### Univerzitet u Beogradu Fakultet organizacionih nauka

**Katedra za softversko inženjerstvo**



#### Napredne Java tehnologije

**Aplikacija za online cvećaru u SpringBoot-u i React-u**

Mentor: prof. Ilija Antović Student: Anđela Bokun 2022/0057

Beograd, 2025.

Sadržaj

[Uvod 3](#_Toc212016768)

[1. Opis aplikacije 4](#_Toc212016769)

[2. Tehnologije korišćene za izradu aplikacije 5](#_Toc212016770)

[2.1 Spring Boot 5](#_Toc212016771)

[2.2 React 7](#_Toc212016772)

[3. Slučajevi korišćenja 9](#_Toc212016773)

[SK1- Kreiraj narudžbinu 9](#_Toc212016774)

[SK2- Kreiraj proizvod 10](#_Toc212016775)

[SK3- Obriši proizvod 11](#_Toc212016776)

[SK4- Prijavi cvećar 12](#_Toc212016777)

[SK5- Ubaci kategoriju 13](#_Toc212016778)

[4. Projektovanje skladišta podataka 14](#_Toc212016779)

[5. Korisničko uputstvo 15](#_Toc212016780)

[6. Arhitektura projekta 21](#_Toc212016781)

[7. Objašnjenje frontend koda 22](#_Toc212016782)

[App.js 23](#_Toc212016783)

[Početna stranica 26](#_Toc212016784)

[Stranica za registraciju 31](#_Toc212016785)

[Stranica za login 34](#_Toc212016786)

[Stranica za proizvode – cvećar 37](#_Toc212016787)

[Stranica za narudžbine – cvećar 44](#_Toc212016788)

[Stranice kojima pristupa Korisnik - kategorije, proizvodi 48](#_Toc212016789)

[Stranica za kreiranje narudžbine – korisnik 52](#_Toc212016790)

[Komponente NavBar I Footer 56](#_Toc212016791)

[Axios - povezivanje sa backendom 60](#_Toc212016792)

[Github 61](#_Toc212016793)

[Zaključak 62](#_Toc212016794)

[Literatura 63](#_Toc212016795)

# Uvod

Ova aplikacija prikazuje razvoj i funkcionalnosti web aplikacije za e-cvećaru (online prodaju cvetnih proizvoda), izrađene u Spring Boot-u na serverskoj strani i u React-u na klijentskoj strani. Aplikacija podržava dve glavne uloge – Cvećar (Florist) i Korisnik (User) – sa jasno razdvojenim odgovornostima. Korisnik ima pristup javnom katalogu kategorija po kojima su raspoređeni proizvodi (npr. rezano cveće, saksijsko cveće, aranžmani), kao i svim proizvodima za odabranu kategoriju. Zatim koristi funkcionalnost korpe (Shopping Cart), koja omogućava dodavanje, uklanjanje i promenu količine proizvoda. Kreira narudžbinu, koja podržava više stavki (proizvoda), uz automatski proračun ukupnog iznosa pre slanja. Cvećar (Florist) ima zaseban dashboard za upravljanje narudžbinama (npr. praćenje i ažuriranje statusa narudžbina: CREATED, CONFIRMED, COMPLETED, CANCELED). Vodi evidenciju i sprovodi CRUD operacije nad cvećem (proizvodima) i kategorijama. Backend je organizovan po standardnoj troslojnoj arhitekturi (kontroleri – servisi – repozitorijumi). Implementirano je mapiranje DTO/entitet sloja radi sigurnog i kontrolisanog prenosa podataka. Podaci se trajno čuvaju u relacinoj bazi. Autentikacija i autorizacija realizovane su pomoću JWT tokena. Korisnik nakon uspešne prijave dobija token, koji se automatski prilaže uz sve zaštićene API zahteve, a na osnovu tokena se korisnik usmerava na odgovarajući deo sistema i dobija ovlašćenja u skladu sa svojom ulogom.

# Opis aplikacije

E-cvećara je web aplikacija sa dve uloge: **Cvećar** i **Korisnik**. Korisnik kada se uloguje može da pristupi proizvodima koji su podeljeni na kategorije. Nakon odabira kategorije prelazi na stranicu gde se nalaze svi proizvodi u ponudi za tačno odabranu kategoriju. Takođe korisnik ima pristup korpi u kojoj se nalaze svi odabrani proizvodi sa tačno navedenim iznosom za plaćanje. Cvećar ima pristup stranicama na kojima se nalaze sve kategorije i svi proizvodi. Njih može da ažurira, dodaje nove ili briše postojeće. Takođe cvećar ima pristup stranici na kojoj se nalaze sve narudžbine, čiji status može da menja.

Aplikacija je podeljena na React klijent (čiste, responzivne tabele i forme) i Spring Boot REST backend. Podaci se čuvaju u bazi (npr. MySQL), validacija se radi na klijentu i serveru, a autentikacija vodi korisnika na odgovarajući dashboard po ulozi. API rute pokrivaju CRUD nad svim entitetima, dok je UI prilagođen brzom unosu i preglednom radu cvećara i korisnika.

# Tehnologije korišćene za izradu aplikacije

U ovom poglavlju će biti dat kratak pregled tehnologija koje su korišćene prilikom

izrade aplikacije.

### Spring Boot

Spring Boot je okvir otvorenog koda zasnovan na Spring ekosistemu koji je namenjen razvoju serverskih Java aplikacija. Njegova osnovna ideja je da smanji početnu konfiguraciju i ubrza pokretanje projekta, tako što obezbeđuje podrazumevana podešavanja i konzistentnu organizaciju koda. Zahvaljujući tome, vreme od definisanja zahteva do prvog funkcionalnog prototipa je kraće, a nove funkcionalnosti se dodaju uz manje „sistemske“ pripreme.

Jedna od ključnih karakteristika je autokonfiguracija. Na osnovu prisutnih biblioteka u projektu, okvir automatski registruje potrebne komponente u aplikacionom kontekstu, pa se izbegava ručno deklarisanje velikog broja bean-ova. Rezultat je stabilniji projekat sa manje konflikata i jednostavnije upravljanje zavisnostima.

Aplikacija se najčešće pokreće sa ugradjenim serverom (Tomcat, Jetty ili Undertow), što znači da se sve isporučuje kao samostalni izvršni JAR. Ovakav pristup olakšava postavljanje u različitim okruženjima, jer nije potrebna posebna instalacija aplikacionog servera. Samostalni paket doprinosi i reproduktivnosti okruženja, što je važno za automatizovano testiranje i kontinuiranu isporuku.

Podešavanja se obično nalaze u datotekama application.properties ili application.yml, a često se koriste profili, poput „dev“, „test“ i „prod“. Profili omogućavaju da se parametri razlikuju po okruženju (na primer, konekcija ka bazi, nivoi logovanja ili sigurnosne postavke), bez menjanja izvornog koda. Na taj način se zadržava ista logika, dok se operativne razlike kontrolišu kroz konfiguraciju.

