Web aplikacija za upravljanje reciklažnim dvorištima

Sudar, Jakov

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

Sveučilišni studij

WEB APLIKACIJA ZA UPRAVLJANJE RECIKLAŽNIM DVORIŠTIMA

Diplomski rad

Jakov Sudar

Osijek, 2021.

Sadržaj

1. UVOD

U današnjem svijetu digitalizacija je prisutna u sve većem broju društvenih sfera. Tehnologija brzo napreduje te je nezamisliv život bez nje. Tehnologija je omogućila razvoj brojnih aplikacija koje društvu olakšavaju svakodnevni život no ipak postoje područja koja nisu digitalizirana. Jedno takvo područje je reciklažno dvorište. Pošto ne postoji nikakva aplikacija za olakšavanje upravljanja reciklažnim dvorištem odlučio sam se za izradu jedne. Broj reciklažnih dvorišta raste zbog zakona o održivom gospodarenju otpadom koji nalaže da svaka jedinica lokalne samouprave koja broji više od 3000 stanovnika mora imati izgrađeno reciklažno dvorište [1]. Uvidom u Očevidnik reciklažnih dvorišta vidimo da je trenutno u Republici Hrvatskoj upisano 225 reciklažnih dvorišta što dovodi do zaključka da je potreba za ovakvom aplikacijom velika [2].

1.1 Zadatak diplomskog rada

Zadatak diplomskog rada je razviti potpuno funkcionalnu web aplikaciju koja ima osnovne elemente za upravljanje reciklažnim dvorištima. Aplikacija ima dva tipa korisnika a to su "ROLE_ADMIN" i "ROLE_DVORISTE". Administrator može dodavati nove gradove te njima dodjeljivati nova reciklažna dvorišta. Dvorišta mogu dodavati korisnike koji dolaze zbrinjavati otpad te unositi kategorije otpada kao i zbrinute količine. Dvorište će imati mogućnost pregledavanja statistike po kategorijama zbrinutog otpada po danu, mjesecu ili proizvoljnom odabranom vremenskom periodu. Postojat će i grafički prikaz zbrinutih količina u obliku grafa gdje će x-os predstavljati vrijeme a y-os predstavljati količinu zbrinutog otpada. Sve kategorije će biti iscrtane na jednom grafu što će omogućiti izravnu vizualnu usporedbu zbrinutih količina kao i analiziranje "udarnih" termina u kojima korisnici dvorišta najviše dolaze reciklirati.

Osim administratora i dvorišta aplikaciju će moći koristiti i neautorizirani korisnici tako da će moći upisivanjem svog OIB-a dobiti podatke o količinama otpada koje su osobno predali na zbrinjavanje svim reciklažnim dvorištima u Republici Hrvatskoj.

2. PRIMJENJENE TEHNOLOGIJE I ALATI

2.1 Java - Spring Boot

Java je moderni objektno orijentirani programski jezik besplatan za korištenje. Drugi je najpopularniji programski jezik te se koristi za razvoj mobilnih, web i desktop aplikacija kao i razvoj različitih servisa [3]. "Programiraj jednom i pokreni bilo gdje" je citat Java programskog jezika te označava njegovu kompatibilnost s različitim uređajima. To je postignuto jer se Java kod ne prevodi direktno u strojni kod već se prebacuje u bytecode kojeg zatim interpretira Java virtualni stroj. Stoga je potrebno samo instalirati Java virtualni stroj kako bi pokretali java aplikacije.

Spring Boot je okvir kreiran na programskom jeziku Java s ciljem jednostavnijeg razvijanja Java aplikacija [4]. Spring Boot pruža jednostavnost objavljivanja aplikacija jer u sebi ima ugrađen Tomcat server te ga u većini slučajeva nije potrebno dodatno podešavati.

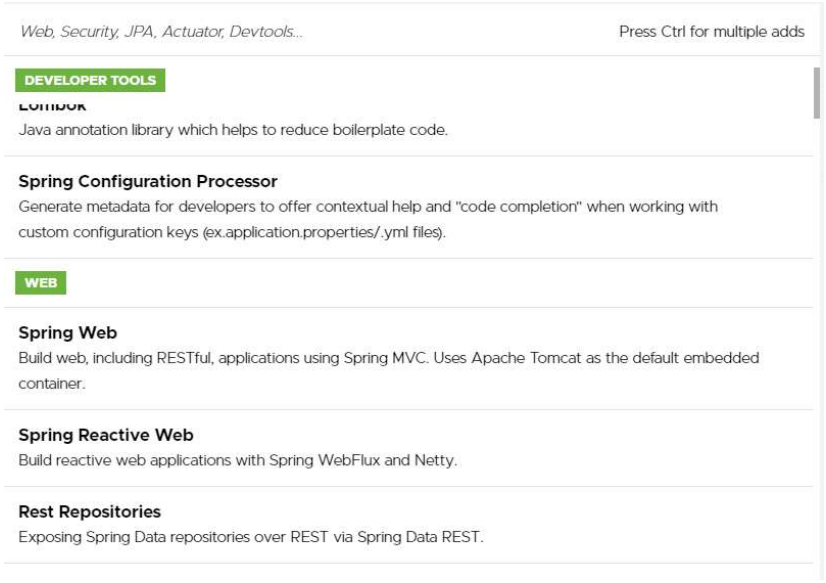
U Spring Boot-u sve se samokonfigurira, a uz pomoć različitih anotacija ne moramo brinuti o međuovisnostima unutar klasa. Spring Boot omogućuje programeru da razvija aplikaciju bez da se brine o previše tehničkih stvari. Ako odlučimo Spring Boot koristiti uz Maven dostupni su nam mnogi paketi za razvoj koji se jednostavno uključuju u projekt kroz pom.xml datoteku. Primjer uključivanja pogonskog programa za PostgreSQL bazu prikazan je u nastavku na slici 2.1.

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
```

Slika 2.1. Dodavanje paketa sa Maven repozitorija

Postoje i Spring Starter paketi koji u sebi sadrže više logički povezanih paketa. Jedan takav paket je Spring Security koji će biti korišten u svrhu zaštite aplikacije. Potreban je oprez kod uključivanja paketa jer se može dogoditi da se uključe paketi koji se ne koriste u aplikaciji te bespotrebno zauzimaju memoriju i usporavaju rad aplikacije.

Danas je jako popularan alat Spring Initializr koji omogućuje kreiranje projekta na jednostavan način putem korisničkog sučelja [5]. Nudi izbor svih paketa uz kratke opise čime ubrzava proces kreiranja projekta kao što je prikazano na slici 2.2.



Slika 2.2. Popis paketa u Spring Initializr

Prilikom izrade ovog rada Spring Boot će se koristiti za razvoj serverskog dijela aplikacije.

2.1.1 Hibernate

Hibernate je Java okvir koji pruža objektno relacijsko mapiranje (ORM). Pomoću Hibernate-a mapiramo entitete baze podataka u Java objekte. Kreiramo Java klasu te joj pridružimo anotaciju @Entity kako bi naznačili da se ne radi o običnoj klasi već da je to mapirana klasa iz baze. Sa @Table definiramo naziv tablice kao i schemu tablice. Kolone iz tablice pridružujemo atributima klase tako da iznad svakog atributa stavimo anotaciju @Column te navedemo ime kolone s kojom želimo mapirati. Ako atribut klase i kolona u bazi imaju isti naziv automatski će se mapirati te nije potrebno stavljati anotaciju. Primjer mapirane klase je na slici 2.3. Relacije između entiteta baze definiramo anotacijama @OneToMany, @OneToOne, @ManyToMany.

```
@Entity
@Table(name = "reciklaza")
public class Recyclation {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="weight")
    private float weight;

    @Column(name = "created_at")
    private java.sql.Timestamp createdAt;

    @Column(name = "updated_at")
    private java.sql.Timestamp updated_at;
```

Slika 2.3. Mapirana klasa

OneToOne anotacija

OneToOne anotacija je najjednostavnija i označava jedan naprama jedan vezu između 2 entiteta u bazi. U Java klasama to znači da jedna klasa sadrži drugu kao svoj atribut i obratno. @JoinColumn anotacija određuje strani ključ u entitetu koji definiramo. Primjer takve klase dan je na slici 2.4. gdje su povezane klase računalo i monitor. U tablici računalo imamo kolonu monitor_id koja je strani ključ prema entitetu monitor.

```
@Entity
@Table(name = "racunalo")
public class Računalo {

    @OneToOne
    @JoinColumn(name="monitor_id")
    private Monitor monitor;

}
```

Slika 2.4. OneToOne anotacija

OneToMany anotacija

OneToMany anotacija označava vezu gdje je jedan entitet vlasnik više entiteta, odnosno gdje više različitih entiteta pripada jednom entitetu. Primjer je dan na slici 2.5. gdje su povezani entiteti Dvoriste i Category. Ova veza označava da svako dvorište ima više kategorija i da svaka kategorija pripada određenom dvorištu. OneToMany anotacija ide u paru s anotacijom ManyToOne. To su istoznačne anotacije no o tome koja će se upotrijebiti ovisi s koje strane relacije se gleda. Kod OneToMany anotacije stavljena je oznaka mappedBy kojom se određuje po kojem atributu iz vezane klase će se taj atribut spariti. Za razliku od JoinColumn anotacije gdje navodimo naziv kolone iz baze koja predstavlja strani ključ.

Slika 2.5. OneToMany anotacija

ManyToMany anotacija

ManyToMany je najsloženija relacija od navedenih. Predstavlja slučaj kad oba entiteta mogu biti dodijeljena na više drugih entiteta te i sami mogu biti vlasnici više drugih entiteta kao što je prikazano na slici 2.6..

Slika 2.6. Veza više na više [6]

Kod veze više na više mora postojati dodatna tablica koja služi za uparivanje entiteta. Najčešće ta tablica sadrži samo glavne ključeve povezanih tablica. U Hibernateu se za ovakav slučaj koristi @ManyToMany anotacija gdje u @JoinTable navodimo naziv dodatne tablice te nazive kolona u kojima se nalaze glavni ključevi. Ukoliko bi u dodatnoj tablici bila potreba za upisivanjem još nekih podataka tada se ne bi mogla koristiti @ManyToMany anotacija već bi se kreirao novi entitet koji predstavlja samo tu dodatnu tablicu te s dvije @ManyToOne veze povezao

na tablice koje smo prvotno htjeli povezati. Primjer @ManyToOne povezivanja nalazi se na slici 2.7.

