korisnik razinu pristupa recycler. Slikom 4.23. prikazana je metoda verifyToken(), a slikom 4.24. metoda isRecycler().

```
6   verifyToken = (req, res, next) => {
7     let token = req.headers["x-access-token"];
8     if (!token) {
9       return res.status(403).send({
10        message: "Nema tokena!"
11      });
12    }
13    jwt.verify(token, config.secret, (err, decoded) => {
14      if (err) {
15        return res.status(401).send({
16          message: "Pristup nije dopušten!"
17        });
18      }
19      req.userId = decoded.id;
20      next();
21    });
22  };
```

Slika 4.23. *Metoda verifyToken()*

Prvo se dohvaća token iz HTTP zaglavlja x-access-token. Ako nije dohvaćen, vraća se pogrješka s takvom obavijesti. Dalje se token provjerava pomoću metode verify() iz modula jsonwebtoken koja radi tako što uspoređuje dohvaćeni token s tajnim ključem definiranim u datoteci auth.config.js. Ako se provjerom utvrdi kako nema podudaranja, neovlašteni pristup se onemogućuje uz odgovarajuću poruku.

```
24  isRecycler = (req, res, next) => {
25    User.findByPk(req.userId).then(user => {
26      user.getRoles().then(roles => {
27        for (let i = 0; i < roles.length; i++) {
28          if (roles[i].name === "recycler") {
29            next();
30            return;
31          }
32        }
33        res.status(403).send({
34          message: "Potrebna je razina pristupa RECYCLER!"
35        });
36        return;
37      });
38    });
39  };
```

Slika 4.24. *Metoda isRecycler()*

Nad modelom User poziva se metoda findByPk() kojom se temeljem predane oznake dohvaća odgovarajući korisnik. Zatim se poziva metoda getRoles() kojom se dohvaća razina pristupa za tog korisnika, a koja se onda uspoređuje s razinom consumer. Ako je usporedbom utvrđeno podudaranje, izvođenje uspješno ide dalje, a u suprotnom se javlja pogrješka.

Kako bi se sve implementirane funkcionalnosti mogle vidljivo koristiti na klijentskoj strani aplikacije, potrebno je u projektu u Angularu prvo stvoriti servis koji će omogućiti njihovu primjenu. Unosom naredbe ng generate service services/authService stvoren je AuthService u kojem su definirane metode za registraciju, prijavu i odjavu, a koje zapravo samo pozivaju odgovarajuće metode s poslužitelja. Slikom 4.25. prikazan je dio datoteke auth.service.ts, odnosno metoda za registraciju.

```
1  import { Injectable } from '@angular/core';
2  import { HttpClient, HttpHeaders } from '@angular/common/http';
3  import { Observable } from 'rxjs';
4
5  const AUTH_API = 'http://localhost:8079/api/auth/';
6  const httpOptions = {
7    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
8  };
9
10 @Injectable({
11   providedIn: 'root'
12 })
13 export class AuthService {
14
15   constructor(private http: HttpClient) {}
16
17   register(name: string, surname: string, vat: string, email: string, password: string): Observable<any> {
18     return this.http.post(
19       AUTH_API + 'signup',
20       {
21         name, surname, vat, email, password,
22       },
23       httpOptions
24     );
25   }
```

Slika 4.25. *Metoda za registraciju u datoteci auth.service.ts*

Osim ovisnosti Injectable, koja je automatski uvezena samim stvaranjem servisa, uvezene su i ovisnosti HttpClient, HttpHeaders te Observable, koje su potrebne za komunikaciju servisa na klijentskoj strani aplikacije sa servisom na poslužiteljskoj strani aplikacije. Zatim je definirana varijabla AUTH_API koja sadrži URL poslužiteljske strane s koje se trebaju dohvatiti metode. Za ostvarivanje same komunikacije s poslužiteljem preko konstruktora je ubrizgana ovisnost HttpClient, čija je metoda post() potom pozvana unutar metode register(), a koja kao parametre prima ime, prezime, OIB, adresu e-pošte i zaporku korisnika za kojeg se obavlja registracija. Parametri metode post() su URL odgovarajuće metode na poslužiteljskoj strani aplikacije, u ovom slučaju metode signup(), potom podatci koji se šalju i na kraju postavke HTTP komunikacije kojima je naznačeno slanje podataka u formatu JSON.

Kako bi se AuthService mogao koristiti u novostvorenoj komponenti za registraciju, RegisterComponent, prvo ga je potrebno uvesti u njenu TypeScript datoteku, čiji je glavni dio prikazan slikom 4.26.

```
10  export class RegisterComponent implements OnInit {
11    form: any = {
12      name: null, surname: null, vat: null, email: null, password: null
13    };
14    isSuccessful = false;
15    isSignUpFailed = false;
16    errorMessage = '';
17
18    constructor(private authService: AuthService) { }
19
20    ngOnInit(): void {
21    }
22
23    onSubmit(): void {
24      const { name, surname, vat, email, password } = this.form;
25      this.authService.register(name, surname, vat, email, password).subscribe({
26        next: data => {
27          console.log(data);
28          this.isSuccessful = true;
29          this.isSignUpFailed = false;
30        },
31        error: err => {
32          this.errorMessage = err.error.message;
33          this.isSignUpFailed = true;
34        }
35      });
36    }
37  }
```

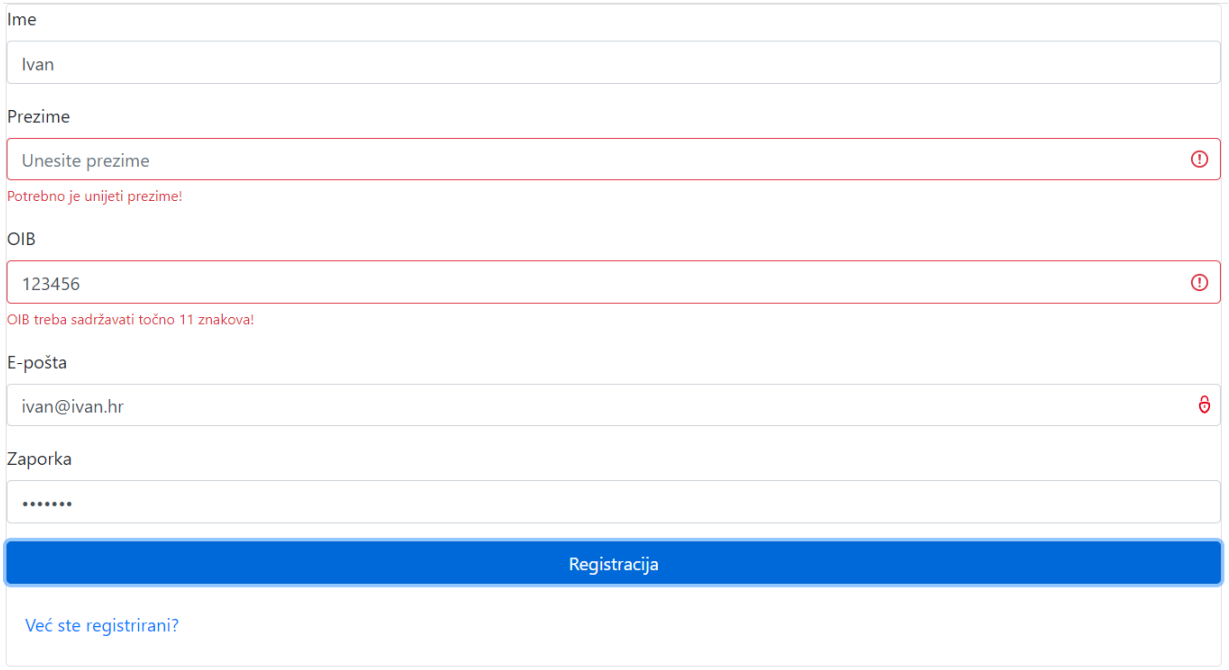
Slika 4.26. *Datoteka register.component.ts*