Rad sa bazom podataka je u praksi najčešće oslonjen na Spring Data JPA, koji uvodi pristup zasnovan na repozitorijumima. Standardne operacije nad entitetima dobijaju jasan API, a po potrebi se definišu prilagođeni upiti. Pored JPA dostupni su i drugi

pristupi (JDBC, r2dbc i dr.), pa se izbor prilagođava zahtevima projekta. Ovakva fleksibilnost olakšava i migracije između različitih rešenja za skladištenje podataka.

Bezbednost se obično implementira uz Spring Security, koji obezbeđuje mehanizme autentikacije i autorizacije. U kombinaciji sa Spring Boot-om, bezbednosne politike se uvode dosledno kroz konfiguraciju i filtere, pa domen aplikacije ostaje odvojen od bezbednosne infrastrukture. Na taj način je moguće menjati pravila pristupa bez narušavanja poslovne logike i bez većih intervencija u ostatku koda.

Sveukupno, Spring Boot predstavlja zrelu i široko prihvaćenu platformu za izgradnju serverskih Java aplikacija. Kombinuje autokonfiguraciju, ugradjeni server, jasan model konfiguracije i dobru integraciju sa bazama, bezbednošću i nadzorom. Zbog takvog skupa osobina primenljiv je i u manjim monolitnim rešenjima i u arhitekturama sa mikroservisima, a omogućava postepeno širenje i prilagođavanje kako se potrebe sistema razvijaju.

### React

React je JavaScript biblioteka za izgradnju korisničkih interfejsa, nastala u Facebook- u i danas široko korišćena u web razvoju. Osnovna ideja je da se interfejs razbije na manje, ponovo upotrebljive komponente koje se sastavljaju. Umesto direktne manipulacije DOM-om, React održava virtuelni prikaz ekrana i efikasno osvežava samo ono što je promenjeno. Zbog takvog pristupa interfejs deluje brže i lakše se kontroliše kada aplikacija poraste.

Komponente se pišu uz pomoć JSX-a, sintakse koja meša HTML-u slične oznake i JavaScript logiku. Ovakav zapis olakšava razumevanje strukture i ponašanja na jednom mestu, jer se prikaz i potrebni podaci nalaze zajedno. Svaka komponenta prima „props“ (ulazne vrednosti) i može da ima sopstveno stanje („state“) koje čuva trenutne podatke važnije za prikaz. Razdvajanje na male, fokusirane komponente obično vodi kôdu koji se lakše održava i testira.

U novijim projektima najčešće se koriste React Hooks, kao što su useState i useEffect, koji uvode stanje i životni ciklus bez pisanja klasa. Ideja je da se logika vezana za određeno ponašanje (na primer, učitavanje podataka ili upravljanje događajima) grupiše u sopstvene hook-ove i zatim deli između komponenti. Time se izbegava dupliranje, a istovremeno je jasnije gde se koja logika nalazi. Za navigaciju kroz stranice uglavnom se koristi poseban paket, pa je upravljanje rutama odvojeno i pregledno.

Ecosistem oko React-a je veoma bogat. Postoje biblioteke za menadžment stanja (Redux, Zustand, Recoil), rešenja za forme i validaciju, kao i gotove kolekcije komponenti koje ubrzavaju izradu interfejsa. Za realne aplikacije ovo znači da se manje vremena troši na „infrastrukturu“, a više na poslovnu logiku. Sa druge strane, veliki izbor alata ponekad dovodi do neodlučnosti i čestih promena u tehnologijama, pa je korisno na početku projekta jasno dogovoriti standarde i zadržati se na njima.

U pogledu performansi React se oslanja na virtuelni DOM i rekonsilijaciju, što u praksi donosi solidne rezultate za većinu slučajeva. Dodatno, postoje tehnike kao što su

„memoizacija“ komponenti, razdvajanje koda po rutama i odloženo učitavanje, koje mogu da smanje opterećenje i ubrzaju prikaz. Za SEO i brzo prvo učitavanje često se primenjuje serversko renderovanje ili „pre-rendering“ preko okvira kao što su Next.js ili Remix, čime se spajaju pogodnosti React-a sa boljom vidljivošću na pretraživačima.

Prednosti koje se najčešće ističu su ponovna upotreba komponenti, jasna podela odgovornosti, snažna zajednica i veliki broj dodatnih biblioteka. Aplikacije su lakše za proširenje, jer se novi delovi interfejsa uklapaju u postojeći koncept. Kao mana se ponekad navodi složenost izbora u ekosistemu i potreba da se razume nekoliko koncepata pre nego što projekat postane potpuno „tečan“. Ipak, u tipičnim studentskim i komercijalnim projektima React pokazuje dobar odnos između krive učenja i dobijenih rezultata, pa se često koristi za izradu modernih, responzivnih i interaktivnih web aplikacija.

# Slučajevi korišćenja

U okviru ove aplikacije je kreiran veliki broj slučajeva korišćenja, a ovde će biti izdvojeni i detaljno opisani neki od njih.

## SK1- Kreiraj narudžbinu

##### Naziv SK

Kreiraj narudžbinu **Aktori SK** Korisnik

##### Učesnici SK

Korisnik, korisnički interfejs (klijentski program) i sistem (serverski program) **Preduslovi:** Korisnički interfejs (klijentski program) i sistem (serverski program) su pokrenuti.Korisnik je prijavljen pod svojom šifrom. Sistem prikazuje formu za rad sa nardžbinom.

##### Osnovni scenario SK:

1. Korisnik poziva sistem da kreira narudžbinu. (APSO)
2. Sistem kreira narudžbinu. (SO)
3. Sistem prikazuje korisniku narudžbinu i poruku: “Sistem je kreirao narudžbinu”.

(IA)

1. Zaposleni unosi podatke o narudžbini. (APUSO)
2. Zaposleni kontroliše da li je korektno uneo podatke o narudžbini. (ANSO)
3. Zaposleni poziva sistem da zapamti podatke o narudžbini. (APSO)
4. Sistem pamti podatke o narudžbini. (SO)
5. Sistem prikazuje korisniku poruku: “Sistem je zapamtio narudžbinu.” (IA)

##### Alternativna scenarija:

3.1 Ukoliko sistem ne može da kreira narudžbinu on prikazuje korisniku poruku: “Sistem ne može da kreira narudžbinu”. Prekida se izvršenje scenarija. (IA)

8.1 Ukoliko sistem ne može da zapamti podatke o narudžbini on prikazuje korisniku poruku: “Sistem ne može da zapamti narudžbinu”. (IA)

## SK2- Kreiraj proizvod

**Naziv SK**

Kreiraj proizvod

**Aktori SK** Cvećar **Učesnici SK**

Cvećar, korisnički interfejs (klijentski program) i sistem (serverski program) **Preduslovi:** Korisnički interfejs (klijentski program) i sistem (serverski program) su pokrenuti. Cvećar je prijavljen pod svojom šifrom. Sistem prikazuje formu za rad sa proizvodom.