```java
@Entity
@Table(name = "Employee")
public class Employee {


    @ManyToMany(cascade = { CascadeType.ALL })
    @JoinTable(
        name = "Employee_Project",
        joinColumns = { @JoinColumn(name = "employee_id") },
        inverseJoinColumns = { @JoinColumn(name = "project_id") }
    )
    private List<Project> projects;
```

```java
@Entity
@Table(name = "Project")
public class Project {

    @ManyToMany(mappedBy = "projects")
    private List<Employee> employees;

}
```

Slika 2.7. ManyToMany anotacija

2.1.2 Spring security

Spring Security je jedan od Spring Startera koji uvelike pojednostavljuje razvoj sigurnih aplikacija. Omogućuje autentifikaciju i autorizaciju korisnika te nudi mogućnosti autentifikacije putem OAuth usluga. Prije početka rada potrebno je dodati Spring Security kao ovisnost kroz datoteku pom.xml kao što je prikazano na slici 2.8.

```xml
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Slika 2.8. Spring Security starter

Nakon ubacivanja paketa dobivamo mnoštvo korisnih klasa na korištenje te je postavljanje zaštita na pojedine krajnje točke iznimno jednostavno. Detaljni opis konfiguriranja sigurnosti je opisan pod poglavljem “Realizacija sustava”

2.1.3 REST API

Prilikom izrade web stranice postoje dva osnovna pristupa: MVC i REST. MVC predstavlja arhitekturu kod koje klijent od servera zatraži stranicu te potom server izgenerira cijelu stranicu i gotovu je vrati klijentu na prikaz. To stvara opterećenja na serveru te se iz tog razloga MVC rijetko koristi u složenim aplikacijama.

S druge strane REST predstavlja arhitekturu kod koje klijent od servera dobije podatke te se sam prikaz web stranice generira na klijentskom računalu što uvelike smanjuje opterećenje na serverskoj strani. Mana je što server u tom slučaju ne može pratiti zahtjeve po sesijama što dovodi do nemogućnosti pamćenja prijavljenog korisnika.

JWT token

Zbog nemogućnosti praćenja prijavljenog korisnika kod REST arhitekture uveo se JWT token (JSON web token). Token radi tako da korisnik u prvom zahtjevu pošalje serveru svoje korisničko ime i lozinku. Server provjeri jesu li podaci ispravni te ako jesu izdaje token u kojem se nalaze kriptirani podaci, najčešće korisničko ime i datum isteka tokena. Token se prije kriptiranja potpisuje privatnim ključem što osigurava da nitko ne može čitati niti mijenjati sadržaj tokena. U svakom idućem zahtjevu klijent uz ostale podatke šalje i taj token pomoću kojeg će ga server autentificirati i autorizirati [7].

2.2 React

React je JavaScript biblioteka za razvoj klijentske strane. Prednost Reacta naspram ostalih najvećih frontend okvira Angular-a i Vue-a je ta što React nije framework već biblioteka što ga čini puno lakšim za pokretanje na serveru te daje veću fleksibilnost prilikom razvoja aplikacija jer nije strogo definiran.

2.2.1 JSX

HTML je opisni jezik za kreiranje web sadržaja i nalazimo ga u svim frontend okvirima kao osnovu za prikaz sadržaja. Ipak kod razvoja aplikacija React-om sve se piše JSX sintaksom te se za vrijeme kompajliranja JSX kod prebacuje u HTML. U Reactu imamo funkcije koje vraćaju JSX kod te ih pozivamo u ostalim funkcijama i tako kreiramo modularnost koda. Primjer vraćanja JSX koda dan je na slici 2.9.

```
const Category = () => {
    //sva logika

    return (
        <div>
            {/* sav html kod */}
        </div>
    )
}
export default Category
```

Slika 2.9. React komponenta

2.2.2 CSS

CSS se koristi za stiliziranje HTML elemenata. U Reactu ga možemo koristiti na dva načina. Jedan način je da kreiramo .css datoteku u kojoj CSS selektorima odabiremo elemente te im postavljamo atribute. Drugi način je inline stiliziranje odnosno dodavanje CSS atributa direktno HTML elementu kao njegov atribut kao što je prikazano na slici 2.10. Inline stiliziranje se razlikuje od pisanja .css datoteke. Nazivi css atributa se pišu camel case umjesto kebab case te se ne stavlja znak jednakosti nego dvotočka.

```
<div style={{width:"100px", marginBottom:"10px"}}>

</div>
```

Slika 2.10. Inline stiliziranje

2.2.3 JavaScript

JavaScript se koristi za svu logiku aplikacije. JavaScript kod pišemo unutar funkcije te su nam dostupne sve naredbe iz JavaScript programskog jezika. React omogućuje ispis JavaScript varijabli u HTML kod tako da varijable obavijemo vitičastim zagradama kao na slici 2.11.

```
const Category = () => {

    let ime = "Jakov"
    return (
        <div style={{width:"100px", marginBottom:"10px"}}>
            {ime}
        </div>
    )
}
export default Category
```

Slika 2.11. Ispisivanje JS varijable

2.2.4 Ant Design

Ant Design je besplatni i trenutno najpopularniji[1] okvir za izradu korisničkog sučelja. Kompatibilan je sa React bibliotekom te nudi mnoštvo komponenti za korištenje. Dostupan je na [8]. Visoke je prilagodljivosti i izvrsne dokumentacije što ga čini pogodnim za korištenje prilikom izrade korisničkog sučelja. Preuzima se pomoću npm ili yarn registara kao na slici 2.12.

```
$ npm install antd
```

```
$ yarn add antd
```

Slika 2.12. Preuzimanje Ant Designa pomoću npm/yarn registra

2.3 PostgreSQL

PostgreSQL je relacijska baza podataka pogodna za rad s većim količinama podataka. Besplatna je i dostupna javnosti. Sintaksa je slična kao i većini SQL baza te postoje Java driveri za komunikaciju s PostgreSQL bazom podataka.

2.4 Eclipse

Eclipse je alat za razvoj Java aplikacija te nudi podršku za razvoj Spring Boot aplikacija. Alat je u potpunosti besplatan te nudi sve što je potrebno za razvoj serverskog dijela aplikacije. [9]

[1] Najpopularniji prema broju „starsa“ na službenom Github repozitoriju

2.5 Visual Studio Code

Visual Studio Code je besplatni alat za razvoj različitih aplikacija. Ima podršku za gotovo sve programske jezike koja se ostvaruje preuzimanjem besplatnih proširenja. Popularan je zbog svoje jednostavnosti i niskih resursnih zahtjeva. Najpopularniji je alat za razvoj klijentske strane aplikacije. [10]

3. REALIZACIJA SUSTAVA

Realizacija sustava se dijeli na dva glavna dijela koji su serverski dio i klijentski dio. Klijentski dio odnosno korisničko sučelje je zamišljeno kao SPA[2] razvijeno u React biblioteci. Serverska strana je razvijena u Java Spring Boot programskom okviru. Serverski dio je zadužen za komunikaciju s bazom, obradu rezultata te dostavljanje traženih podataka.

3.1 Baza podataka

Za uspješnu realizaciju cijelog sustava potrebno je spremati podatke o korisnicima, reciklažnim dvorištima, količinama zbrinutog otpada, kategorijama otpada i slično. Definiranje i implementiranje baze podataka je prvi korak u realizaciji sustava. Za bazu je odabrana PostgreSQL baza opisana u poglavlju 2.3. Za izbor baze jedini uvijet je bio da to bude relacijska baza zbog povezanosti entiteta.

Entiteti baze su: users, reciklaza, categories, dvorista, citites, roles te pogled weight_extremes. Povezanost entiteta prikazana je slikom 3.1.

users predstavlja korisnike reciklažnih dvorišta koji dolaze zbrinuti otpad. Sadrži osnovne podatke o korisniku: Ime, prezime, email te oib. Email atribut nije obavezan.

reciklaza predstavlja jedno zbrinjavanje otpada. Ukoliko osoba donese odjednom više različitih kategorija otpada za svaku kategoriju se unosi po jedan zapis u entitet reciklaza. Sadrži strane ključeve na osobu koja je zbrinula otpad, dvorište u kojem je zbrinut otpad te kategoriju otpada. Osim stranih ključeva sadrži i količinu zbrinutog otpada.

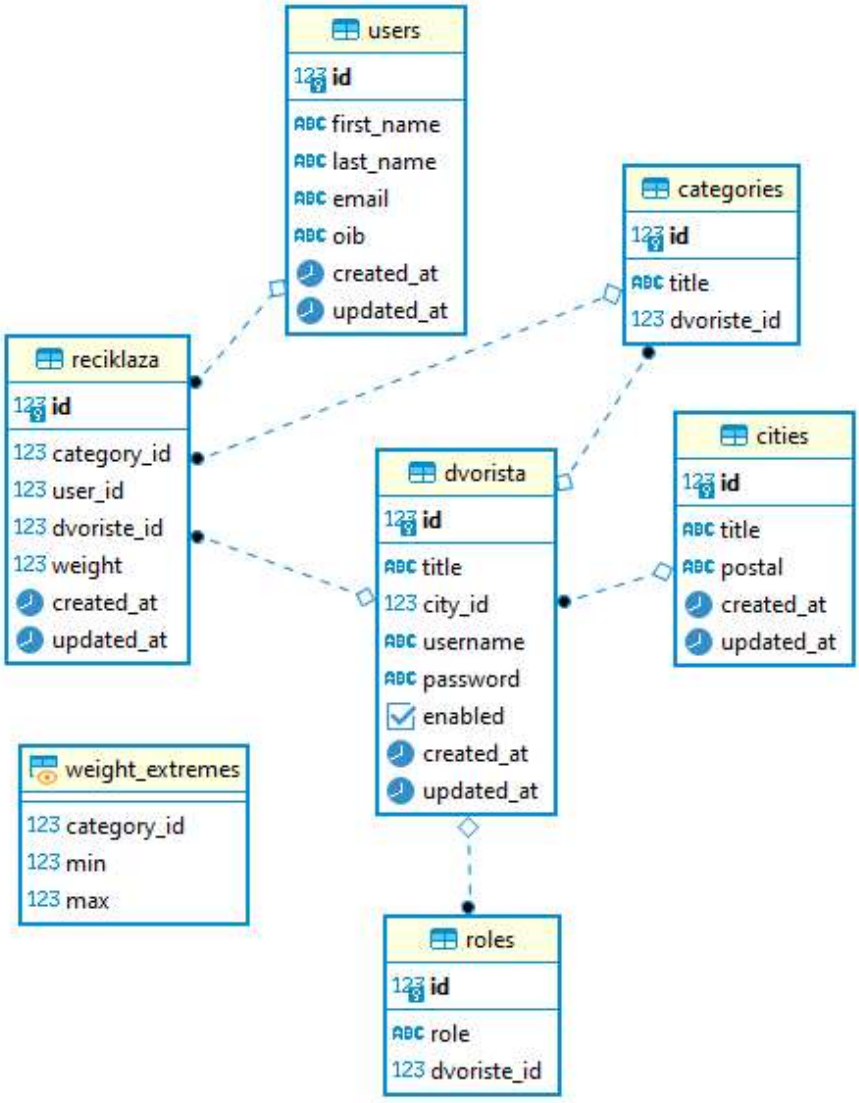
dvorista predstavlja tablicu u kojoj se nalaze sva registrirana dvorišta. Osim osnovnih atributa dvorišta sadrži i strani ključ na tablicu cities.