Nakon početnog definiranja potrebnih varijabli, preko konstruktora je ubrizgan AuthService te je implementirana metoda onSubmit() unutar koje se poziva metoda register() iz AuthService-a. Nad metodom register() poziva se metoda subscribe(), pri čemu se varijabli next, koja je svojstvo sučelja Observer, dodjeljuje vrijednost varijable data, odnosno svih podataka. U slučaju pogrješke ispisuje se obavijest. Varijable isSuccessful i isSignUpFailed definirane su radi ispitivanja određenih uvjeta, prije svega vezanih za promjenu u prikazu komponente, točnije HTML predloška, čiji je jedan dio prikazan slikom 4.27.

```
1  <form name="form" *ngIf="!isSuccessful" #f="ngForm" (ngSubmit)="f.form.valid && onSubmit()" novalidate>
2    <div class="form-group">
3      <label for="vat">OIB</label>
4      <input type="text" name="vat" placeholder="Unesite OIB" [(ngModel)]="form.vat" required
5              minlength="11" maxlength="11" #vat="ngModel" [ngClass]="{ 'is-invalid': f.submitted && vat.errors }" />
6        <div class="invalid-feedback" *ngIf="vat.errors && f.submitted">
7          <div *ngIf="vat.errors['required']">Potrebno je unijeti OIB!</div>
8          <div *ngIf="vat.errors['minlength']">OIB treba sadržavati točno 11 znakova!</div>
9          <div *ngIf="vat.errors['maxlength']">OIB treba sadržavati točno 11 znakova!</div>
10       </div>
11   </div>
12   <div class="form-group">
13     <button class="btn btn-primary btn-block">Registracija</button>
14   </div>
15   <div class="form-group">
16     <a class="nav-link" routerLink="../login">Već ste registrirani?</a>
17   </div>
18   <div class="alert alert-warning" *ngIf="f.submitted && isSignUpFailed">
19     Registracija nije uspjela: <br />{{ errorMessage }}
20   </div>
21 </form>
22
23 <div class="alert alert-success" *ngIf="isSuccessful">
24   Registracija je uspješna!
25 </div>
```

Slika 4.27. *Isječak iz datoteke register.component.html*

Promjena u prikazu omogućena je korištenjem strukturne direktive NgIf. U retku 1 vidljiv je uvjet !isSuccesful, što znači da se blok koji predstavlja obrazac za registraciju prikazuje kada varijabla isSuccessful ima vrijednost false. S uspjehom registracije vrijednost varijable mijenja se u true i obrazac tada nestaje, a umjesto njega se pojavljuje okvir s obavijesti o uspješnoj registraciji, budući da je u retku 23 postavljen uvjet isSuccessful. Strukturna direktiva NgIf korištena je također i unutar samog obrasca, kod validacije unosa, točnije pojavljivanja obavijesti o neispravnosti unosa. Sama validacija ostvarena je pomoću koncepta obrasca temeljenog na predlošku (engl. *template-driven form*) i direktiva NgForm te NgModel. Direktiva NgForm preko lokalne varijable #f prati sve sastavnice obrasca te na vezani događaj NgSubmit, kojem je pridružena već spomenuta metoda onSubmit(), obavještava o neispravnosti unosa. Direktiva NgModel dodaje se svakom elementu za unos podataka te se preko nje prikupljaju sve „željene“ neispravnosti unosa za isti taj element, a za koje se kriteriji mogu postaviti navođenjem atributa poput required, minlength, maxlength itd. Slikom 4.28. prikazan je konačan izgled komponente za registraciju s vidljivom primjenom validacije unosa.



Slika 4.28. *Izgled komponente register*

Na identičan je način realizirana i komponenta za prijavu korisnika, uz razliku što je nakon prijave potrebno zadržati podatke o prijavljenom korisniku. Zato je potrebno stvoriti novi servis i potom ga ubrizgati u komponentu za prijavu. Slikom 4.29. prikazan je glavni dio datoteke storage.service.ts.

```
9   export class StorageService {
10
11    constructor() {}
12
13    clean(): void {
14      window.sessionStorage.clear();
15    }
16    public saveToken(token: string): void {
17      window.sessionStorage.removeItem(TOKEN_KEY);
18      window.sessionStorage.setItem(TOKEN_KEY, token);
19    }
20    public getToken(): string | null {
21      return window.sessionStorage.getItem(TOKEN_KEY);
22    }
23    public saveUser(user: any): void {
24      window.sessionStorage.removeItem(USER_KEY);
25      window.sessionStorage.setItem(USER_KEY, JSON.stringify(user));
26    }
27    public getUser(): any {
28      const user = window.sessionStorage.getItem(USER_KEY);
29      if (user) {
30        return JSON.parse(user);
31      }
32      return {};
33    }
34    public isLoggedIn(): boolean {
35      const user = window.sessionStorage.getItem(USER_KEY);
36      if (user) {
37        return true;
38      }
39      return false;
40    }
41  }
```

Slika 4.29. *Datoteka storage.service.ts*

StorageService je, najjednostavnije rečeno, servis za rukovanje podatcima o trenutačno prijavljenom korisniku u pregledniku. Vidljive su metode za čišćenje sjednice, za spremanje i dohvaćanje tokena, za spremanje i dohvaćanje korisnika te za provjeru prijavljenosti korisnika. Navedene metode korištene su u komponenti za prijavu, čija je TypeScript datoteka, odnosno njen glavni dio, prikazana slikom 4.30.

```
10  export class LoginComponent implements OnInit {
11    form: any = {
12      email: null, password: null
13    };
14    isLoggedIn = false;
15    isLoginFailed = false;
16    errorMessage = '';
17    roles: string[] = [];
18    constructor(private authService: AuthService, private storageService: StorageService) { }
19
20    ngOnInit(): void {
21      if (this.storageService.getToken()) {
22        this.isLoggedIn = true;
23        this.roles = this.storageService.getUser().roles;
24      }
25    }
26    onSubmit(): void {
27      const { email, password } = this.form;
28      this.authService.login(email, password).subscribe({
29        next: data => {
30          this.storageService.saveToken(data.accessToken);
31          this.storageService.saveUser(data);
32          this.isLoginFailed = false;
33          this.isLoggedIn = true;
34          this.roles = this.storageService.getUser().roles;
35          this.reloadPage();
36        },
37        error: err => {
38          this.errorMessage = err.error.message;
39          this.isLoginFailed = true;
40        }
41      });
42    }
43    reloadPage(): void {
44      window.location.reload();
45    }
46  }
```