##### Osnovni scenario SK:

1. Cvećar poziva sistem da kreira proizvod. (APSO)
2. Sistem kreira proizvod. (SO)
3. Sistem prikazuje cvećaru proizvod i poruku: “Sistem je kreirao proizvod”. (IA)
4. Cvećar unosi podatke o proizvodu. (APUSO)
5. Cvećar kontroliše da li je korektno uneo podatke o proizvodu. (ANSO)
6. Cvećar poziva sistem da zapamti podatke o proizvodu. (APSO)
7. Sistem pamti podatke o proizvodu. (SO)
8. Sistem prikazuje cvećaru proizvod i poruku: “Sistem je zapamtio proizvod.” (IA)

##### Alternativna scenarija:

3.1 Ukoliko sistem ne može da kreira proizvod on prikazuje cvećaru poruku: “Sistem ne može da kreira proizvod”. Prekida se izvršenje scenarija. (IA)

8.1 Ukoliko sistem ne može da zapamti podatke o proizvodu on prikazuje cvećaru poruku: “Sistem ne može da zapamti proizvod”. (IA)

## SK3- Obriši proizvod

**Naziv SK** Obriši cvet **Aktori SK** Cvećar **Učesnici SK**

Cvećar, korisnički interfejs (klijentski program) i sistem (serverski program) **Preduslovi:** Korisnički interfejs (klijentski program) i sistem (serverski program) su pokrenuti. Cvećar je prijavljen pod svojom šifrom. Korisnički interfejs prikazuje formu za rad sa proizvodom.

##### Osnovni scenario SK:

1. Cvećar bira proizvod. (APUSO)
2. Cvećar poziva sistem da nađe proizvod. (APSO)
3. Sistem traži proizvod. (SO)
4. Sistem prikazuje cvećaru proizvod i poruku: “Sistem je našao proizvod”. (IA)
5. Cvećar poziva sistem da obriše proizvod. (APSO)
6. Sistem briše proizvod. (SO)
7. Sistem prikazuje cvećaru poruku: “Sistem je obrisao proizvod.” (IA)

##### Alternativna scenarija:

4.1 Ukoliko sistem ne može da nađe proizvod on prikazuje cvećaru poruku: “Sistem ne može da nađe proizvod”. Prekida se izvršenje scenarija. (IA)

7.1 Ukoliko sistem ne može da obriše proizvod on prikazuje cvećaru poruku: “Sistem ne može da obriše proizvod”. (IA)

## SK4- Prijavi cvećar

**Naziv SK** Prijavi cvećar **Aktori SK** Cvećar **Učesnici SK**

Cvećar, korisnički interfejs (klijentski program) i sistem (serverski program) **Preduslovi:** Korisnički interfejs (klijentski program) i sistem (serverski program) su pokrenuti. Korisnički interfejs prikazuje formu za prijavljivanje.

##### Osnovni scenario SK:

1. Cvećarunosi korisničko ime i šifru. (APUSO)
2. Cvećarkontroliše da li je korektno uneo korisničko ime i šifru. (ANSO)
3. Cvećarpoziva sistem da proveri korisničko ime i šifru. (APSO)
4. Sistem proverava korisničko ime i šifru. (SO)
5. Sistem prikazuje cvećaruporuku: “Korisničko ime i šifra su ispravni.” (IA)
6. Korisnički interfejs poziva glavnu formu i meni. (KIPGFM)

##### Alternativna scenarija:

5.1 Ukoliko sistem proverom ustanovi da korisničko ime i/ili šifra nisu ispravni on prikazuje cvećaruporuku: “Korisničko ime i šifra nisu ispravni”. (IA)

6.1 Ukoliko korisnički interfejs ne može da otvori glavnu formu i meni prikazuje cvećaruporuku: “Ne može da se otvori glavna forma i meni”. (NPGFM) **Postuslovi:** Otvorena glavna forma i meni.

## SK5- Ubaci kategoriju

##### Naziv SK

Ubaci kategoriju **Aktori SK** Cvećar **Učesnici SK**

Cvećar, korisnički interfejs (klijentski program) i sistem (serverski program) **Preduslovi:** Korisnički interfejs (klijentski program) i sistem (serverski program) su pokrenuti. Cvećarje prijavljen pod svojom šifrom. Sistem prikazuje formu za rad sa kategorijama.

##### Osnovni scenario SK:

1. Cvećarunosi podatke o kategoriji. (APUSO)
2. Cvećarkontroliše da li je korektno uneo podatke o kategoriji. (ANSO)
3. Cvećarpoziva sistem da zapamti podatke o kategoriji. (APSO)
4. Sistem pamti podatke o kategoriji. (SO)
5. Sistem prikazuje korisniku aranžman i poruku: “Sistem je zapamtio kategoriju.”

(IA)