cities je tablica u kojoj se upisuju gradovi. Za svaki grad se upisuje naziv grada i poštanski broj.

roles je tablica s popisom uloga u aplikaciju. Sadrži naziv uloge i strani ključ na dvorište. Služi za autorizaciju prijavljenih dvorišta.

weight_extremes je pogled koji izračunava za svaku kategoriju najvećeg i najmanjeg predavatelja otpada

[2] engl. Single Page Application – arhitektura klijentske strane gdje postoji samo jedna stranica, a sadržaj se dinamički mijenja



Slika 3.1. Entiteti baze

Sintaksa kreiranja tablica je slična kao u svim SQL bazama Primjer koda za kreiranje tablice dvorista dan je slikom 3.2. U danom primjeru kreirana je tablica dvorista koja ima tri uvjeta. Prvi uvjet određuje da je id glavni ključ tablice. Drugi uvjet propisuje da username mora biti jedinstvena vrijednost u tablici. Treći uvjet je strani ključ na tablicu cities.

```
CREATE TABLE public.dvorista (
    id serial ,
    title varchar(100) NULL,
    city_id int4 NULL,
    username varchar NULL,
    "password" varchar NULL,
    enabled bool NULL,
    created_at timestamp NULL,
    updated_at timestamp NULL,
    CONSTRAINT dvorista_pk PRIMARY KEY (id),
    CONSTRAINT dvorista_un UNIQUE (username),
    CONSTRAINT dvorista_fk FOREIGN KEY (city_id) REFERENCES cities(id)
);
```

Slika 3.2. Kreiranje tablice u PostgreSQL

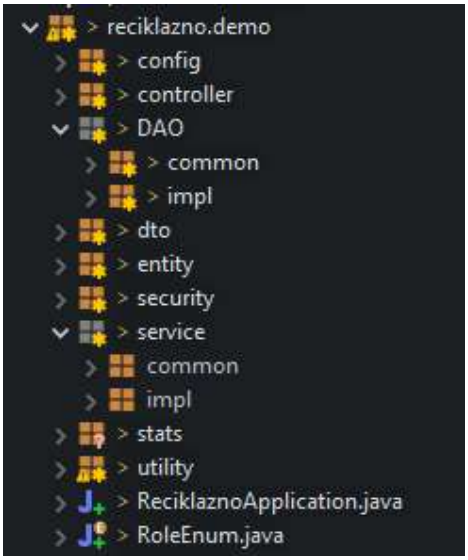
3.2 Izrada serverske strane

Za realizaciju serverske strane korišten je Java Spring Boot programski okvir opisan u poglavlju 2.1.

3.2.1 Arhitektura

Arhitektura je podijeljena u više razina. Krenuvši od najviše razine slojevi arhitekture su: kontroleri, servisni dio te dio za pristup objektima (DAO[3]). Prikaz paketa nalazi se na slici 3.3. gdje vidimo kako paketi sloja „service“ i „DAO“ imaju potpakete „common“ te „impl“. U common paketu se nalaze sučelja klasa dok se u impl paketu nalaze stvarne implementacije navedenih sučelja.

[3] engl. Data Access Object – objekt za pristupanje podacima



Slika 3.3. Arhitektura projekta

Entity paket sadrži klase koje predstavljaju sve entitete nabrojane u poglavlju 3.1.

Utility paket sadrži pomoćne klase kao što je TimeParser koje se koriste na više mjesta u kodu.

Stats paket sadrži sve potrebne klase za dohvaćanje i pripremu podataka za statistički prikaz na klijentskoj strani.

Security paket sadrži sve vezano uz sigurnost aplikacije.

Config je paket koji sadrži konfiguracijske klase

3.2.2 Application.properties datoteka

Application.properties je datoteka u koju možemo spremiti bilo kakav podatak kojem kasnije jednostavno pristupamo iz Java koda. Gotovo svi Spring paketi koriste ovu datoteku za određene konfiguracije. Primjer jednog takvog paketa je i Spring Data Starter koji nam omogućuje jednostavnu konekciju na bazu. Da bi paket funkcionirao potrebno je u application.properties datoteku dodati linije koda koje se vide na slici 3.4. prije znaka jednakosti navodi se naziv varijable a nakon znaka jednakosti se navodi vrijednost varijable. Kod korištenja gotovih paketa naziv varijable je bitan jer se paket oslanja na točno takav naziv no prilikom definiranja svojih varijabli naziv se bira proizvoljno.

```
spring.datasource.url= jdbc:postgresql://localhost:5432/Reciklazno
spring.datasource.password=admin
spring.datasource.username=postgres
spring.datasource.continue-on-error=true
```

Slika 3.4. Konfiguracija spajanja na bazu

U ovom projektu informacije o JWT tokenu su spremljene u takve varijable što se vidi na slici 3.5, te se tim varijablama pristupa iz koda putem @Value anotacije prikazane na slici 3.6.

```
jwt.secret = yiw1z2XJKQ7VHI/ck49j/RUAWm1gr
jwt.issuer = Sudo
jwt.type = JWT
jwt.audience = MyApp
```

Slika 3.5. Definiranje novih varijabli

```
@Value("${jwt.secret}")
private String jwtSecret;
@Value("${jwt.issuer}")
private String jwtIssuer;
@Value("${jwt.type}")
private String jwtType;
@Value("${jwt.audience}")
private String jwtAudience;
```

Slika 3.6. Pristupanje definiranim varijablama pomoću @Value anotacije

Osim application.properties datoteke moguće je i koristiti application.yaml datoteku koja ima istu funkciju no zapis imena varijabli je hijerarhijski, što vidimo na slici 3.7., te se mora paziti na uvučenost naziva varijabli.

```
spring:
        datasource:
            url: jdbc:postgresql://localhost:5432/Reciklazno
            username: admin
            password: postgres
            continue-on-error: true
```

Slika 3.7. Primjer application.yaml datoteke

3.2.3 Kontroleri

Kontroleri su najviša razina aplikacije i njihova uloga je primanje zahtjeva od klijenata. Registriranje kontrolera u Spring Bootu je jednostavno. Za definiranje kontrolera koristi se anotacija @RestController nad klasom kontrolera. Primjer jednog kontrolera dan je na slici 3.8. U danom primjeru vidimo i anotaciju @RequestMapping koja se koristi kada na sve rute kontrolera želimo dodati prefiks, u ovom slučaju svaka ruta ovog kontrolera počinje sa /api.

U kontroleru se nalazi više metoda od kojih u pravilu svaka predstavlja po jednu krajnju točku serverskog dijela aplikacije. Iznad svake metode se navodi anotacija koja označava vrstu HTTP zahtjeva koju će ta metoda obraditi. Te anotacije mogu biti @GetMapping, @PostMapping, @DeleteMapping i na isti način za sve ostale HTTP vrste zahtjeva. Nakon anotacije u zagradu se navodi ostatak putanje nakon već spomenutog prefiksa.

```
@RestController
@RequestMapping("/api")
public class CategoryController {

    @Autowired
    private CategoryService categoryService;


    @GetMapping("/categories")
    public List<Category>getAll(){
        return categoryService.findAll();
    }

    @GetMapping("/categories/{categoryId}")
    public Category findById(@PathVariable int categoryId) {
        return categoryService.findById(categoryId);
    }

    @PostMapping("/categories")
    @Transactional
    public Category addCategory(@RequestBody Category category,  Principal pwToken) {
        var user = (LoggedUser) ((UsernamePasswordAuthenticationToken) pwToken).getPrincipal();
        return categoryService.save(category, user.getId());
    }
}
```

Slika 3.8. Primjer kontrolera

Metode mogu primiti podatke s klijentske strane na više načina. U danom primjeru vidimo dva načina primanja podataka: putem @PathVariable i putem @RequestBody.

@PathVariable podrazumijeva da se u URL ubaci određeni podatak kao u primjeru metode findById gdje se u URL predaje podatak categoryId. Potrebno je upariti naziv argumenta metode i navedeni naziv u vitičastim zagradama u putanji krajnje točke.

@RequestBody koristimo za veće količine podataka te podrazumijeva da se s klijentske strane u tijelo zahtjeva ubaci JSON vrijednost podataka koje želimo primiti. Potrebno je spariti nazive varijabli u tijelu zahtjeva s nazivima parametara objekta koji smo anotirali @RequestBody anotacijom. Primjer uparivanja je na slici 3.9.

Slika 3.9. Upravianje naziva varijabli objekta i JSON

Ako u objektu postoji atribut koji nije naveden u tijelu zahtjeva vrijednost tog atributa u objektu će biti null. Mapiranje iz JSON-a u Java objekt se odvija automatski u Spring Boot frameworku.

3.2.4 Servis

Servis predstavlja sloj aplikacije gdje se primjenjuje poslovna logika nad podacima prije nego ih se proslijedi kontroleru. U manjim aplikacijama servis samo poziva DAO sloj te taj rezultat vraća direktno kontroleru bez modifikacije podataka.

Svaka klasa koja predstavlja servis se anotira @Service anotacijom. @Service anotacija tu klasu registrira kao poseban objekt te nam omogućava korištenje @Autowired anotacije u bilo kojoj drugoj klasi. Primjer @Autowired anotacije se nalazi na slici 3.8 gdje se vidi da je samo naveden naziv klase bez korištenja „new“ operatora i usprkos tomu možemo pozivati metode te klase.

3.2.5 DAO sloj

Dao sloj predstavlja najniži sloj serverskog dijela aplikacije. Zaslužan je za povezivanje na bazu, spremanje i čitanje iz baze. Svi upiti prema bazi odvijaju se u ovom sloju. Klase koje predstavljaju DAO sloj anotiraju se sa @Repository što ponovno omogućuje korištenje @Autowired anotacije.

Primjer klase DAO sloja dan je slikom 3.10. U danom primjeru nalazi se klasa EntityManager koja je Hibernate implementacija JPA[4] sučelja. Sva komunikacija s bazom se odvija kroz tu klasu. U primjeru metode findAll vidimo način kreiranja upita u HQL[5] koji se razlikuje od standardnih SQL upita.

U metodi findById vidi se jednostavnost pretraživanja entiteta po id. Koristi se find metoda klase EntityManager te joj se predaje klasa entiteta kojeg pretražujemo i id.