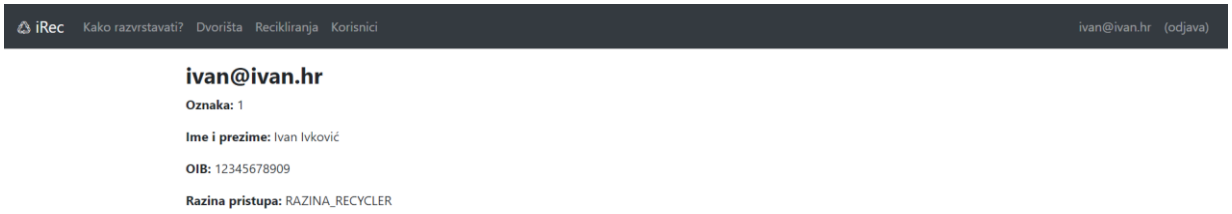
Slika 4.30. *Datoteka login.component.ts*

Preko konstruktora su ubrizgani AuthService i StorageService, čime je omogućeno pozivanje njihovih metoda. Pri inicijalizaciji komponente, točnije unutar metode ngOnInit(), pomoću metode za dohvaćanje tokena provjerava se je li korisnik prijavljen i temeljem toga se dohvaća razina pristupa. Unutar metode onSubmit() koja se poziva pri predaji korisničkih podataka za prijavu, metodom saveToken() sprema se token, a metodom saveUser() postavljaju se podatci o prijavljenom korisniku. Nakon ovoga isti su dostupni cijelo vrijeme dok je korisnik prijavljen, a može ih se vidjeti na prikazu korisničkog profila, radi izrade kojega je stvorena komponenta profile, a njena TypeScript datoteka dana je slikom 4.31.

```typescript
import { Component, OnInit } from '@angular/core';
import { StorageService } from 'src/app/services/storage.service';
import { User } from 'src/app/interfaces/user';

@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.css']
})
export class ProfileComponent implements OnInit {
  currentUser?: User;

  constructor(private storageService: StorageService) { }

  ngOnInit(): void {
    this.currentUser = this.storageService.getUser();
  }
}
```

Slika 4.31. *Datoteka profile.component.ts*

Unutar metode ngOnInit() pozvana je metoda getUser() kojom je dohvaćen trenutačno prijavljeni korisnik te su podatci o njemu spremljeni u varijablu currentUser koja je tipa User. Slikom 4.32. prikazan je konačni izgled komponente profile.



Slika 4.32. *Izgled komponente profile*

4.4. Rad s reciklažnim dvorištima

Dio aplikacije predviđen za rad s reciklažnim dvorištima namijenjen je isključivo korisniku s razinom pristupa recycler, odnosno pružatelju usluga recikliranja koji kao takav jedini treba imati pristup korisničkom sučelju za dodavanje novih te uređivanje i brisanje postojećih reciklažnih dvorišta, koja su određena oznakom, nazivom, adresom, brojem telefona i web-mjestom. U tu je svrhu na poslužiteljskoj strani prvo stvoren model Yard, prikazan slikom 4.33.

```
1  module.exports = (sequelize, Sequelize) => {
2    const Yard = sequelize.define("yard", {
3      name: {
4        type: Sequelize.STRING
5      },
6      address: {
7        type: Sequelize.STRING
8      },
9      phone: {
10       type: Sequelize.STRING
11     },
12     website: {
13       type: Sequelize.STRING
14     }
15   });
16   return Yard;
17 };
```

Slika 4.33. *Model Yard*

Metode za rad s reciklažnim dvorišta definiraju se u pripadajućem kontroleru, a u njima su također zapravo pozivane metode koje podržava modul Sequelize, poput create(), findAll(), findByPk(), update() i destroy(). Slikom 4.34. prikazana je metoda za dodavanje reciklažnog dvorišta.

```
5  exports.create = (req, res) => {
6    if (!req.body.name || !req.body.address || !req.body.phone || !req.body.website) {
7      res.status(400).send({
8        message: "Potrebno je unijeti sve podatke!"
9      });
10     return;
11   }
12   const yard = {
13     name: req.body.name,
14     address: req.body.address,
15     phone: req.body.phone,
16     website: req.body.website
17   };
18   Yard.create(yard)
19     .then(data => {
20       res.send(data);
21     })
22     .catch(err => {
23       res.status(500).send({
24         message:
25           err.message || "Dogodila se pogrješka pri dodavanju novog dvorišta!"
26       });
27     });
28 };
```

Slika 4.34. *Metoda za dodavanje reciklažnog dvorišta u datoteci yard.controller.js*

U prvom dijelu metode provodi se validacija unosa kojom je onemogućeno slanje zahtjeva dok nisu uneseni svi traženi podatci, odnosno ime, adresa, broj telefona i web-mjesto dvorišta. U drugom dijelu se metodom create() stvara samo dvorište.

Na slici 4.35. nalaze se dvije metode za dohvaćanje reciklažnih dvorišta.

```
exports.findAll = (req, res) => {
    Yard.findAll()
      .then(data => {
        res.send(data);
      })
      .catch(err => {
        res.status(500).send({
          message:
            err.message || "Dogodila se pogrješka pri dohvaćanju dvorišta!"
        });
      });
  };

exports.findOne = (req, res) => {
    const id = req.params.id;
    Yard.findByPk(id)
      .then(data => {
        if (data) {
          res.send(data);
        } else {
          res.status(404).send({
            message: "Dvorište oznake " + id + " nije pronađeno!"
          });
        }
      })
      .catch(err => {
        res.status(500).send({
          message: "Dogodila se pogrješka pri dohvaćanju dvorišta oznake " + id + "!"
        });
      });
  };
```

Slika 4.35. *Metode za dohvaćanje reciklažnih dvorišta u datoteci yard.controller.js*

Prva metoda dohvaća sva reciklažna dvorišta, i to pozivanjem metode findAll(). Druga metoda dohvaća jedno reciklažno dvorište određeno oznakom, pa se pritom poziva metoda findByPk.

Slikom 4.36. prikazana je metoda za uređivanje podataka o reciklažnom dvorištu.

```
exports.update = (req, res) => {
    const id = req.params.id;
    Yard.update(req.body, {
      where: { id: id }
    })
      .then(num => {
        if (num == 1) {
          res.send({
            message: "Podatci o dvorištu uspješno su promijenjeni!"
          });
        } else {
          res.send({
            message: "Podatke o dvorištu oznake " + id + " nije moguće promijeniti!"
          });
        }
      })
      .catch(err => {
        res.status(500).send({
          message: "Pogrješka pri izmjeni podataka o dvorištu oznake " + id + "!"
        });
      });
  };
```

Slika 4.36. *Metoda za uređivanje podataka o reciklažnom dvorištu u datoteci yard.controller.js*

Prije samog ažuriranja podataka dohvaća se oznaka predana u zahtjevu te se, ako je u bazi pronađeno dvorište s navedenom oznakom, ažuriranje uspješno obavlja, a u suprotnom se dojavljuje pogrješka.