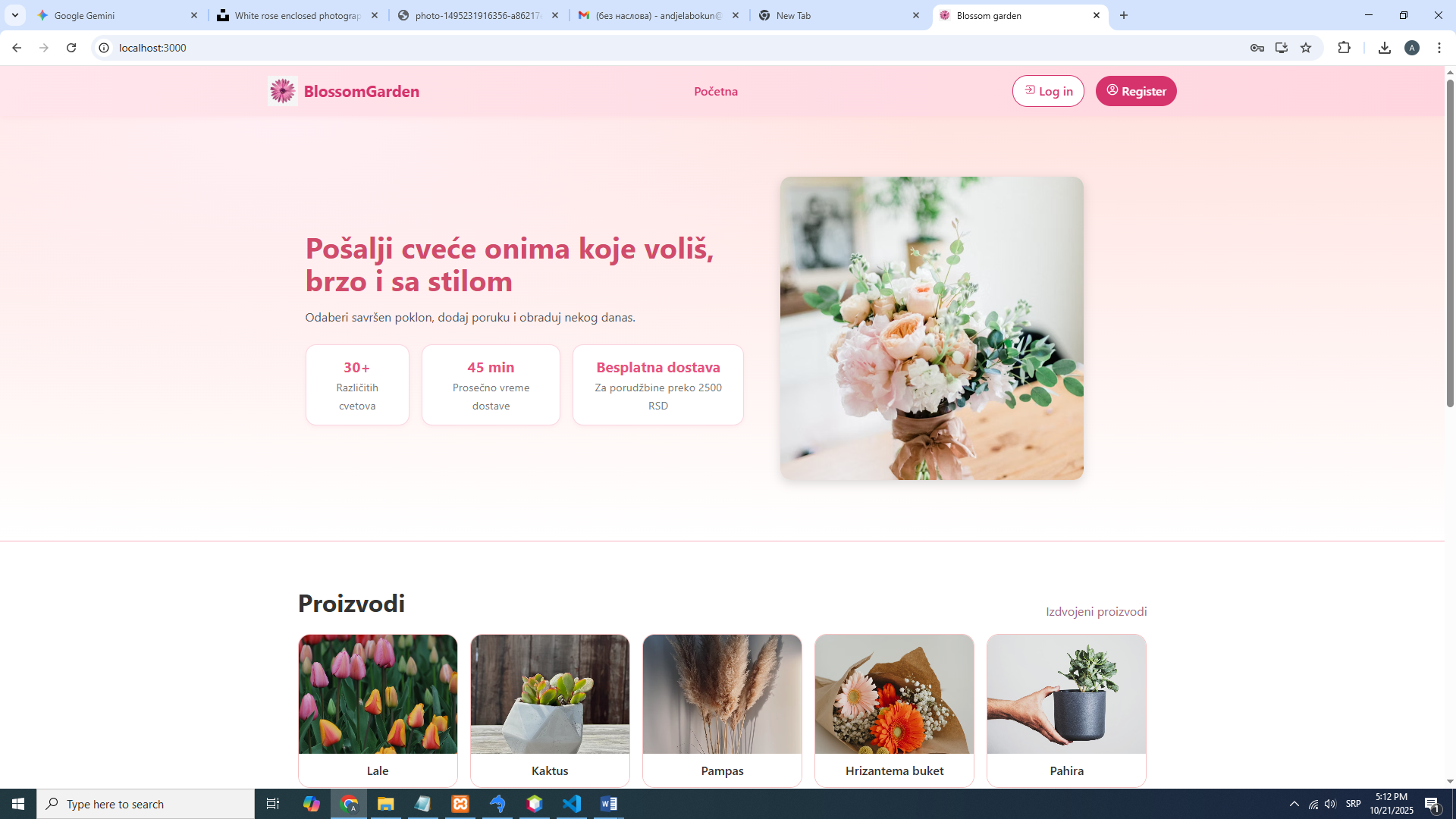
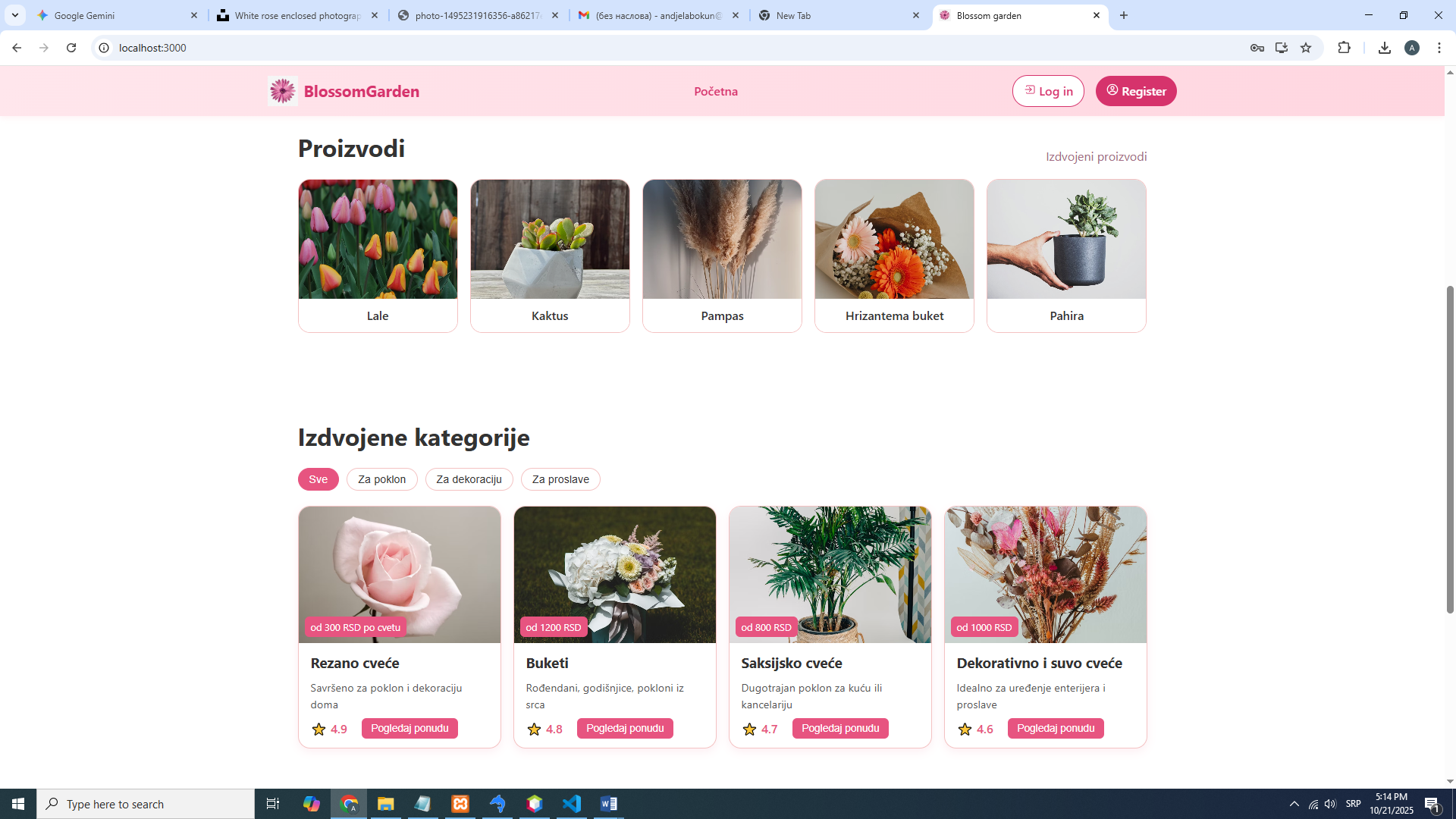
##### Alternativna scenarija:

* 1. Ukoliko sistem ne može da zapamti podatke o aranžmanu on prikazuje cvećaruporuku: “Sistem ne može da zapamti kategoriju”. (IA)