Spremanje pomoću Hibernate-a se vrši pozivom metode merge čiji je parametar objekt koji se sprema u bazu.

```
@Repository
public class CategoryDAOImpl implements CategoryDAO {

    @Autowired
    private EntityManager entityManager;

    @Override
    public List<Category> findAll() {
        Query query = entityManager.createQuery("from Category");
        return query.getResultList();
    }

    @Override
    public Category findById(int id) {
        return entityManager.find(Category.class, id);
    }

    @Override
    public Category save(Category category, int dvoriste_id) {
        var dvoriste = entityManager.find(Dvoriste.class, dvoriste_id);
        category.setDvoriste(dvoriste);
        return entityManager.merge(category);
        }

    @Override
    public void deleteById(int id) {
        Query query = entityManager.createQuery
                ("delete from Category where id=:id");
        query.setParameter("id", id);
        query.executeUpdate();
    }
}
```

Slika 3.10. Primjer klase u DAO sloju

[4] enlg. Java Persistance API – Niz specifikacija za upravljanje vezom Java objekt – relacijska baza
[5] engl. Hibernate Query Language – specifični jezik za kreiranje upita koristeći Hibernate

Osim dohvaćanja mapiranih entiteta iz baze, moguće je vršiti kombinirane upite nad više tablica te rezultat spremiti u objekt koji ne predstavlja entitet u bazi. Primjer se nalazi na slici 3.11.

```
@Override
public List<RecyclationUserOptimised> getAllfromUserGroupedByCategory(int userId) {
    String query = "select new com.reciklazno.demo.dto.RecyclationUserOptimised( c.id as id, c.title as title ,"
            + "(select sum(weight) from Recyclation r where r.user.id =:userId and r.category.id = c.id group by c.id),"
            + "(select maxVal from WeightExtremes wx where wx.categoryId = c.id ),"
            + "(select minVal from WeightExtremes wx where wx.categoryId = c.id ))"
            + "from \r\n"
            + " Recyclation r, Category c\r\n"
            + " where r.user.id = :userId  and r.category.id = c.id\r\n"
            + " group by  c.id, c.title\r\n"
            + " order by c.id";

    Query q = entityManager.createQuery(query).setParameter("userId", userId);
    var result = q.getResultList();

    return result;
}
```

Slika 3.11. Izvođenje prilagođenih upita

U danom primjeru u upitu stvaramo objekt RecyclationUserOptimised te u konstruktor predajemo vrijednosti dohvaćene iz baze.

Često Hibernate nije najbolje rješenje za pisanje prilagođenih upita te se u tim slučajevima može koristiti klasa JdbcTemplate koju također uključimo putem @Autowired anotacije. Ta klasa sadrži metode koje koriste nativne upite nad bazom te se često koristi kod zamršenih upita zbog boljih performansi u odnosu na Hibernate. Primjer korištenja JdbcTemplate klase dan je na slici 3.12. Kod korištenja ovog načina ne može se izravno kreirati objekt već je potrebno kreirati mapper, u ovom slučaju RowMapper, koji će popuniti objekt. Također kod ovakvog načina ne trebamo povezivati klasu kao što je opisano u poglavlju 2.1.1.

```
String queryStat = "select sum(r.weight) as \"total\", date_trunc('"+regex+"', r.created_at) as date, c.title from reciklaza r \r\n"
        + "inner join categories c on c.id = r.category_id \r\n"
        + "where r.dvoriste_id = ? and r.created_at between to_timestamp(? ,'DD/MM/YYYY HH24:MI:SS')AND to_timestamp(?, 'DD/MM/YYYY HH24:MI:SS')"
        + "group by date_trunc('"+regex+"', r.created_at), c.title\r\n"
        + "order by date_trunc('"+regex+"', r.created_at)";

List<StatistikaModel> ukupniRezultat = jdbcTemplate.query(queryStat, new Object[] {userId,startDate, endDate}, new RowMapper<StatistikaModel>() {
    @Override
    public StatistikaModel mapRow(ResultSet rs, int rowNum) throws SQLException {
        StatistikaModel temp = new StatistikaModel();
        Timestamp ts = rs.getTimestamp("date");
        temp.setDate(ts);
        temp.setValue(rs.getFloat("total"));
        temp.setTitle(rs.getString("title"));
        return temp;
    }

});
```

Slika 3.12. Primjer korištenja JdbcTemplate klase

3.2.6 Zaštita aplikacije

Za sigurnost aplikacije korišten je paket Spring Security opisan u poglavlju 2.1.2. Nakon uključivanja paketa u projekt potrebno je kreirati konfiguracijsku klasu koja proširuje klasu WebSecurityConfigurerAdapter. Na svaku konfiguracijsku klasu stavlja se anotacija @Configuration te će se prilikom pokretanja aplikacije te klase izvršiti na samom početku.

U navedenoj klasi potrebno je pregaziti implementacije nekoliko metoda kako bi se dobilo željeno ponašanje. Prva metoda koju treba pregaziti je configure metoda u kojoj postavljamo klasu za učitavanje korisnika iz baze kao što je prikazano na slici 3.11.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService);
}
```

Slika 3.13. Postavljanje klase za učitavanje korisnika

Klasa koju predajemo se nalazi na slici 3.14. Jedina metoda koja mora postojati u toj klasi propisana je sučeljem UserDetailsService i to je metoda loadUserByUsername. U toj metodi dohvaćamo dvorište iz baze te njegove pripadajuće uloge. Rezultat metode je UserDetails klasa odnosno u ovom slučaju klasa LoggedUser koja proširuje ugrađenu UserDetails klasu. LoggedUser je kreiran zbog dodavanja dodatnih atributa koji ne postoje u klasi UserDetails koju koristi Spring Security.

```java
@Service("userDetailsService")
public class MyUserDetailsService implements UserDetailsService {

    @Autowired
    private DvoristeDAO dvoristeDao;


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        Dvoriste tempDvoriste = dvoristeDao.findByUsername(username);

        List<GrantedAuthority> authorities = buildUserAuthority(tempDvoriste.getRoles());

        return buildUserForAuthentication(tempDvoriste, authorities);
    }

    private List<GrantedAuthority> buildUserAuthority(Set<Role> roles) {

        Set<GrantedAuthority> setAuths = new HashSet<GrantedAuthority>();
        for(Role userRole: roles) {
            setAuths.add(new SimpleGrantedAuthority(userRole.getRole().toString()));
        }
        return new ArrayList<GrantedAuthority>(setAuths);
    }

    private UserDetails buildUserForAuthentication(Dvoriste dvoriste, List<GrantedAuthority> authorities) {

        var username = dvoriste.getUsername();
        var password = dvoriste.getPassword();
        boolean enabled = true;
        boolean accountNonExpired = true;
        boolean credentialsNonExpired = true;
        boolean accountNonLocked = true;

        var loggedUser = new LoggedUser(username, password, enabled, accountNonExpired,
                credentialsNonExpired, accountNonLocked, authorities);
        loggedUser.setId(dvoriste.getId());
        loggedUser.setTitle(dvoriste.getTitle());

        return loggedUser;
    }

}
```

Slika 3.14. Klasa za dohvaćanje korisnika iz baze

Druga metoda koja se treba pregaziti je također metoda configure no s drugim parametrima. U toj metodi se mogu dodavati filteri za autentifikaciju i autorizaciju korisnika kao i odrediti koje krajnje točke aplikacije su dostupne kojim korisnicima. Izgled te metode dan je slikom 3.15.

```
@Override
protected void configure(HttpSecurity http) throws Exception {

    http.cors().and().csrf().disable()
    .addFilter(new JwtAuthenticationFilter(authenticationManager(), jwtAudience, jwtIssuer, jwtSecret, jwtType))
    .addFilter(new JwtAuthorizationFilter(authenticationManager(), jwtAudience, jwtIssuer, jwtSecret,
            jwtType,jdbcTemplate, userDetailsService))
    .authorizeRequests(authorizeRequests ->
            authorizeRequests
                    .antMatchers("/api/**").access("hasAnyRole('DVORISTE', 'ADMIN')")
                    .antMatchers("/admin").hasAnyRole("ADMIN")
                    .antMatchers("/statistika/dohvatiStatistiku/user").permitAll()
                    .anyRequest().fullyAuthenticated()
    )
    .httpBasic().realmName("My org ream")
    .and()
    .sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

Slika 3.15. Pregažena konfiguracijska metoda

Na slici 3.15 dodana su dva filtera JwtAuthenticationFilter i JwtAuthorizationFilter. Autentifikacijski filter provjerava korisničko ime i lozinku te ako su ispravni kreira token. Autorizacijski filter čita korisničko ime iz tokena te pomoću imena dohvaća ostale podatke i uloge iz baze. Uloge su ključne za pristup resursima jer su pojedini resursi dostupni samo korisnicima s određenom ulogom.

Oba filtera su klase koje proširuju već postojeće, ugrađene klase kako bi postigli željenu funkcionalnost. Ugrađena autentifikacijska klasa ne kreira token već korisnika sprema u context te je iz tog razloga bilo potrebno kreirati novi filter. Kod ugrađene autorizacijske klase bilo je potrebno promijeniti način autoriziranja korisnika tako da korisničko ime dohvaća iz tokena a ne iz contexta kako je implementirano u paketu.

3.3 Izrada klijentskog dijela aplikacije

Klijentski dio aplikacije predstavlja dio aplikacije koji korisnik vidi i s kojim interaktira. U ovom radu klijentski dio izrađen je koristeći React opisan u poglavlju 2.2. Klijentski dio kreiran je kao jednostranična aplikacija. Na samom početku postoji tek jena HTML stranica s jednim elementom koji predstavlja korijen aplikacije. U taj korijenski element će se učitavati sadržaj u ovisnosti od korisničkih radnji.