Slikom 4.37. prikazane su dvije metode za brisanje reciklažnih dvorišta.

```
88  exports.delete = (req, res) => {
89      const id = req.params.id;
90      Yard.destroy({
91        where: { id: id }
92      })
93        .then(num => {
94          if (num == 1) {
95            res.send({
96              message: "Dvorište je uspješno obrisano!"
97            });
98          } else {
99            res.send({
100             message: "Dvorište oznake " + id + " nije moguće obrisati jer ono nije pronađeno!"
101           });
102         }
103       })
104       .catch(err => {
105         res.status(500).send({
106           message: "Nije moguće obrisati dvorište oznake " + id + "!"
107         });
108       });
109   };
110
111 exports.deleteAll = (req, res) => {
112     Yard.destroy({
113       where: {},
114       truncate: false
115     })
116       .then(nums => {
117         res.send({ message: "Sva dvorišta (njih " + nums + ") uspješno je obrisano!"});
118       })
119       .catch(err => {
120         res.status(500).send({
121           message:
122             err.message || "Dogodila se pogrješka pri brisanju svih dvorišta!"
123         });
124       });
125   };
```

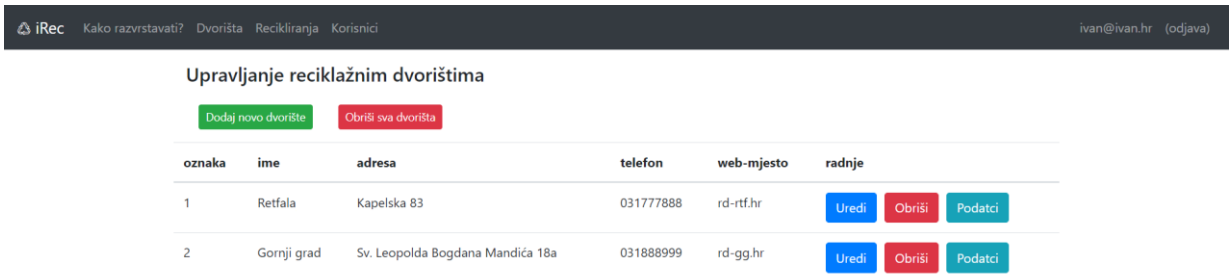
Slika 4.37. *Metode za brisanje reciklažnih dvorišta u datoteci yard.controller.js*

Prva metoda briše jedno reciklažno dvorište određeno oznakom, dok druga metoda briše sva reciklažna dvorišta. Po završetku implementiranja svih metoda na poslužiteljskoj strani, iste je potrebno učiniti dostupnima za primjenu i na klijentskoj strani, zbog čega je ondje prvo stvoreno sučelje Yard, a zatim i servis YardService, prikazan slikom 4.38.

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Yard } from '../interfaces/yard';
const baseUrl = 'http://localhost:8079/api/yards';

@Injectable({
  providedIn: 'root'
})
export class YardService {
  constructor(private http: HttpClient) { }
  getAll(): Observable<Yard[]> {
    return this.http.get<Yard[]>(baseUrl);
  }
  get(id: any): Observable<Yard> {
    return this.http.get(`${baseUrl}/${id}`);
  }
  create(data: any): Observable<any> {
    return this.http.post(baseUrl, data);
  }
  update(id: any, data: any): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }
  delete(id: any): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }
  deleteAll(): Observable<any> {
    return this.http.delete(baseUrl);
  }
}
```

Slika 4.38. *Datoteka yard.service.ts*

Glavna komponenta za rad s reciklažnim dvorištima je yardsList, čiji je izgled prikazan slikom 4.39.



Slika 4.39. *Izgled komponente yardsList*

Radi se o tabličnom prikazu reciklažnih dvorišta, pri čemu je moguće dodavanje novog i brisanje svih reciklažnih dvorišta. Također, za svaki redak, koji predstavlja jedno reciklažno dvorište, moguće je uređivanje podataka o dvorištu, pojedinačno brisanje te prikaz podataka o dvorištu. Brisanje svih dvorišta odjednom i pojedinačno brisanje rezultiraju samo osvježenim tabličnim prikazom, a potrebne metode definirane su u TypeScript datoteci komponente, čiji je sadržaj prikazan slikom 4.40.

```
20  retrieveYards(): void {
21    this.yardService.getAll()
22      .subscribe({
23        next: (data) => {
24          this.yards = data;
25          console.log(data);
26        },
27        error: (e) => console.error(e)
28      });
29  }
30
31  removeAllYards(): void {
32    this.yardService.deleteAll()
33      .subscribe({
34        next: (res) => {
35          console.log(res);
36          this.retrieveYards();
37        },
38        error: (e) => console.error(e)
39      });
40  }
41
42  deleteYard(id: any) {
43    this.yardService.delete(id)
44      .subscribe({
45      next: (res) => {
46        console.log(res);
47        this.retrieveYards();
48      },
49      error: (e) => console.error(e)
50    });
51  }
```

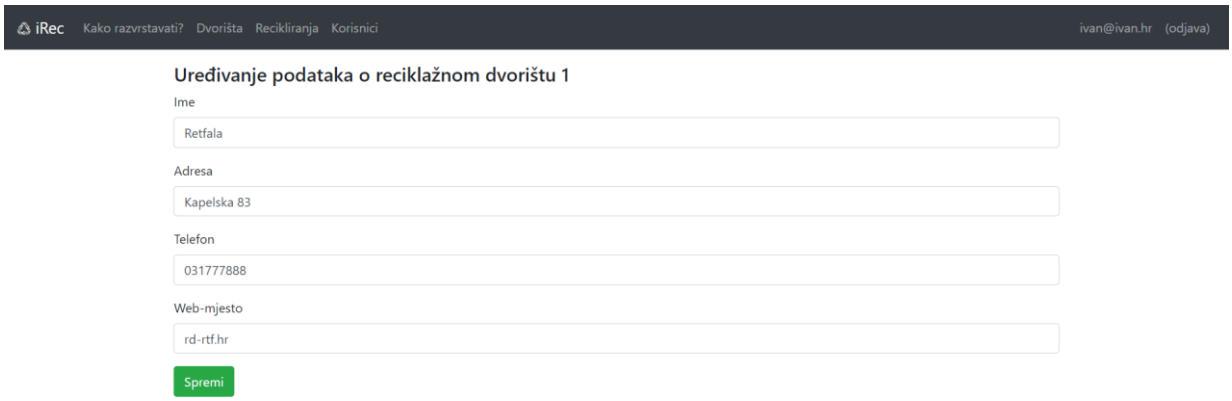
Slika 4.40. *Metode za dohvaćanje svih dvorišta te za skupno i pojedinačno brisanje u datoteci yards-list.component.ts*

Sve tri metode zapravo pozivaju odgovarajuće metode iz servisa, nad kojima se potom poziva metoda subscribe(). Metoda za dohvaćanje se, osim pri inicijalizaciji komponente, poziva i u dvjema metodama za brisanje kako bi se odmah ažurirao tablični prikaz reciklažnih dvorišta. Funkcionalnosti pojedinačnog uređivanja i pregledavanja postojećih reciklažnih dvorišta realizirane su korištenjem drugih komponenti, na koje je potrebno napraviti preusmjeravanje, kako je prikazano slikom 4.41.