# Projektovanje skladišta podataka

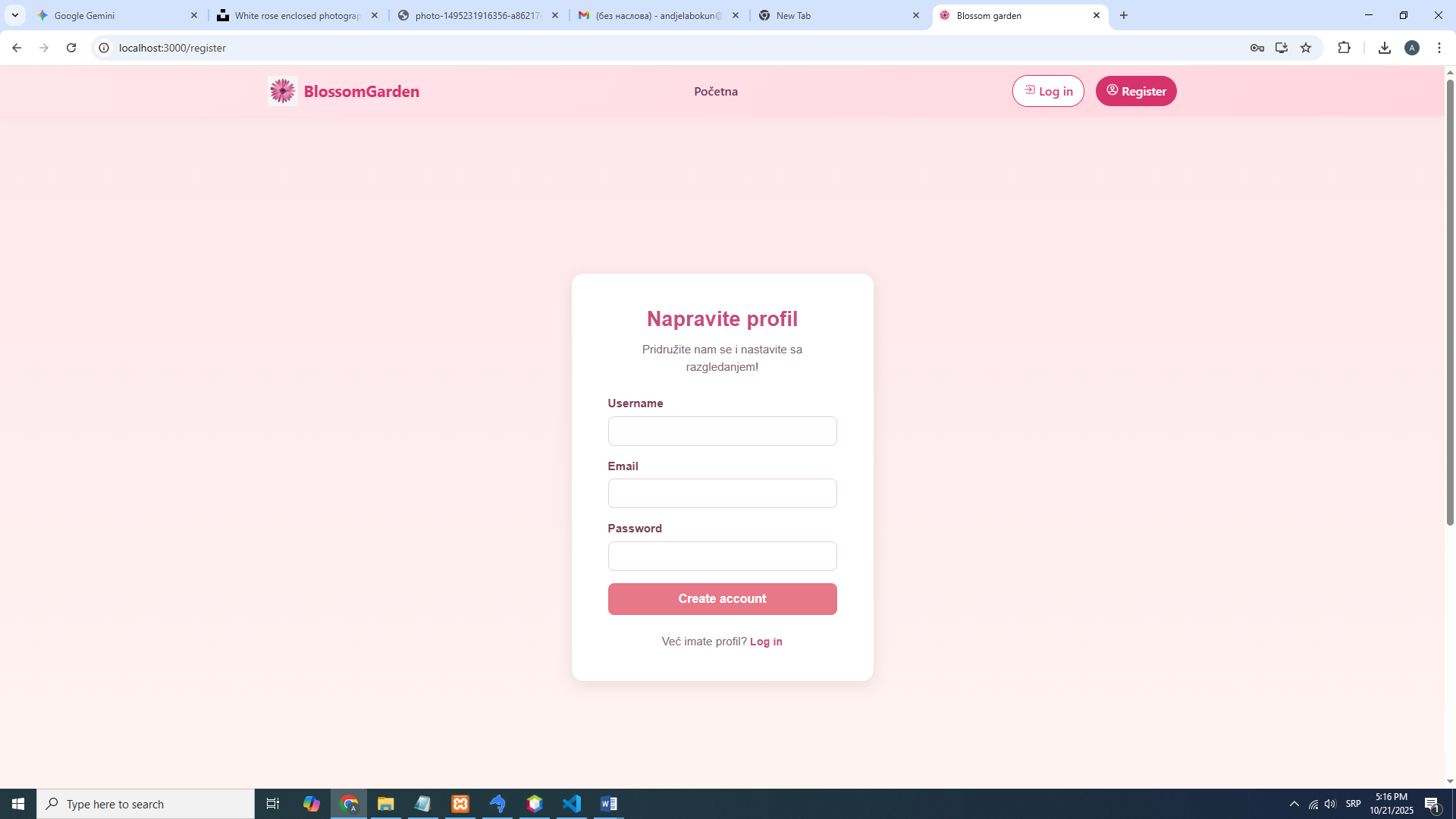
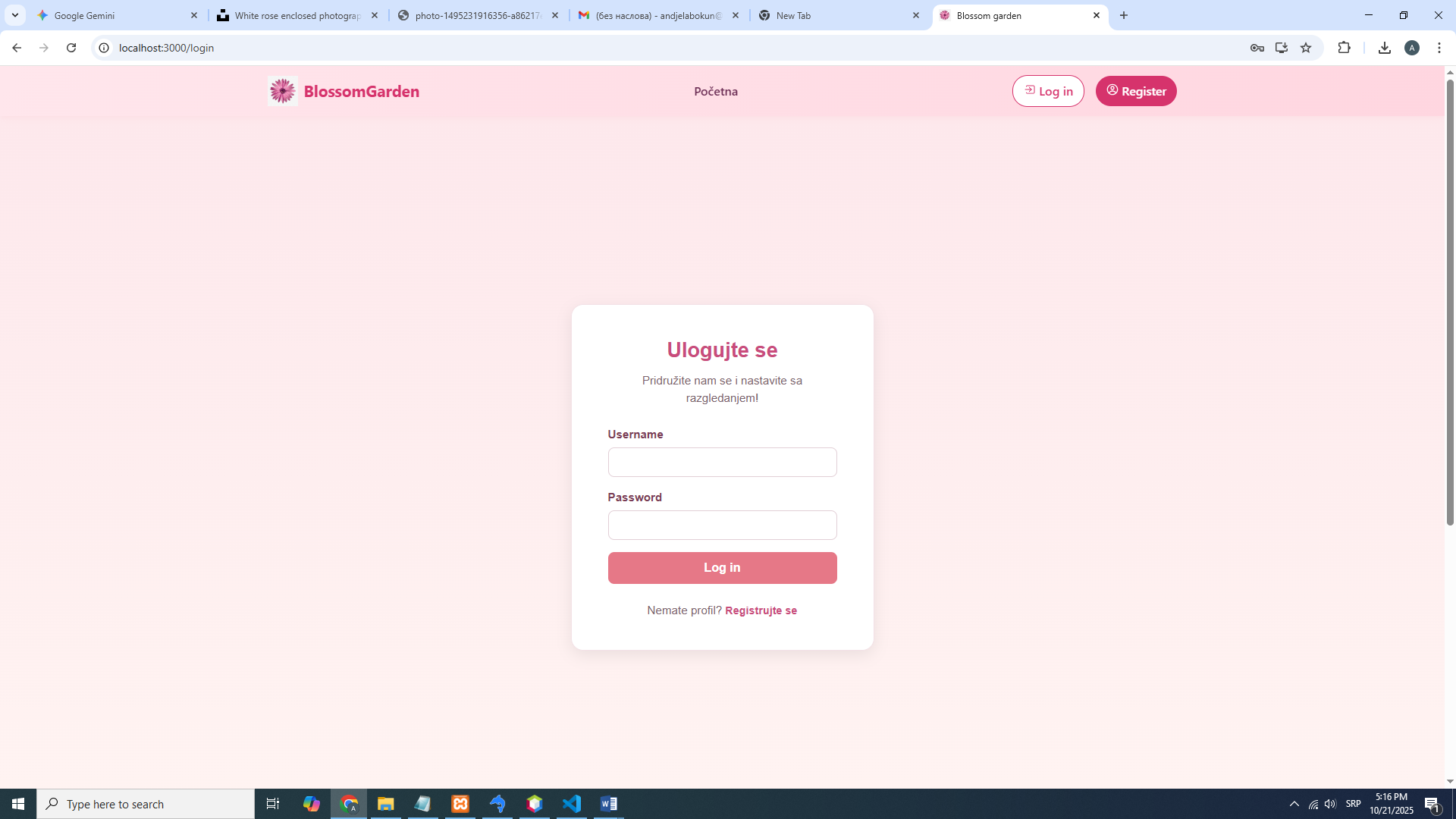
Projektovanje skladišta podataka za ovu aplikaciju zasniva se na normalizovanom modelu: entiteti Zaposleni, Putnik, TipAranzmana i Aranzman čuvaju opisne atribute (npr. ime/prezime, naziv, destinacija), dok Rezervacija predstavlja poslovni događaj koji referencira odgovorni Aranžman i zaposlenog (a opcionalno i glavnog putnika), a StavkaRezervacije razlaže jednu narudžbinu na više stavki sa količinom, cenom, popustom i iznosom. Na ovaj način eliminiše se dupliranje podataka (npr. podaci o putniku postoje jednom), a sve agregacije (ukupno po narudžbini, prodaja po aranžmanu, performanse po agentu) dobijaju se preko JOIN-ova i indeksiranih stranih ključeva. Integritet se obezbeđuje NOT NULL/FOREIGN KEY ograničenjima i kaskadnim brisanjem stavki kada se obriše rezervacija; performanse se postižu indeksima nad FK kolonama i datumima, a iznos stavke se računa kao izvedeno polje i sumira u ukupno na nivou rezervacije pri snimanju.