Kako bi koristili React biblioteku potrebno ju je uključiti u svaku JavaScript datoteku. Primjer uključivanja React biblioteke kao i vlastite definirane datoteke dan je slikom 3.16. Dani primjer je datoteka index.js koja predstavlja početnu stranicu. Na primjeru se vidi postavljanje komponente AppRouter u korijenski element.

```
import React from 'react';
import ReactDOM from 'react-dom';
import AppRouter from './components/AppRouter'

ReactDOM.render( <AppRouter/>,document.getElementById('root'));
```

Slika 3.16. Izgled datoteke index.js

3.3.1 React router

Ruteri su komponente koje simuliraju višestraničnu aplikaciju. U ovisnosti o odabranom URL-u ruter će prikazati određenu komponentu. Primjer rutera je na slici 3.17. Na prikazanom primjeru nalaze četiri različita URL-a za koje se prikazuju različite komponente. Ruteri se mogu ugnježđivati tako da u komponentu DvoristeProfil dodamo još jedan ruter koji će dalje prikazivati tražene komponente. To je i učinjeno kroz komponentu DvoristeContent koja predstavlja glavni dio komponente DvoristeProfil kao što je prikazano na slici 3.18.

```
const AppRouter = () => {

    const [user, dispatch] = useReducer(userReducer, [])
    const [recyclations, recDispatch] = useReducer(recyclationReducer, [])

    return(
    <RecContext.Provider value= {{recyclations, recDispatch}}>
        <ProfileContext.Provider value={{dispatch, user}}>
            <BrowserRouter>
            <div>
            <Switch>
                <Route path="/" component={HomePage} exact="true"/>
                <Route path="/login" component={LoginPage} />
                <Route path="/profile"  component={DvoristeProfil} />
                <Route path="/start"  component={StartComponent} />
            </Switch>
            </div>
            </BrowserRouter>
        </ProfileContext.Provider>
    </RecContext.Provider>
    )
};

export default AppRouter;
```

Slika 3.17. Primjer rutera

```
const DvoristeContent = () => (
    <>
        <Route path="/profile" component={Profil} exact="true"/>
        <Route path="/profile/add-recyclation" component={DodajReciklazu} />
        <Route path="/profile/daily" component={DnevnaStatistika} />
        <Route path="/profile/graph" component={GrafStatistika} />
        <Route path="/profile/period" component={RazdobljeStatistika} />
        <Route path="/profile/monthly" component={MjesecnaStatistika} />
        <Route path="/profile/admin" component={AdminProfile} />
        <Route path="/profile/settings" component={Settings} />
    </>
)
```

Slika 3.18. Ugnježđeni ruter

3.3.2 Komponente

Sve što na kraju korisnik vidi su komponente. React razlikuje klasne komponente i funkcijske komponente. Funkcijske komponente su novije te su samo one korištene u ovom radu. Svaka komponenta ima tri glavna dijela:

- Import dio – dio gdje se uključuju ostale komponente i funkcije
- Logički dio – dio JavaScript koda gdje se programira logika
- Return dio – dio koji vraća JSX kod opisan u poglavlju 2.2.1.

Kod programiranja logičkog dijela dostupni su nam brojni React koncepti koji olakšavaju programiranje. React Hooks je među najnovijim konceptima te olakšavaju praćenje stanja komponente.

3.3.3 React hook useState i useEffect

Dva najpopularnija React hooka su useState i useEffect. Hook useState se koristi za praćenje stanja React komponente. State hook se definira na način prikazan slikom 3.19. Data predstavlja getter za definirano stanje dok setData predstavlja setter. Vrijednost u zagradama nakon ključne riječi useState je inicijalna vrijednost stanja.

```
const [data, setData] = useState("")
```

Slika 3.19. useState hook

Stanje mijenjamo pozivom funkcije setData() te predajemo novu vrijednost stanja. Nova vrijednost stanja može biti bilo kojeg tipa podatka, može biti objekt ili polje no nije preporučljivo mijenjati tip. Na svaku promjenu stanja React komponenta će se ponovno prikazati na ekranu s novim vrijednostima. Bitno je napomenuti da funkcija setData() nije atomska te je potrebno određeno vrijeme dok se stanje zapravo ne postavi. Vrijeme postavljanja stanja je reda 10 ms.

UseEffect hook se koristi za kontrolu životnih metoda komponente. Sintaksa korištenja useEffect hooka dana je slikom 3.20.

Slika 3.20. useEffect hook

Na mjesto "input" upisuje se naziv stanja koje želimo promatrati. Ukoliko bi tu pisalo data onda bi se taj hook izvršio na svaku promjenu stanja data. Ako umjesto "input" ne piše ništa već ostanu prazne zagrade taj će se hook izvršiti samo prilikom inicijaliziranja komponente. S time mijenjamo metodu componentWillMount. U return dio ide kod za čišćenje komponente (npr. brisanje listenera). Return dio mijenja metodu componentWillUnmount.

3.3.4 Konzumiranje API-ja

Konzumiranje API-ja znači pozivanje krajnjih točaka serverskog dijela aplikacije. Na taj način se povezuju klijentski i serverski dio. Za pozivanje API-ja je korištena JavaScript funkcija fetch(). Za svaki poslan zahtjev potrebno je iz lokalne memorije preglednika učitati token i poslati ga skupa sa zahtjevom. Kako se taj postupak ne bi ponavljao kreirana je fetcher() funkcija koja pruža jednostavnije sučelje za pozivanje API-ja te sakriva učitavanje tokena. Funkcija se nalazi na slici 3.21.

```
function updateOptions(options){
    const update = {...options}
       update.headers = {
           ...update.headers,
           'Cache-Control': 'no-cache',
           'Access-Control-Allow-Origin': '*',
       }
    if(localStorage.token){
        update.headers ={
            ...update.headers,
            Authorization: localStorage.token
        };
    }
    return update;
}

export default async function fetcher(url,options){
   return await fetch(baseURL + url,updateOptions(options)).
        then((res)=>{
           if(res.status === 403){
            window.location.href = "http://localhost:3000/";
           }
            return res;
   }).catch((e)=>{
       console.log(e,"error")
   })
}
```

Slika 3.21. Funkcija fetcher()

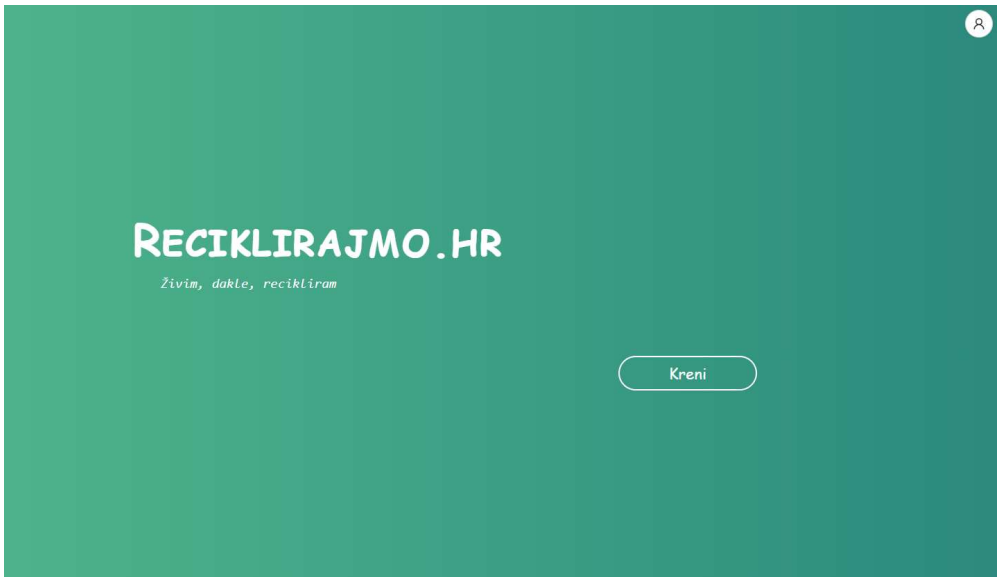
Osim postavljanja tokena funkcija djeluje i kao posrednik između poziva i zahtjeva te za svaki odgovor statusa 403 (zabranjen pristup) će odjaviti korisnika i preusmjeriti ga na početnu stranicu. Tako se ne mora kod slanja svakog zahtjeva provjeravati je li korisniku istekao token ili više nije autoriziran. Primjer korištenja fetcher() funkcije dan je slikom 3.22. Na primjeru se odmah prilikom prikaza komponente pomoću useEffect hooka dohvaćaju podaci potrebni za ispravan prikaz komponente.

```
useEffect(() => {
fetcher("api/dvorista/"+localStorage.userId)
.then(res=>res.json())
.then(res=> setcity(res.city))
    fetcher("api/recyclations/dvoriste/stats/"+ localStorage.userId)
    .then(res=>res.json())
    .then(res=>{
        setsumStats(
            res
        )
    })
},[])
```

Slika 3.22. Pozivanje API-ja koristeći fetcher() funkciju

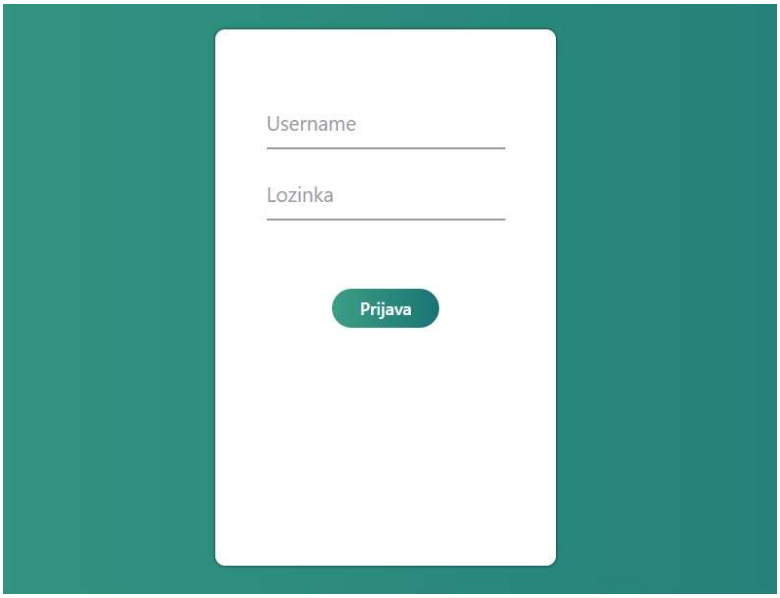
4. IZGLED APLIKACIJE

Ulaz u aplikaciju je početna stranica odakle se može kliknuti na gumb kreni ili na okrugli gumb u gornjem desnom kutu.



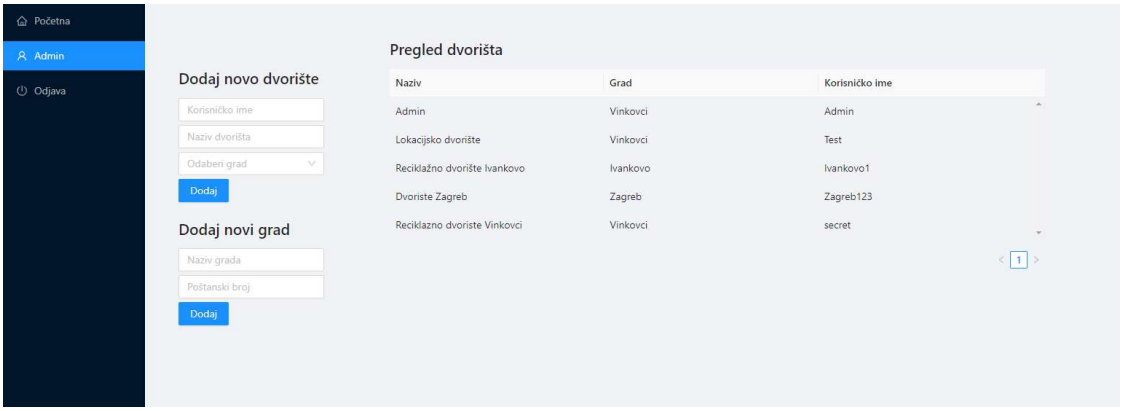
Slika 4.1. Početna stranica aplikacije

Klikom na gumb u gornjem desnom kutu dolazimo do stranice za prijavu.