```
53  showUpdateYard(id: any){
54    this.router.navigate(['updateYard', id]);
55  }
56
57  showYardDetails(id: any){
58    this.router.navigate(['yardDetails', id]);
59  }
```

Slika 4.41. *Metode za preusmjeravanje na druge komponente u datoteci yards-list.component.ts*

Kako bi se navedeno postiglo, prethodno je potrebno u komponentu uvesti modul Router i ubrizgati ga preko konstruktora. Slikom 4.42. prikazana je komponenta za uređivanje, a slikom 4.43. komponenta za prikaz podataka o pojedinom reciklažnom dvorištu.



Slika 4.42. *Izgled komponente updateYard*

Podatci o reciklažnom dvorištu dohvaćaju se preko oznake predane zahtjevom i prikazuju u poljima obrasca.

Slika 4.43. *Izgled komponente yardDetails*

Isto kao i kod komponente za uređivanje, i ovdje se podatci dohvaćaju pomoću oznake predane u zahtjevu. Dohvaćanje podataka obavlja se u TypeScript datoteci komponente, kako je prikazano slikom 4.44.

```
11  export class YardDetailsComponent implements OnInit {
12    currentYard: Yard = {};
13
14    constructor(private yardService: YardService, private route: ActivatedRoute) { }
15
16    ngOnInit(): void {
17      this.getYard(this.route.snapshot.params["id"]);
18    }
19
20    getYard(id: string): void {
21      this.yardService.get(id)
22        .subscribe({
23          next: (data) => {
24            this.currentYard = data;
25            console.log(data);
26          },
27          error: (e) => console.error(e)
28        });
29    }
30  }
```

Slika 4.44. *Datoteka yard-details.component.ts*

Dohvaćeno dvorište sprema se u varijablu currentYard koja se potom koristi u HTML predlošku. Kako bi se dvorište predano zahtjevom moglo dohvatiti uvezen je i ubrizgan modul ActivatedRoute, kao i sučelje Yard.

Komponenta za dodavanje novog dvorišta izgledom je slična komponenti za uređivanje postojećeg dvorišta, a njena TypeScript datoteka prikazana je slikom 4.45

```
12  export class AddYardComponent implements OnInit {
13    yard: Yard = {};
14    submitted = false;
15
16    constructor(private yardService: YardService, private router: Router) { }
17
18    ngOnInit(): void {
19    }
20
21    saveYard(): void {
22      const data = {
23        name: this.yard.name,
24        address: this.yard.address,
25        phone: this.yard.phone,
26        website: this.yard.website
27      };
28      this.yardService.create(data)
29        .subscribe({
30          next: (res) => {
31            console.log(res);
32            this.submitted = true;
33            this.router.navigate(['yards']);
34          },
35          error: (e) => console.error(e)
36        });
37    }
38  }
```

Slika 4.45. *Datoteka add-yard.component.ts*

Nakon predavanja podataka za novo reciklažno dvorište, prikaz se pomoću metode navigate() prebacuje na komponentu yardsList, do koje vodi ruta yards.

4.5. Rad s recikliranjima

Evidentiranje recikliranja zamišljeno je i ostvareno prije svega kao unos količina otpadnih predmeta koji su kategorizirani u više skupina određenih Pravilnikom o gospodarenju otpadom. Slikom 4.46. prikazan je model Recycling koji sadrži neke od skupina otpada.

```
module.exports = (sequelize, Sequelize) => {
    const Recycling = sequelize.define("recycling", {
      solvents: {
        type: Sequelize.DOUBLE
      },
      acids: {
        type: Sequelize.DOUBLE
      },
      pesticides: {
        type: Sequelize.DOUBLE
      },
      metals: {
        type: Sequelize.DOUBLE
      },
      paper: {
        type: Sequelize.DOUBLE
      },
      textile: {
        type: Sequelize.DOUBLE
      },
      batteries: {
        type: Sequelize.DOUBLE
      },
      tires: {
        type: Sequelize.DOUBLE
      },
      glass: {
        type: Sequelize.DOUBLE
      },
      plastic: {
        type: Sequelize.DOUBLE
      }
    });
    return Recycling;
  };
```

Slika 4.46. *Model Recycling*

Vidljivo je kako su navedene sljedeće skupine otpada: otapala, kiseline, pesticidi, metali, papir, tekstil, baterije, gume, staklo i plastika. Po potrebi je moguće dodati i neke nove kategorije. Pri recikliranju je osim količina otpadnih predmeta potrebno evidentirati i korisnika koji je predmete donio na recikliranje, kao i reciklažno dvorište koje obavlja prihvat otpada. Kako bi to bilo moguće, prvo je potrebno definirati veze između recikliranja (model Recycling) s jedne, te korisnika (model User) i reciklažnog dvorišta (model Yard) s druge strane. Slikom 4.47. prikazan je isječak programskog koda koji je dodan u datoteku index.js, a kojim se definiraju odgovarajuće veze.

```
db.yards = require("./yard.model.js")(sequelize, Sequelize);
db.recyclings = require("./recycling.model.js")(sequelize, Sequelize);

db.yards.hasMany(db.recyclings, { as: "recyclings" });
db.recyclings.belongsTo(db.yards, {
  foreignKey: "yardId",
  as: "yard",
});

db.user.hasMany(db.recyclings, { as: "recyclings" });
db.recyclings.belongsTo(db.user, {
  foreignKey: "userId",
  as: "user",
});
```

Slika 4.47. *Definiranje veza s recikliranjem u datoteci index.js*

Preduvjet definiranju veza je uvoz preostalih potrebnih modela, modela Yard i modela Recycling, što je prvo učinjeno. S obzirom da se jedno recikliranje može obaviti u samo jednom dvorištu, a dvorište može obaviti puno recikliranja, veza između modela Yard i modela Recycling je 1:N, koju Sequelize omogućuje metodama hasMany() i belongsTo(). Ona se ostvaruje tako što se primarni ključ entiteta sa strane veze 1 (model Yard) doda kao strani ključ u entitet sa strane veze N (model Recycling). Istom se vezom i na isti način definira i odnos između modela User i modela Recycling. Uvidom u bazu podataka moguće je utvrditi kako su se tablici recyclings, pored stupaca definiranih modelom Recycling, dodali i stupci yardId i userId, koji predstavljaju odgovarajuće strane ključeve. Nakon stvaranja modela i definiranja veza moguće je u okviru datoteke recycling.controller.js stvoriti metodu za dodavanje novog recikliranja, koja je prikazana slikom 4.48.