# Korisničko uputstvo

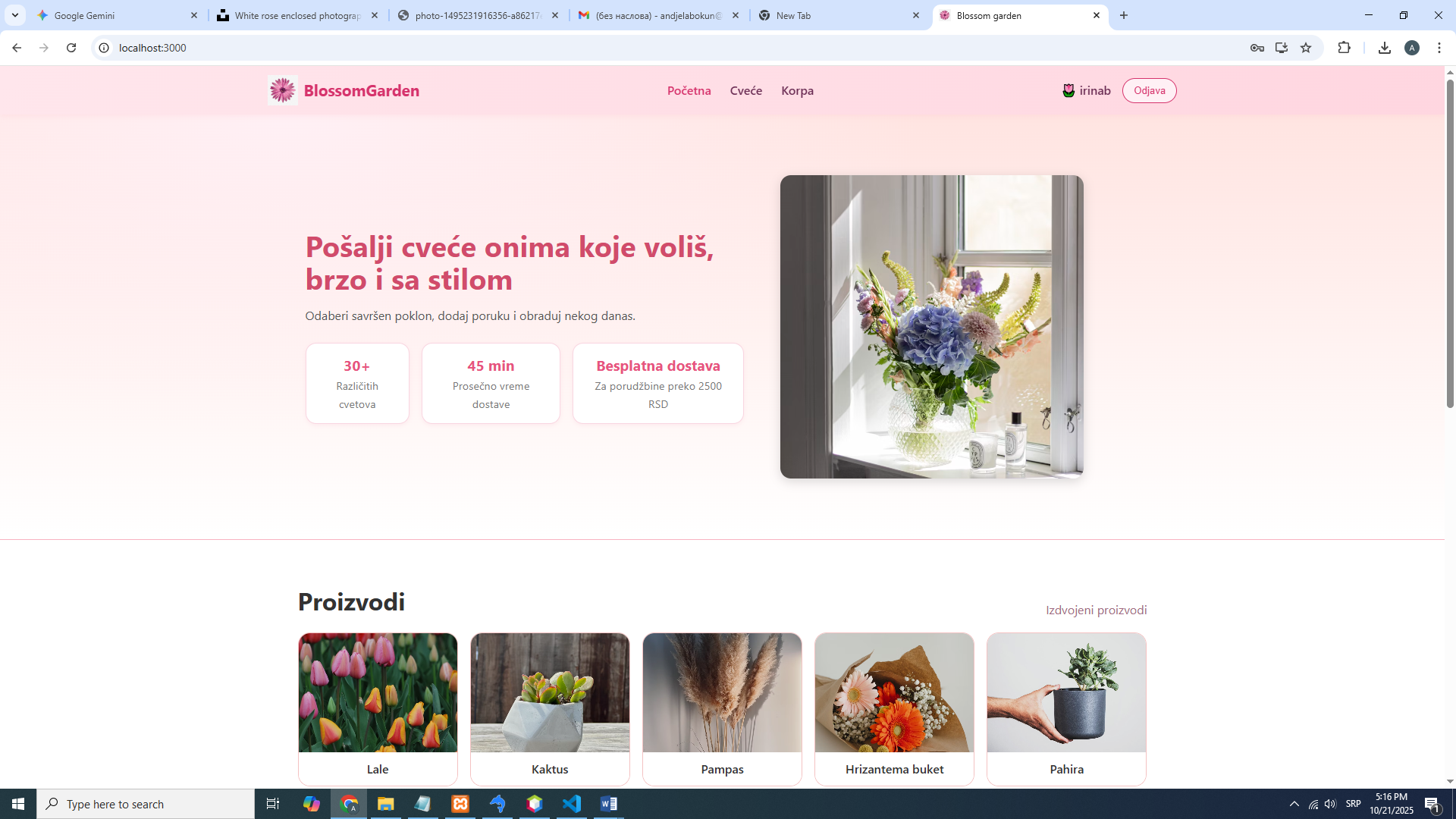
Kada se pokrene aplikacija otvara se početna stranica, kao na slici ispod.