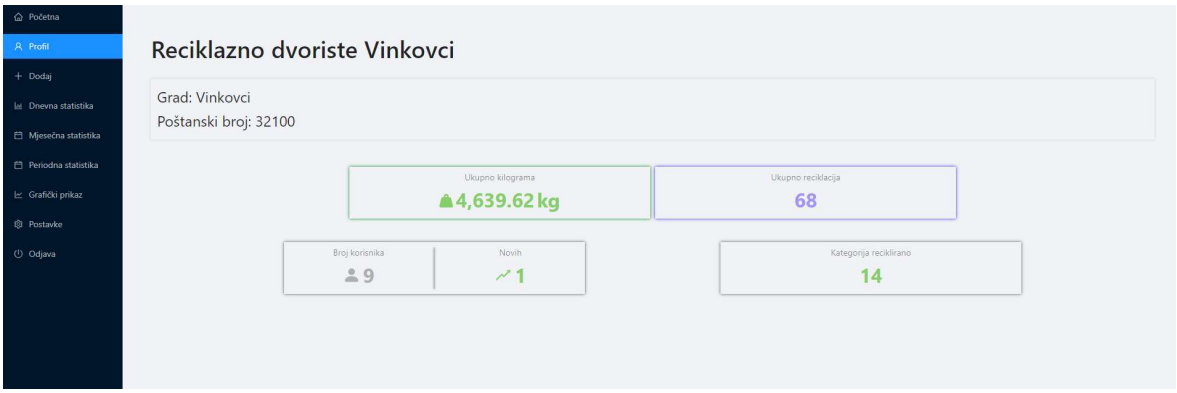
Slika 4.2. Stranica za prijavu

Upisivanjem korisničkog imena i lozinke korisnik se prijavljuje u sustav. Ako ima ulogu administrator prikazuje mu se administratorski profil gdje može dodavati novo dvorište ili novi grad te dobiti uvid u sva registrirana dvorišta kao što je prikazano na slici 4.3.



Slika 4.3. Administratorski profil

Ako se prijavi korisnik s ulogom dvorište prikazuje mu se njegov profil gdje na početnom zaslonu može vidjeti osnovne podatke o dvorištu kao što su ukupna količina zbrinutog otpada, broj zbrinjavanja, broj korisnika kao i broj novih korisnika u tekućem mjesecu te koliko kategorija otpada je reciklirano u dvorištu. Navedeno je prikazano na slici 4.4.

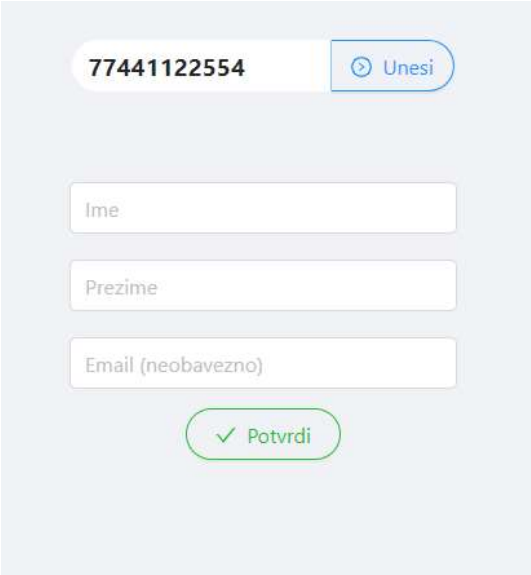


Slika 4.4. Profil dvorišta

S lijeve strane nalazi se navigacija kroz menije:

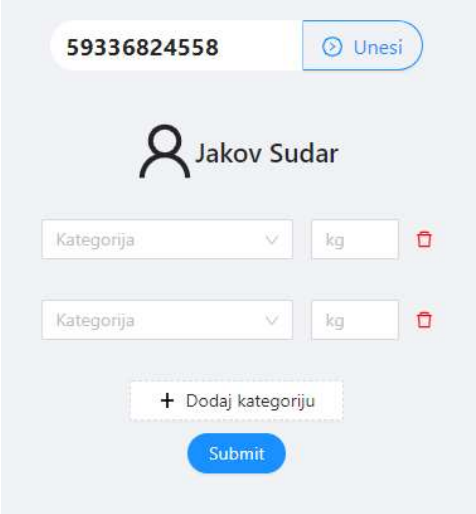
- Početna
- Profil
- Dodaj
- Dnevna statistika
- Mjesečna statistika
- Periodna statistika
- Grafički prikaz
- Postavke
- Odjava

Klikom na „Početna“ vraća se na stranicu prikazanu slikom 4.1. Klikom na profil dolazi se na stranicu prikazanu slikom 4.4. „Dodaj“ služi za dodavanje novog zbrinjavanja kad korisnik dvorišta donese otpad. Prvo se unosi OIB osobe te ako osoba već postoji u bazi odmah se prelazi na unos zbrinutog otpada . Ako osoba ne postoji u bazi prvo se unose njeni osnovni podaci što je prikazano na slici 4.5



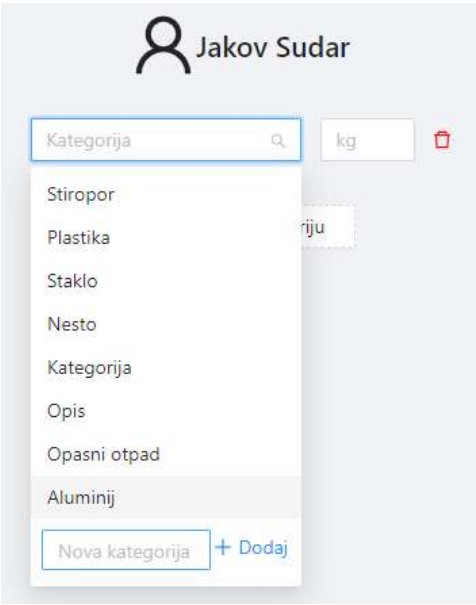
Slika 4.5. Unošenje osobnih podataka

Kod unosa zbrinutog otpada dinamički se mogu dodati forme za unos klikom na gumb „Dodaj kategoriju“ kao što je prikazano na slici 4.6.



Slika 4.6. Dodavanje zbrinutog otpada

Iz padajućeg izbornika biramo kategoriju koju je osoba predala na zbrinjavanje. Ako nema tražene kategorije moguće ju je upisivanjem naziva kategorije i klikom na „dodaj“ u padajućem izborniku prikazanom na slici 4.7.



Slika 4.7. Padajući izbornik za odabir kategorije

Nakon završetka unosa podataka klikom na „potvrdi“ dobivamo pregled svih unesenih kategorija te ih je moguće obrisati ako je došlo do pogreške kao što je prikazano na slici 4.8.

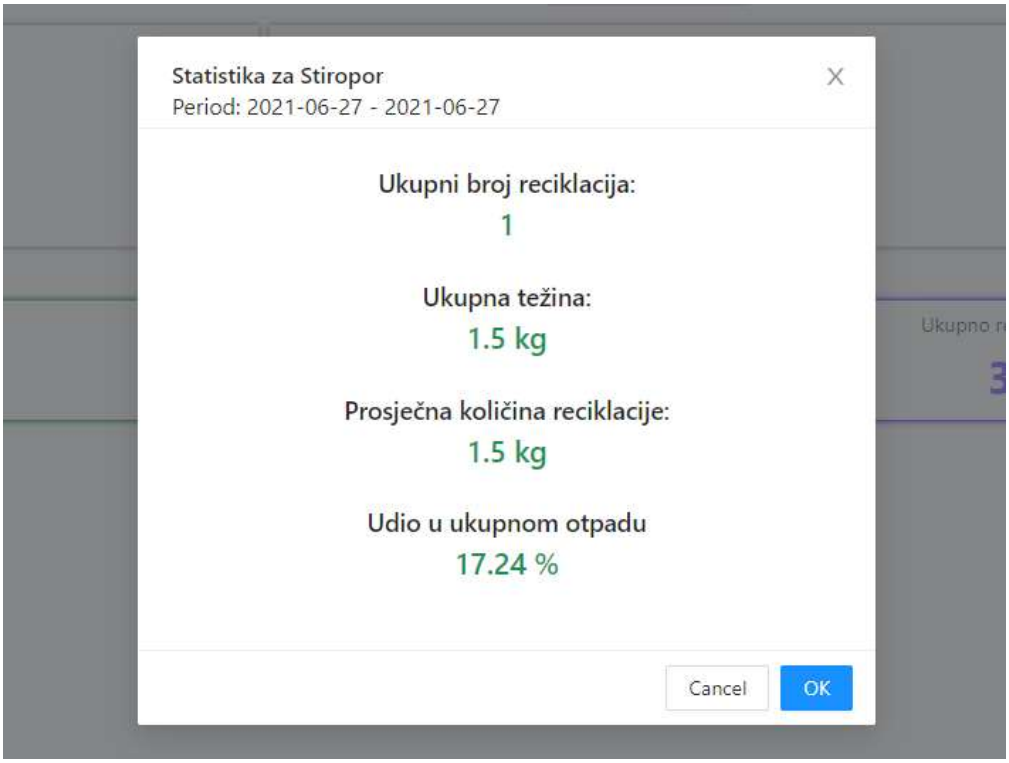
Slika 4.8. Pregled unesenih vrijednosti

Ako iz navigacije odaberemo „Dnevna statistika“ dobijemo prikaz svih recikliranih kategorija za dan koji odaberemo. Navedeno je prikazano na slici 4.9.