```
exports.createRecycling = (req, res) => {
  if (!req.body.solvents || !req.body.acids || !req.body.pesticides || !req.body.metals || !req.body.paper || !req.body.textile ||
    !req.body.batteries|| !req.body.tires || !req.body.glass || !req.body.plastic || !req.body.yardId || !req.body.userId) {
    res.status(400).send({
      message: "Potrebno je unijeti sve podatke!"
    });
    return;
  }
  const recycling = {
    solvents: req.body.solvents,
    acids: req.body.acids,
    pesticides: req.body.pesticides,
    metals: req.body.metals,
    paper: req.body.paper,
    textile: req.body.textile,
    batteries: req.body.batteries,
    tires: req.body.tires,
    glass: req.body.glass,
    plastic: req.body.plastic,
    yardId: req.body.yardId,
    userId: req.body.userId
  };
  Recycling.create(recycling)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Dogodila se pogrješka pri dodavanju novog recikliranja!"
      });
    });
};
```

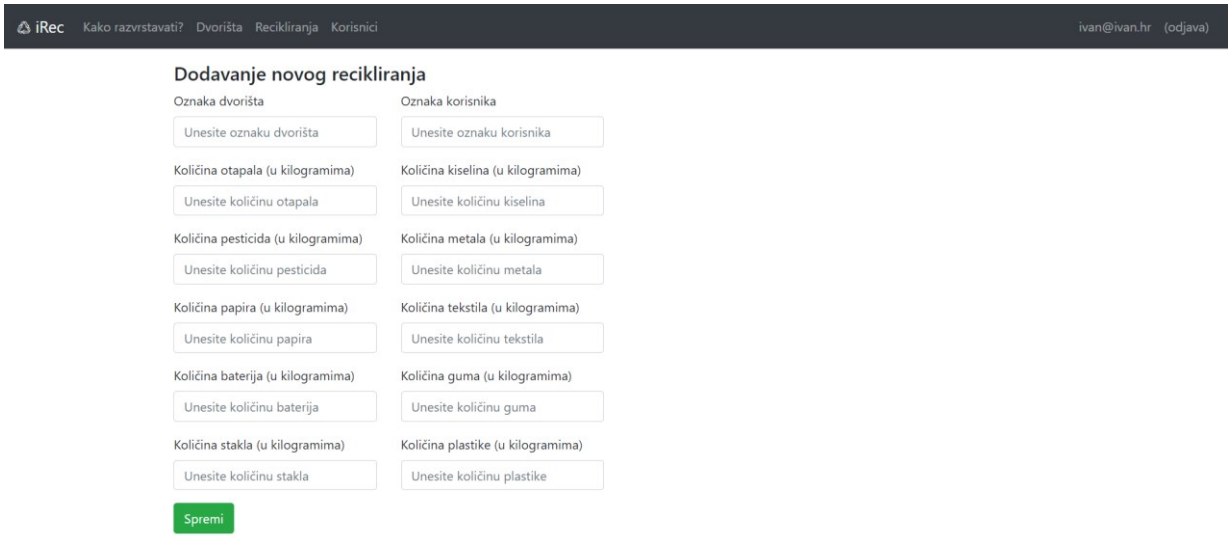
Slika 4.48. *Metoda za dodavanje recikliranja u datoteci recycling.controller.js*

Prvo se provjerava jesu li uneseni svi traženi podatci te se, ako nije unesen barem jedan, javlja pogrješka. Ako je sve u redu, stvara se novo recikliranje i zatim sprema u bazu podataka. Osim metode za dodavanje novog recikliranja, implementirane su i metode za brisanje, uređivanje i dohvaćanje koje su također identične srodnim metodama opisanima u prethodnim potpoglavljima. Sve one dostupne su klijentskoj strani preko RecyclingService-a koji je ondje generiran, a koristi sučelje Recycling, koje je prikazano slikom 4.49.

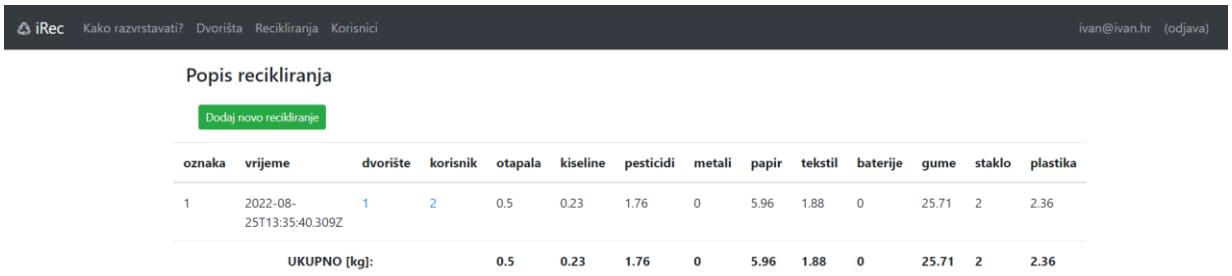
```
export interface Recycling {
    id?: number;
    solvents?: string,
    acids?: string,
    pesticides?: string,
    metals?: string,
    paper?: string,
    textile?: string,
    batteries?: string,
    tires?: string,
    glass?: string,
    plastic?: string,
    yardId?: number,
    userId?: number
    createdAt?: string;
}
```

Slika 4.49. *Sučelje Recycling*

Osim već spomenutih svojstava vezanih za količine otpadnih predmeta i strane ključeve, dodano je i svojstvo createdAt koje služi za prikaz nadnevka i vremena obavljanja pojedinog recikliranja. Iako ono nije eksplicitno definirano u modelu Recycling na poslužiteljskoj strani, ipak je automatski stvoreno zajedno sa svojstvom updatedAt, s obzirom da Sequelize podrazumijevano to čini osim ako nije naznačeno drukčije. Nakon svih obavljenih priprema stvorene su komponente addRecycling i recyclingsList, čiji su konačni izgledi prikazani slikama 4.50. i 4.51.



Slika 4.50. *Izgled komponente addRecycling*



Slika 4.51. *Izgled komponente recyclingsList*

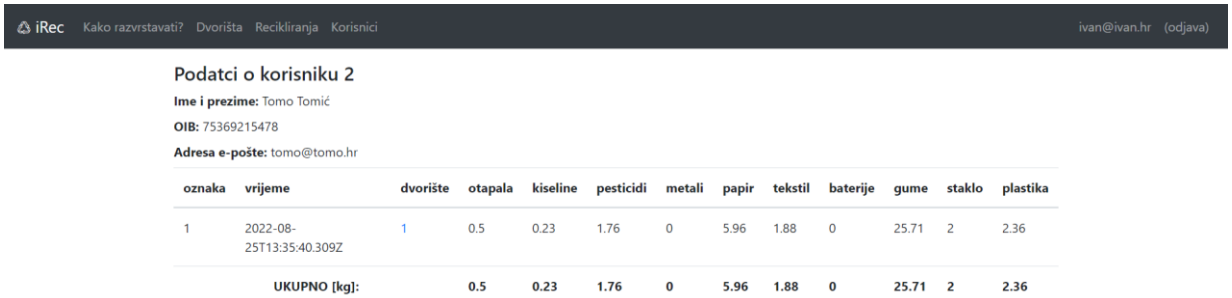
Primjetno je kako se radi o tabličnom prikazu svih obavljenih recikliranja, a na dnu tablice nalazi se ukupan iznos recikliranog otpada za svaku od već spomenutih kategorija. Za potrebe zbrajanja je u TypeScript datoteci komponente implementirana metoda findsum() i pozvana unutar metode za dohvaćanje svih recikliranja, kako je prikazano slikom 4.52.