Slika 2 – Početna stranica

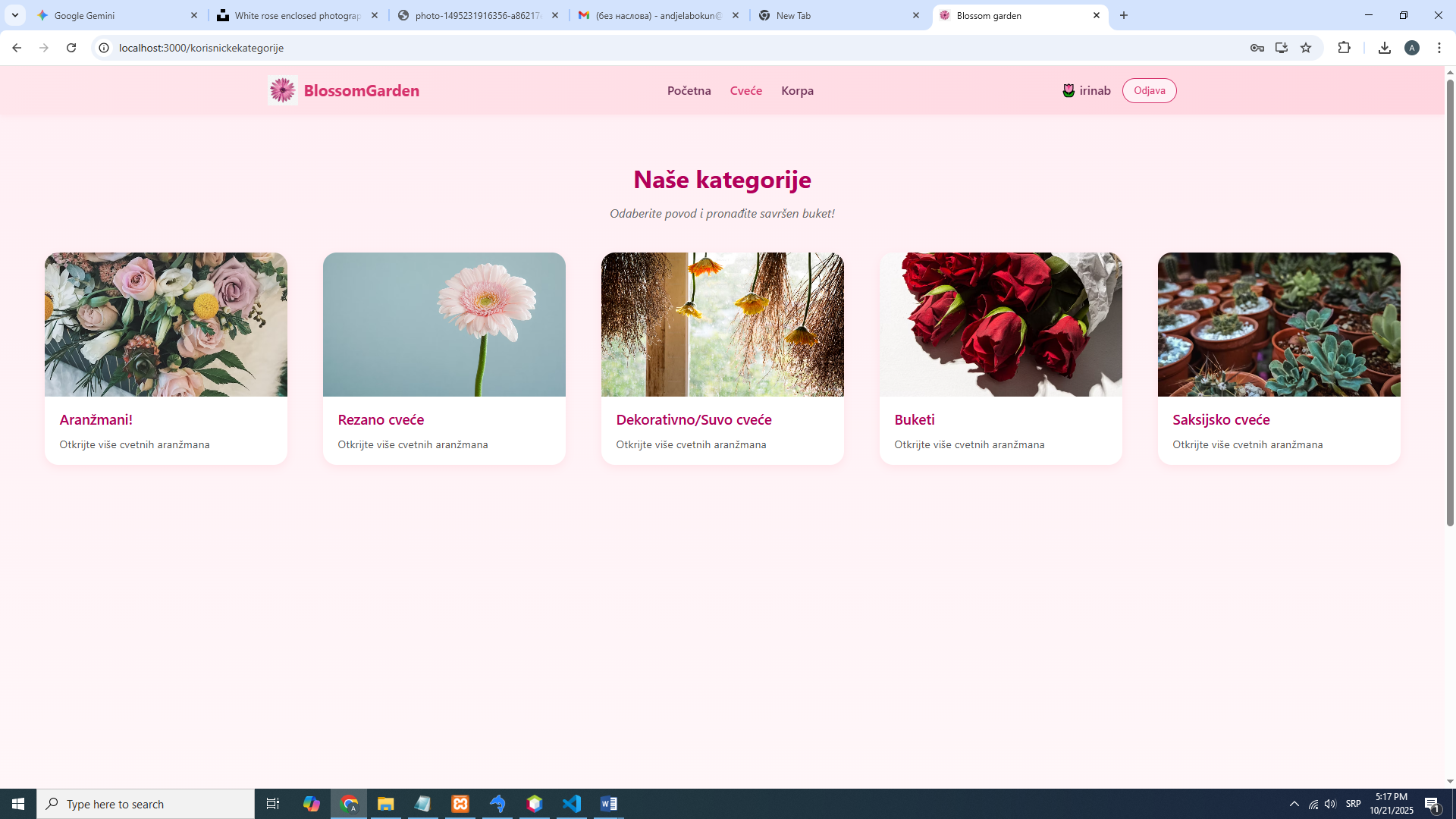
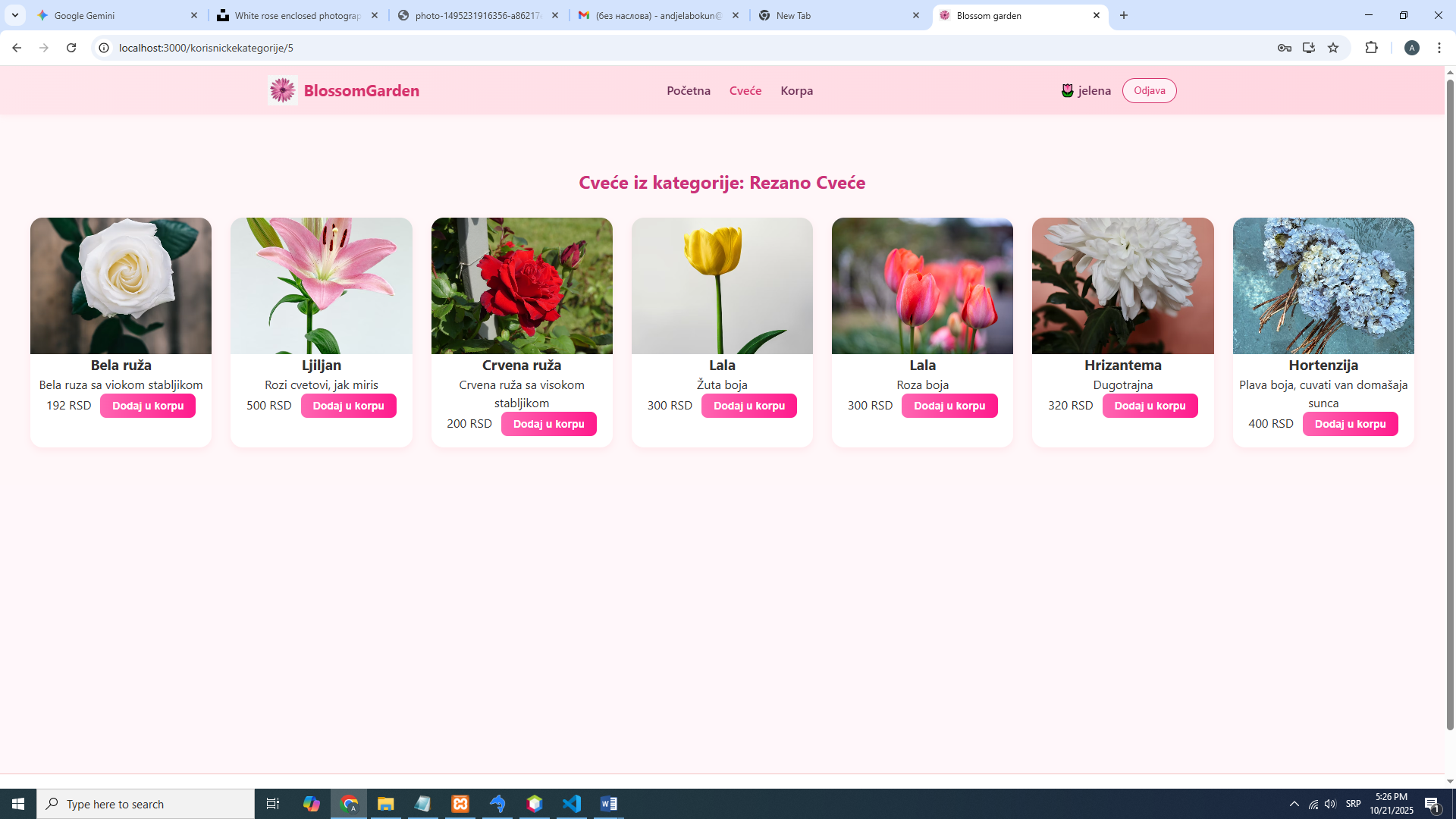
Slika 1 – Početna stranica



*Slika 3, 4 - Kako bi korisnici mogli da koriste aplikaciju, neohpodno je da se registruju ili uloguju.*