Slika 4.9. Prikaz dnevne statistike

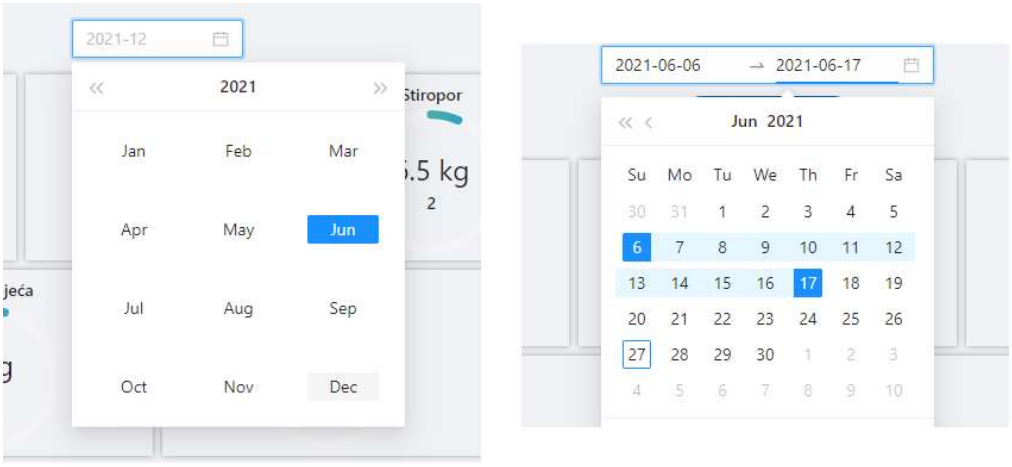
Svaka kategorija prikazana je krugom koji je napunjen do određene vrijednosti u ovisnosti o tome kolika je ukupna težina zbrinute kategorije. Kategorija s najvećom težinom imat će do kraja pun krug. Kategorija s najmanjom težinom imat će najmanje pun krug te će se ostale kategorije u odnosu na minimalnu i maksimalnu normalizirati i napuniti krug s obzirom na rezultat normalizacije. Unutar kruga je navedena ukupna težina otpada kao i ukupan broj zbrinjavanja te kategorije za odabrani dan.

Klikom na neku od kategorija otvara se skočni prozor u kojem se može pročitati dodatna statistika za tu kategoriju kao što je postotni udio u recikliranom otpadu taj dan te prosječna količina zbrinutog otpada po osobi kao što je prikazano na slici 4.10.



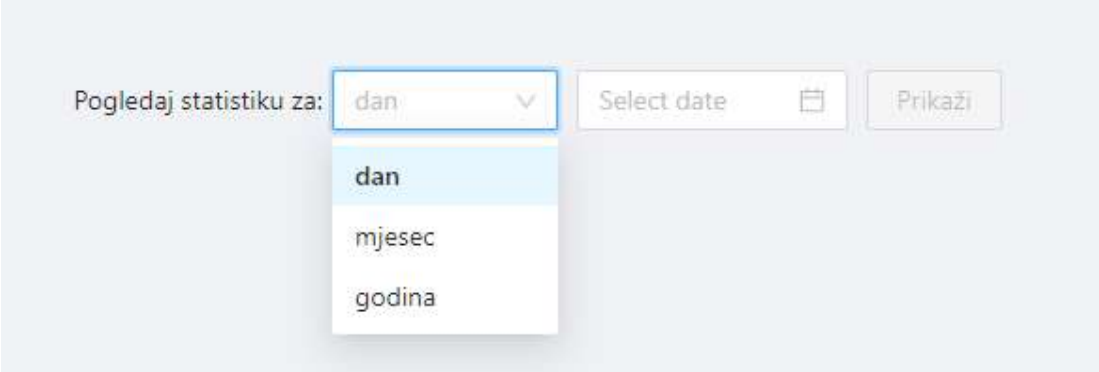
Slika 4.10. Skočni prozor s detaljima kategorije

Mjesečna statistika nudi isti prikaz podataka kao dnevna samo što se bira mjesec a ne dan. Periodna statistika nudi izbor vremenskog perioda od dana do dana te se dobije ista statistika kao za dnevnu i mjesečnu.



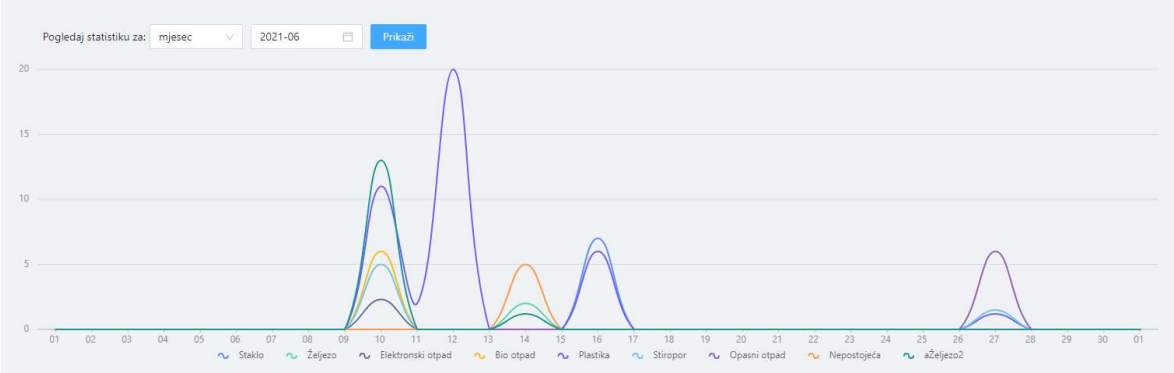
Slika 4.11. Izbor mjeseca i razdoblja

Klikom na „Grafički prikaz“ iz navigacije dobivamo mogućnost biranja dan, mjesec, godina te u skladu s tim se mijenja izbornik za odabir datuma kao što je prikazano na slici 4.12.



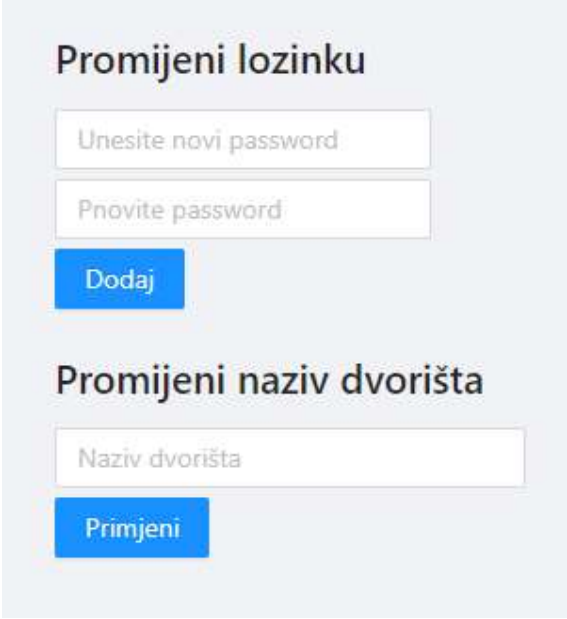
Slika 4.12. Odabir vremena za grafički prikaz

Nakon što se odabere vrijeme za koje se želi prikazati grafička statistika klikom na „prikaži“ iscrtava se graf kao na slici 4.13. na y osi je težina u kilogramima, a x os je promjenjiva u ovisnosti odabere li se dan, mjesec ili godina. Pri odabiru dana na x osi su sati u danu od 5 ujutro do 22 navečer što je pogodno za određivanje u kojem dijelu dana se najviše otpada donosi. Ako se odabere mjesec tada su na x osi prikazani dani u mjesecu od prvog dana tekućeg mjeseca do prvog dana idućeg mjeseca. Tako se može promatrati ima li utjecaja početak ili kraj mjeseca na količine recikliranog otpada. Ako se odabere godina tada x os predstavlja mjesece u godini što omogućuje stvaranje poveznice između količina recikliranog otpada i godišnjih doba.



Slika 4.13. Grafička statistika za mjesec

Zadnja opcija za odabir prije „Odjave“ je „Postavke“ gdje korisnik može mijenjati lozinku i naziv dvorišta kao što je prikazano na slici 4.14.



Slika 4.14. Mijenjanje postavki

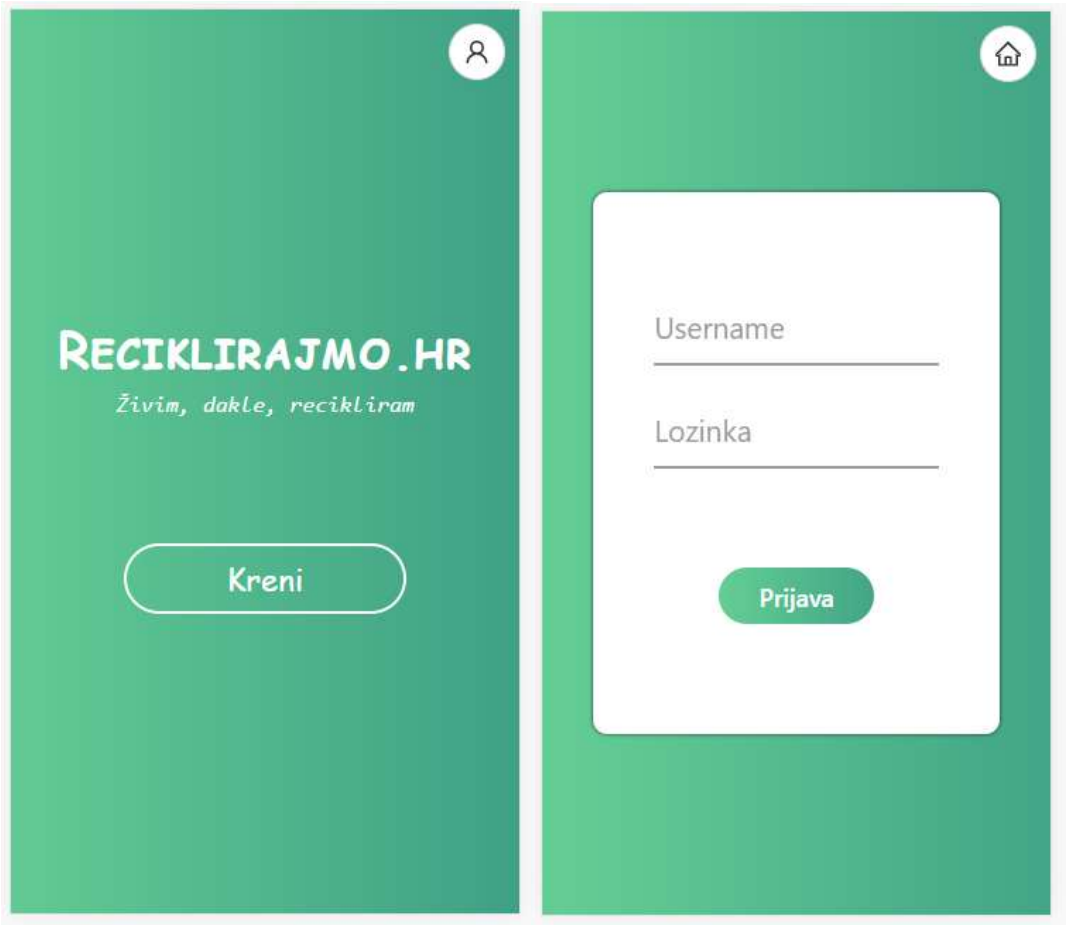
Ako na početnoj stranici umjesto stranice za prijavu korisnik klikne „kreni“ dobiva prikaz u kojem unosom svog OIB-a i odabirom godine dobije grafički prikaz recikliranog otpada koje je on osobno predao na zbrinjavanje kao što je prikazano na slici 4.15.