```
36  findsum(data : any){
37    this.value=data
38    console.log(this.value);
39    for(let j=0;j<data.length;j++){
40      this.totalSolvents+= this.value[j].solvents
41      this.totalAcids+= this.value[j].acids
42      this.totalPesticides+= this.value[j].pesticides
43      this.totalMetals+= this.value[j].metals
44      this.totalPaper+= this.value[j].paper
45      this.totalTextile+= this.value[j].textile
46      this.totalBatteries+= this.value[j].batteries
47      this.totalTires+= this.value[j].tires
48      this.totalGlass+= this.value[j].glass
49      this.totalPlastic+= this.value[j].plastic
50    }
51  }
52
53  retrieveRecyclings(): void {
54    this.recyclingService.getAll()
55      .subscribe({
56        next: (data) => {
57          this.recyclings = data;
58          this.findsum(this.recyclings);
59          console.log(data);
60        },
61        error: (e) => console.error(e)
62      });
63  }
```

Slika 4.52. *Metoda findsum() u datoteci recyclings-list.component.ts*

Svaka kategorija ima svoju varijablu za spremanje ukupnog zbroja recikliranog otpada. Metoda findsum() poziva se unutar metode za dohvaćanje svih recikliranja te kao parametar prima upravo dohvaćena recikliranja, a sve navedeno izvršava se pri inicijalizaciji komponente.

Osim zbrajanja po kategorijama, na slici 4.51. također je primjetno kako je moguće kliknuti na oznake dvorišta i korisnika, a rezultat je prikaz podataka o istima. Na slici 4.53. nalazi se konačan izgled komponente userDetails, koja prikazuje podatke o određenom korisniku.



Slika 4.53. *Izgled komponente userDetails*

U ovom slučaju dohvaćaju se podatci o korisniku na način o kojem je bilo riječi na 21. stranici, u potpoglavlju 4.3., ali i recikliranja istog korisnika, za što je bilo potrebno učiniti određene preinake u datoteci user.controller.js na poslužiteljskoj strani, kao što je prikazano slikom 4.54.

```
17  exports.findOne = (req, res) => {
18      const id = req.params.id;
19      User.findByPk(id, {include: ["recyclings"]})
20        .then(data => {
21          if (data) {
22            res.send(data);
23          } else {
24            res.status(404).send({
25              message: "Korisnik oznake " + id + " nije pronađen!"
26            });
27          }
28        })
29        .catch(err => {
30          res.status(500).send({
31            message: "Dogodila se pogrješka pri dohvaćanju korisnika oznake " + id + "!"
32          });
33        });
34  };
```

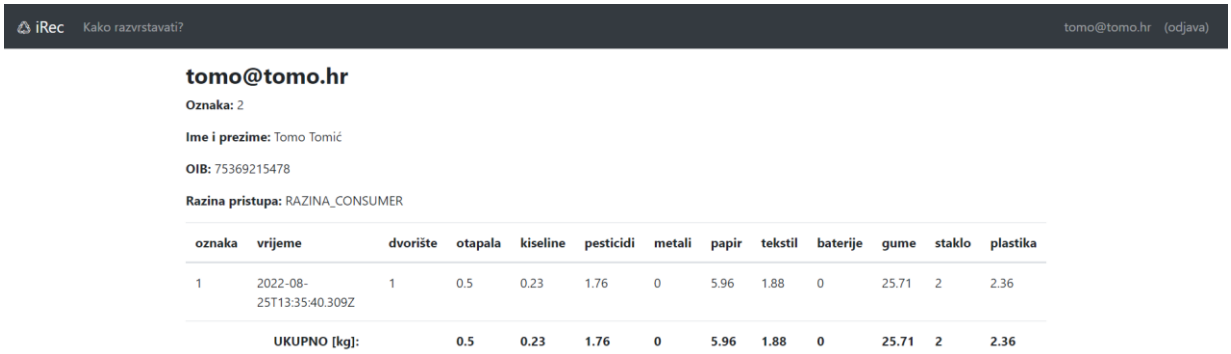
Slika 4.54. *Izmjena u datoteci user.controller.js*

Promjena je napravljena u retku 19, i to dodavanjem drugog parametra metodi findByPk(), kojim se naznačuje da se pri dohvaćanju korisnika s predanom oznakom također dohvate i pripadajuća recikliranja. Na isti su način dohvaćena i recikliranja po dvorištima. Osim na poslužiteljskoj, potrebne su male izmjene i na klijentskoj strani, točnije na sučeljima User i Yard kojima je dodano svojstvo recyclings koje je tipa Recycling[], kako je prikazano na slici 4.55., na primjeru sučelja Yard.

```typescript
import { Recycling } from "./recycling";

export interface Yard {
    id?: number;
    name?: string;
    address?: string;
    phone?: string;
    website?: string;
    recyclings?: Recycling[];
}
```

Slika 4.55. *Sučelje Yard*

Sve prethodno objašnjene funkcionalnosti namijenjene su isključivo korisniku razine pristupa recycler, dok korisnici razine consumer mogu vidjeti isključivo svoja recikliranja, i to na svom korisničkom profilu, tek nakon što su se prijavili. Slikom 4.56. prikazan je jedan takav profil.



Slika 4.56. *Profil korisnika razine pristupa consumer*

Neovlašteni pristup dijelovima aplikacije koji su namijenjeni isključivo razini recycler spriječen je, kako je već objašnjeno na 22. stranici, na poslužiteljskoj strani, odnosno tamošnjim metodama verifyToken() te isRecycler(). Također, na klijentskoj su strani u navigaciji pomoću strukturne direktive NgIf skrivene poveznice koje vode prema prikazima za rad s dvorištima, recikliranjima i korisnicima, a dio datoteke app.component.html u kojem se to vidi prikazan je slikom 4.57.

```html
<li class="nav-item">
  <a class="nav-link" routerLink="wasteItems" >Kako razvrstavati?</a>
</li>
<li class="nav-item" *ngIf="showRecyclerContent">
  <a class="nav-link" routerLink="yards">Dvorišta</a>
</li>
<li class="nav-item" *ngIf="showRecyclerContent">
  <a class="nav-link" routerLink="recyclings">Recikliranja</a>
</li>
<li class="nav-item" *ngIf="showRecyclerContent">
  <a class="nav-link" routerLink="users">Korisnici</a>
</li>
```

Slika 4.57. *Isječak iz datoteke app.component.html*

Za prikaz navedenih poveznica koristi se varijabla showRecyclerContent, definirana u TypeScript datoteci komponente, čiji je sadržaj prikazan slikom 4.58.