Slika 4.15. Prikaz statistike za osobu

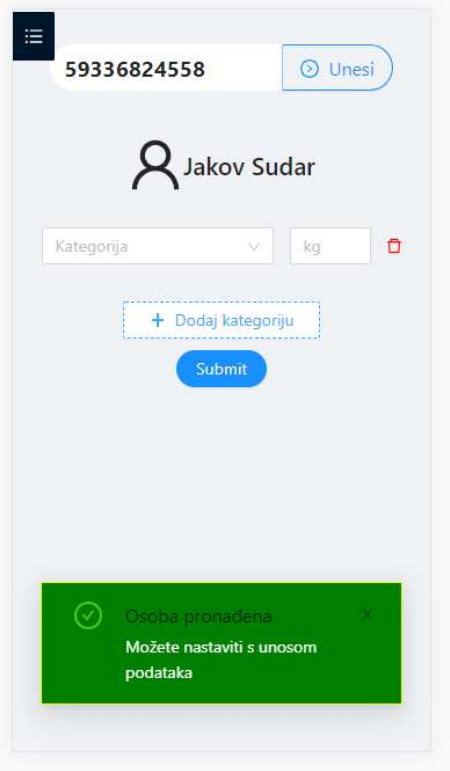
Taj dio aplikacije namijenjen je svim ostalim ljudima koji nisu registrirani u aplikaciju ali žele vidjeti koliko su otpada reciklirali.

4.1 Responzivnost

Sve komponente osim grafičkog prikaza su potpuno prilagođene mobilnim uređajima. U nastavku slijede slike iz prošlog poglavlja ali snimljene na mobilnom uređaju.

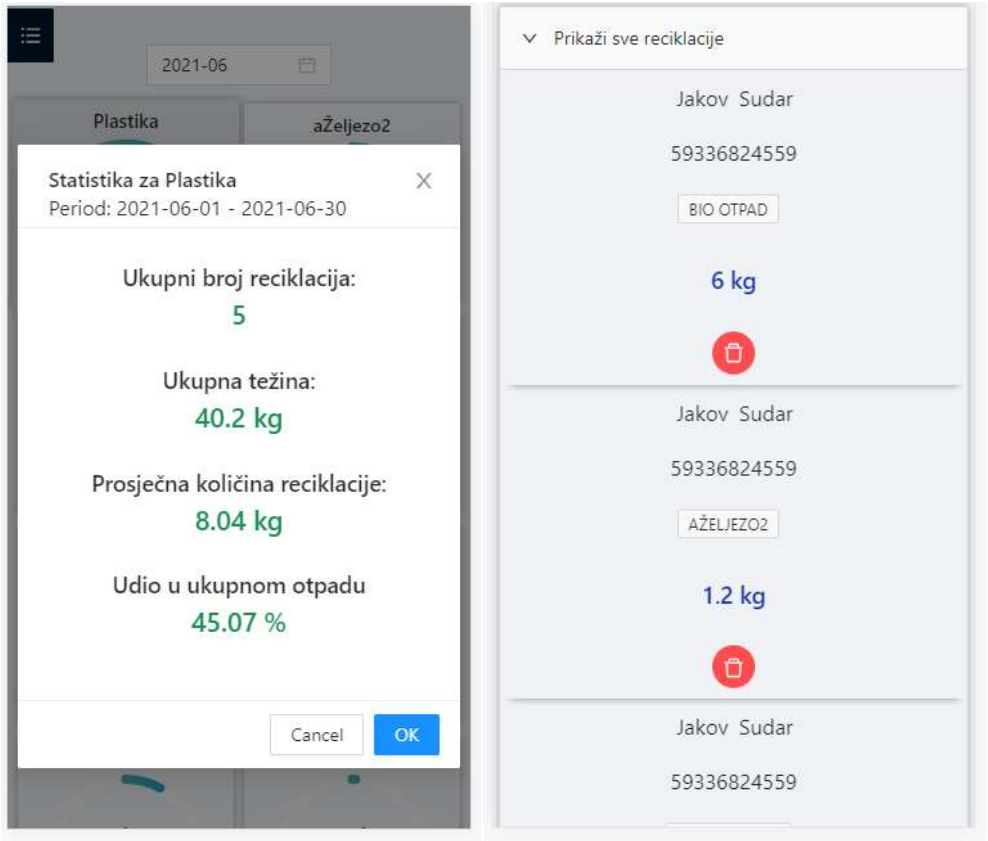


Slika 4.16. Početna stranica i stranica za prijavu



Slika 4.17. Forma za unos zbrinutog otpada

Slika 4.18. Prikazi statistika



Slika 4.19. Prikaz skočnog prozora i tablice

5. ZAKLJUČAK

U ovom diplomskom radu opisan je postupak kreiranja web aplikacije „Upravljanje reciklažnim dvorištem“. Opisane su sve korištene tehnologije kao i najbitniji programski dijelovi koji povezani u cjelinu čine cjelokupan sustav ove aplikacije. Aplikacija je podijeljena na dva dijela, serverski dio i klijentski dio. Serverski dio pruža sve potrebne krajnje točke za realizaciju aplikacije. Aplikacija je osigurana pomoću Spring Security paketa koji ne dozvoljava neautoriziranim korisnicima pristup bilo kojim resursima. Klijentski dio pruža korisničko sučelje za sve potrebne radnje u aplikaciji. Klijentski dio je obogaćen animacijama te sve skupa izgleda fluidno. Velika prednost je responzivnost aplikacije jer se na tako može koristiti putem mobilnog uređaja na terenu. Aplikacija je u prvom planu namijenjena reciklažnim dvorištima kako bi mogli pratiti količine zbrinutog otpada i pregledavati određene statistike koje bi im pomogle u donošenju daljnjih odluka prilikom poslovanja reciklažnog dvorišta. Osim njima namijenjena je i svakoj osobi koja reciklira otpad tako da može vidjeti napismeno na jednom mjestu ukupne količine otpada što može biti i dodatna motivacija i poticaj za recikliranje otpada čemu svi trebamo težiti. Aplikacija je otvorena za razna proširenja. Jedno takvo proširenje bila bi toplinska karta koja bi prikazivala kartu Republike Hrvatske i količinu recikliranog otpada po glavi stanovnika svakog pojedinog grada što bi uvelo natjecateljski duh u recikliranje i želju za titulom najboljeg grada po recikliranju.

LITERATURA

[1] – Narodne novine - Zakon o izmjenama Zakona o održivom gospodarenju otpadom - Članak 19. - 30.siječnja 2019.

[2] – Očevidnik o reciklažnim dvorištima https://mingor.gov.hr/o-ministarstvu-1065/djelokrug-4925/otpad/ocevidnici/1283 (lipanj 2021.)

[3] – PYPL Popularity of programming langugage https://pypl.github.io/PYPL.html (lipanj 2021.)

[4] – Spring Boot https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm (lipanj 2021.)

[5] – Spring Initializr alat https://start.spring.io/ (lipanj 2021.)

[6] – Baedlung, ManyToMany https://www.baeldung.com/hibernate-many-to-many (lipanj 2021.)

[7] – JWT token dokumentacija https://jwt.io/ (lipanj 2021.)

[8] – Ant Design https://ant.design/ (lipanj 2021.)

[9] – Eclipse službena stranica https://www.eclipse.org/ (lipanj 2021.)

[10] – Visual Studio Code https://code.visualstudio.com/ (lipanj 2021.)

SAŽETAK

Potrebna znanja za izradu diplomskog rada su poznavanje React biblioteke, HTML, CSS, JavaScript-a, znanje Java programskog jezika, poznavanje Spring Boot okvira te znanja baza podataka. Prije svega dizajnirana je baza podataka u koju su se spremali svi podaci korišteni u aplikaciji. Aplikacija rješava problem praćenja količina zbrinutog otpada u reciklažnom dvorištu stoga je kreirano korisničko sučelje koje omogućuje upisivanje osobe koja je predala otpad na zbrinjavanje te kategorije i količine otpada. Aplikacija omogućuje pregled raznih statistika koje olakšavaju donošenje poslovnih odluka kao što su radni dani, godišnji odmori, veličine kontejnera za pojedine kategorije i slično. Administrator ima mogućnost registrirati reciklažna dvorišta te dodavati nove gradove koji još nisu spremljeni u bazu. Svi ostali ljudi mogu aplikaciju koristiti tako da upišu svoj OIB te dobiju informacije o količinama otpada koje su osobno predali.

Ključne riječi: Hibernate,Java, PostgreSQL, React, Spring Boot

ABSTRACT

Title: Web application for managing recycling yard

Necessary knowledge for the preparation of the thesisi are knowledge of the React library, HTML, CSS, JavaScript, knowledge of the Java programming language, knowledge of the Spring Boot framework and knowledge of databases. First of all, a database was designed in which all the dana used in the application where stored. The application solves the problem of monitoring the amount of waste disposed of in the recycling yard, so a user interface has been cerated that allows you to register the person who recycled waste also as category and weight of that waste. The application allows you to view various statistics that faciliate business decisions such as working day, vacations, container sizes for certain categories. The administrator has the ability to register recycling yards and add new cities that have not yet been saved to the database, All other people can use the application by entering their ID number and getting information about amount of waste they have personally recycled.

Key words: Hibernate,Java, PostgreSQL, React, Spring Boot

ŽIVOTOPIS

Jakov Sudar rođen je 02.07.1997. u Vinkovcima. Nakon završene osnovne škole upisuje Tehničku školu Ruđera Boškovića Vinkovci u Vinkovcima. Kao najbolji učenik generacije dobiva izravan upis na Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek te odabire smjer računarstvo. Bio je član studentskog zbora kao i studentskog ogranka IEEE. Na temelju prosjeka ocjena postaje stipendist tvrtke Comping d.o.o. Nakon uspješnog završetka preddiplomskog studija upisuje diplomski studij računarstva, smjer Programsko inženjerstvo na istom fakultetu. Odradio je studentske prakse u Mono te u FINA nakon čega se kao student zaposlio u FINA kao full stack programer. Na diplomskom studiju dobiva dekanovo priznanje za uspješnost u studiranju.

PRILOZI

Na CD-u priloženom uz Diplomski rad nalaze se dokumenti:

DiplomskiRadJakovSudar.pdf

DiplomskiRadJakovSudar.doc

WebAplikacija.zip