```
11  export class AppComponent {
12    title = 'Irec';
13    private roles: string[] = [];
14    isLoggedIn = false;
15    showRecyclerContent = false;
16    email?: string;
17
18    constructor(private storageService: StorageService, private authService: AuthService) { }
19
20    ngOnInit(): void {
21      this.isLoggedIn = this.storageService.isLoggedIn();
22      if (this.isLoggedIn) {
23        const user = this.storageService.getUser();
24        this.roles = user.roles;
25        this.showRecyclerContent = this.roles.includes('RAZINA_RECYCLER');
26        this.email = user.email;
27      }
28    }
29
30    logout(): void {
31      this.authService.logout().subscribe({
32        next: res => {
33          console.log(res);
34          this.storageService.clean();
35        },
36        error: err => {
37          console.log(err);
38        }
39      });
40      window.location.reload();
41    }
42  }
```

Slika 4.58. *Datoteka app.component.ts*

Vidljivo je kako je vrijednost varijable showRecyclerContent početno postavljena na false. Ako je korisnik prijavljen dohvaćaju se njegovi podatci te se, ako je dohvaćena razina recycler, varijabla postavlja na true. Time je ispunjen uvjet strukturne direktive iz predloška, pa se u tom slučaju prikazuje odgovarajući element <li>. Odjavom korisnika varijabla se vraća na svoje početno stanje. Na isti se način, dakle pomoću strukturne direktive NgIf i varijable isLoggedIn, u gornjem desnom kutu pojavljuje adresa e-pošte ako je korisnik prijavljen, a u suprotnom slučaju ondje stoji poveznica prema obrascu za prijavu.

5. ZAKLJUČAK

Recikliranje materijala veliki je izazov i vrlo važna djelatnost suvremene civilizacije koja osim ekoloških omogućuje i određene ekonomske učinke u smislu ušteda resursa, materijala i energije. S obzirom da je riječ o iznimno složenom procesu koji je, također, u ovom obliku aktivno zaživio tek u novije vrijeme, najučinkovitiji je kada se s njegovom provedbom počinje već u privatnim kućanstvima, koja imaju mogućnost odraditi prvi korak ili preduvjet, a to je razvrstavanje otpada. Ono je samo po sebi jednostavan postupak, no u praksi ljude nerijetko dovodi u nedoumicu kada ne znaju kojoj kategoriji otpada pripada, što osobito dolazi do izražaja kod predmeta načinjenih od više različitih materijala, koji su, uz to, nepoznatog sastava. Nakon razvrstavanja otpada, koje može obavljati svaki čovjek, slijedi samo recikliranje, za čiju su provedbu zaduženi isključivo pružatelji komunalnih usluga, točnije reciklažna dvorišta. U sklopu ovog diplomskog rada izrađena je aplikacija čija je svrha svim ljudima pružiti pomoć kod nedoumica pri razvrstavanju različitih otpadnih predmeta iz kućanstava, ali i omogućiti pružatelju komunalnih usluga dodavanje pripadajućih reciklažnih dvorišta i rad s istima te evidentiranje svih dolaznih recikliranja. Korisničko sučelje cjelokupne aplikacije vrlo je jednostavno i funkcionalno, čemu je doprinijelo prije svega korištenje Angulara i Node.js-a. Od njihovih mnogobrojnih prednosti posebno su do izražaja došle same direktive i ubrizgavanje ovisnosti kod Angulara, te kompatibilnost Node.js-a i mnogih modula, čime je razvoj aplikacije učinjen znatno jednostavnijim i bržim. Uz mogućnost primjene i drugih njihovih značajki i prednosti otvara se mogućnost za daljnje nadogradnje i poboljšanja aplikacije, poput primjerice omogućavanja grafičkog prikaza recikliranih materijala i sl.

LITERATURA

[1] Hrvatska enciklopedija, *Recikliranje*, https://www.enciklopedija.hr/Natuknica.aspx?ID=52144, pristupljeno 18. ožujka 2022.

[2] Bureau of International Recycling (BIR), *Non-ferrous metals*, https://www.bir.org/the-industry/non-ferrous-metals, pristupljeno 18. ožujka 2022.

[3] Bureau of International Recycling (BIR), *Paper*, https://www.bir.org/the-industry/paper, pristupljeno 18. ožujka 2022.

[4] Razvrstaj.me, *O aplikaciji*, https://www.razvrstaj.me/hr/o-aplikaciji/, pristupljeno 30. kolovoza 2022.

[5] Gencat, *Waste, how to place?*, https://residus.gencat.cat/en/ambits_dactuacio/sensibilitzacio/einesdigitals/residuonvas/index.html, pristupljeno 30. kolovoza 2022.

[6] RecycleSmart, *About us*, https://www.recyclesmart.com/about-us, pristupljeno 30. kolovoza 2022.

[7] Earth911, *iRecycle*, https://earth911.com/irecycle/, pristupljeno 30. kolovoza 2022.

[8] Descargas.com, *Telodoygratis*, https://www.descargas.com/en/app/telodoygratis-app-para-reciclar-y-regalar-cosas/android/, pristupljeno 30. kolovoza 2022.

[9] TypeScript, *The TypeScript Handbook*, https://www.typescriptlang.org/docs/handbook/intro.html, pristupljeno 20. ožujka 2022.

[10] Angular, *Angular Components Overview*, https://angular.io/guide/component-overview, pristupljeno 26. svibnja 2022.

[11] Angular, *Built-in directives*, https://angular.io/guide/built-in-directives, pristupljeno 26. svibnja 2022.

[12] OpenJS Foundation, *Introduction to Node.js*, https://nodejs.dev/learn, pristupljeno 26. svibnja 2022.

[13] Angular, *Workspace and project file structure*, https://angular.io/guide/file-structure, pristupljeno 1. lipnja 2022.

[14] TutorialsTeacher.com, *Two-way Data Binding in Angular*, https://www.tutorialsteacher.com/angular/two-way-data-binding, pristupljeno 15. srpnja 2022.

[15] Angular, *Lifecycle hooks*, https://angular.io/guide/lifecycle-hooks, pristupljeno 15. srpnja 2022.

[16] NPM, *body-parser*, https://www.npmjs.com/package/body-parser, pristupljeno 1. kolovoza 2022.

[17] Section, *Understanding Node.js Sequelize ORM Models*, https://www.section.io/engineering-education/understanding-nodejs-sequelize-orm-models/, pristupljeno 1. kolovoza 2022.

SAŽETAK

Tema diplomskog rada izrada je aplikacije za vođenje i savjetovanje pri reciklaži materijala. U prvom dijelu rada teoretski su opisane tehnologije i alati korišteni tijekom razvoja, s naglaskom na Angular, Node.js i PostgreSQL. Glavni dio rada opisuje tijek razvoja aplikacije, koji je podijeljen na nekoliko etapa. Prva etapa odnosi se na izradu korisničkog sučelja za pomoć pri razvrstavanju otpada. U drugoj etapi izrađen je sustav za autentikaciju i autorizaciju, koje su preduvjet za treću i četvrtu etapu, odnosno realizaciju upravljanja reciklažnim dvorištima i evidentiranje recikliranja.

Ključne riječi: Angular, Node.js, PostgreSQL, razvrstavanje, recikliranje

ABSTRACT

Application for guiding and consulting in material recycling

The topic of the master thesis is the creation of an application for guiding and consulting in material recycling. The first part of the thesis theoretically describes the technologies and tools used during development, with an emphasis on Angular, Node.js, and PostgreSQL. The main part of the thesis describes the application development process, divided into several stages. The first stage refers to the creation of a user interface for helping with waste sorting. In the second stage, a system for authentication and authorization was developed, which is a prerequisite for the third and fourth stages, i. e. the realization of the management of recycling yards and recording of recycling.

Keywords: Angular, Node.js, PostgreSQL, sorting, recycling

ŽIVOTOPIS

Ivan Ivković rođen je 12. prosinca 1996. u Osijeku. Nakon završetka gimnazije 2015. godine upisao je preddiplomski sveučilišni studij Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek kojeg je završio 2019. godine. Potom je na istom fakultetu upisao diplomski sveučilišni studij Računarstvo, izborni blok Informacijske i podatkovne znanosti. Stručnu praksu odradio je u Financijskoj agenciji (FINA).

PRILOZI

CD:

1. Diplomski rad „Aplikacija za vođenje i savjetovanje pri reciklaži materijala.docx“
2. Diplomski rad „Aplikacija za vođenje i savjetovanje pri reciklaži materijala.pdf“
3. Izvorni kod aplikacije