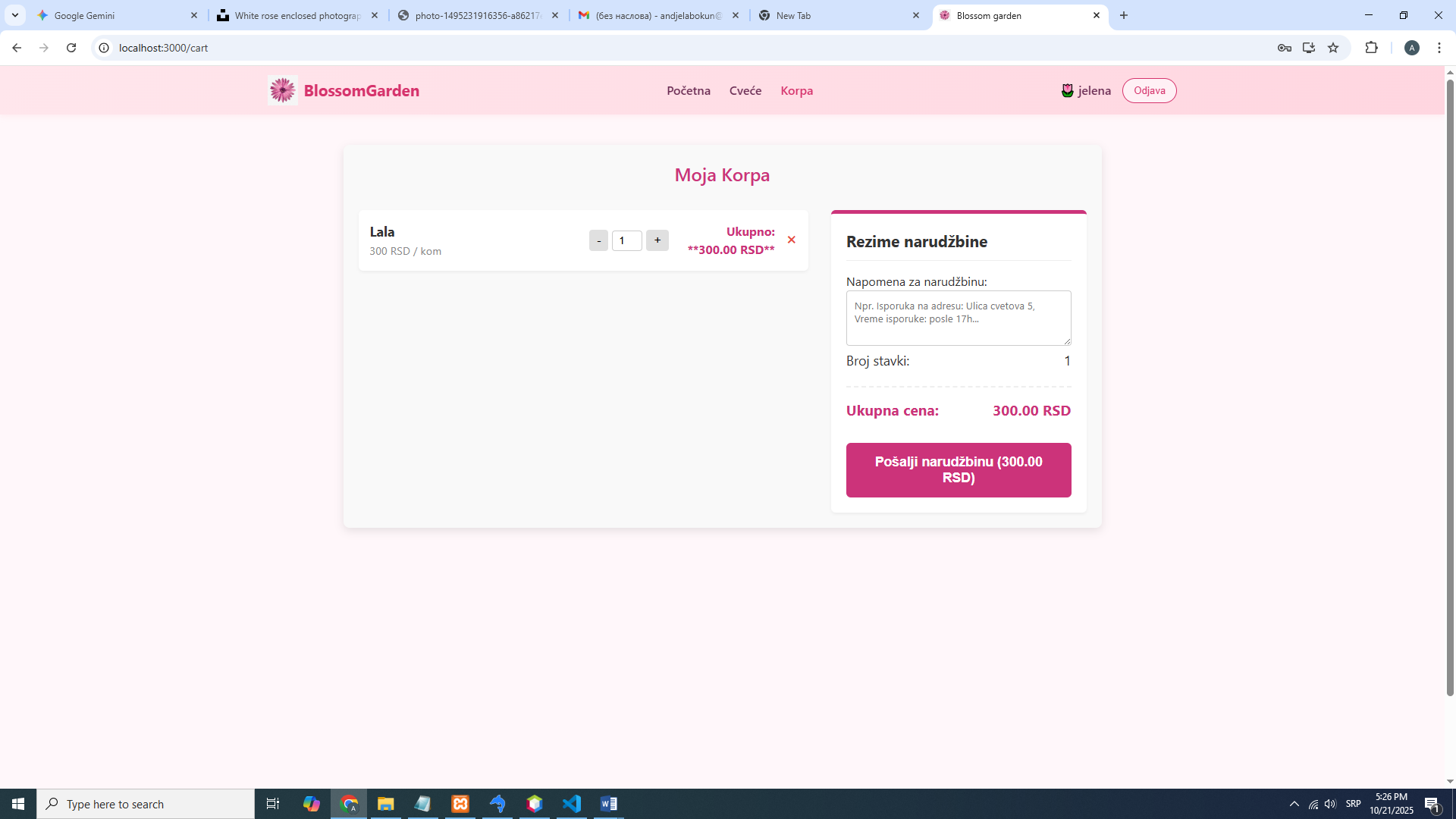


Slika 5 – Početna stranica ulogogvanog korisnika

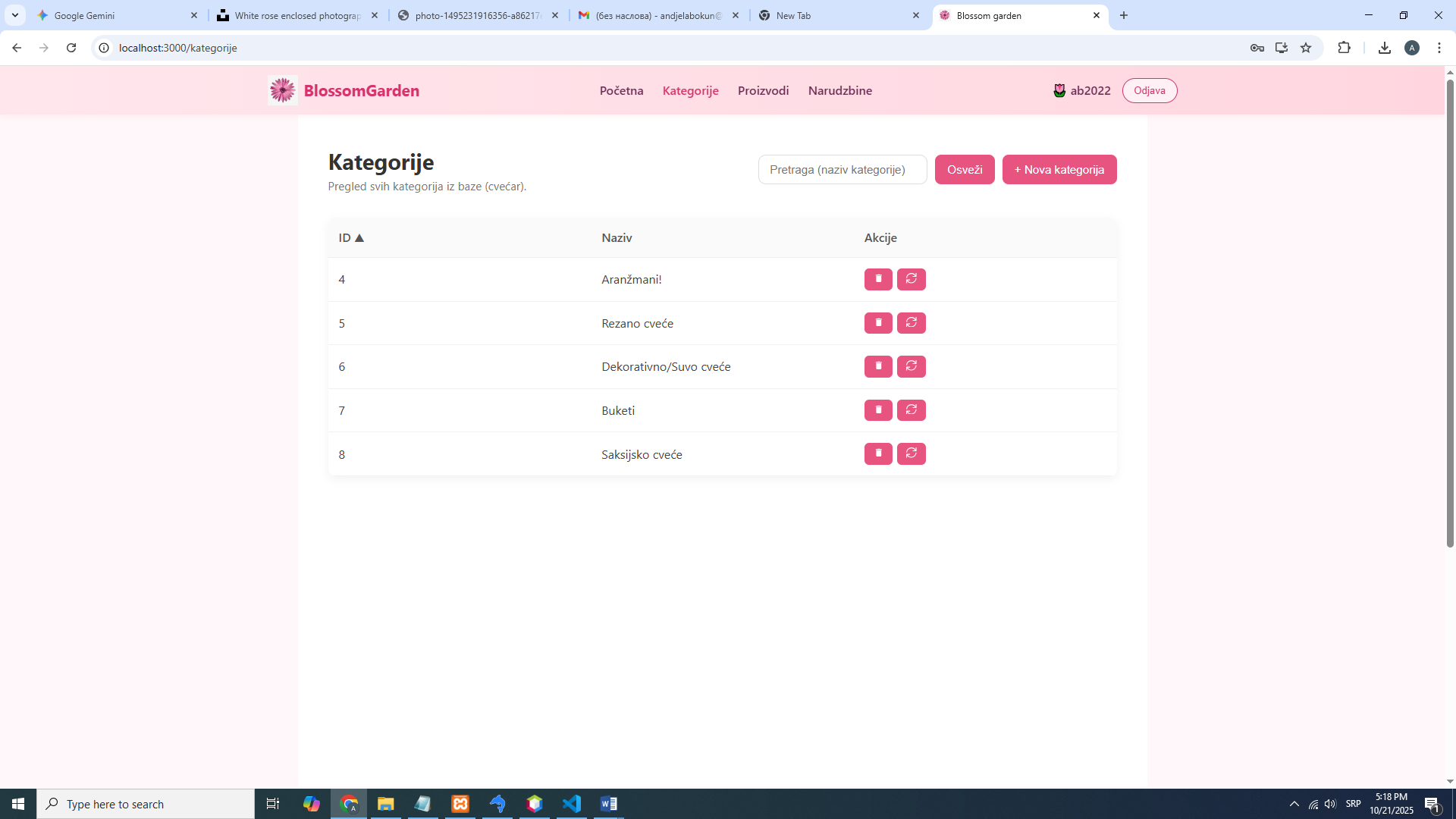


Slika 7 – Proizvodi iz odabrane kategorije

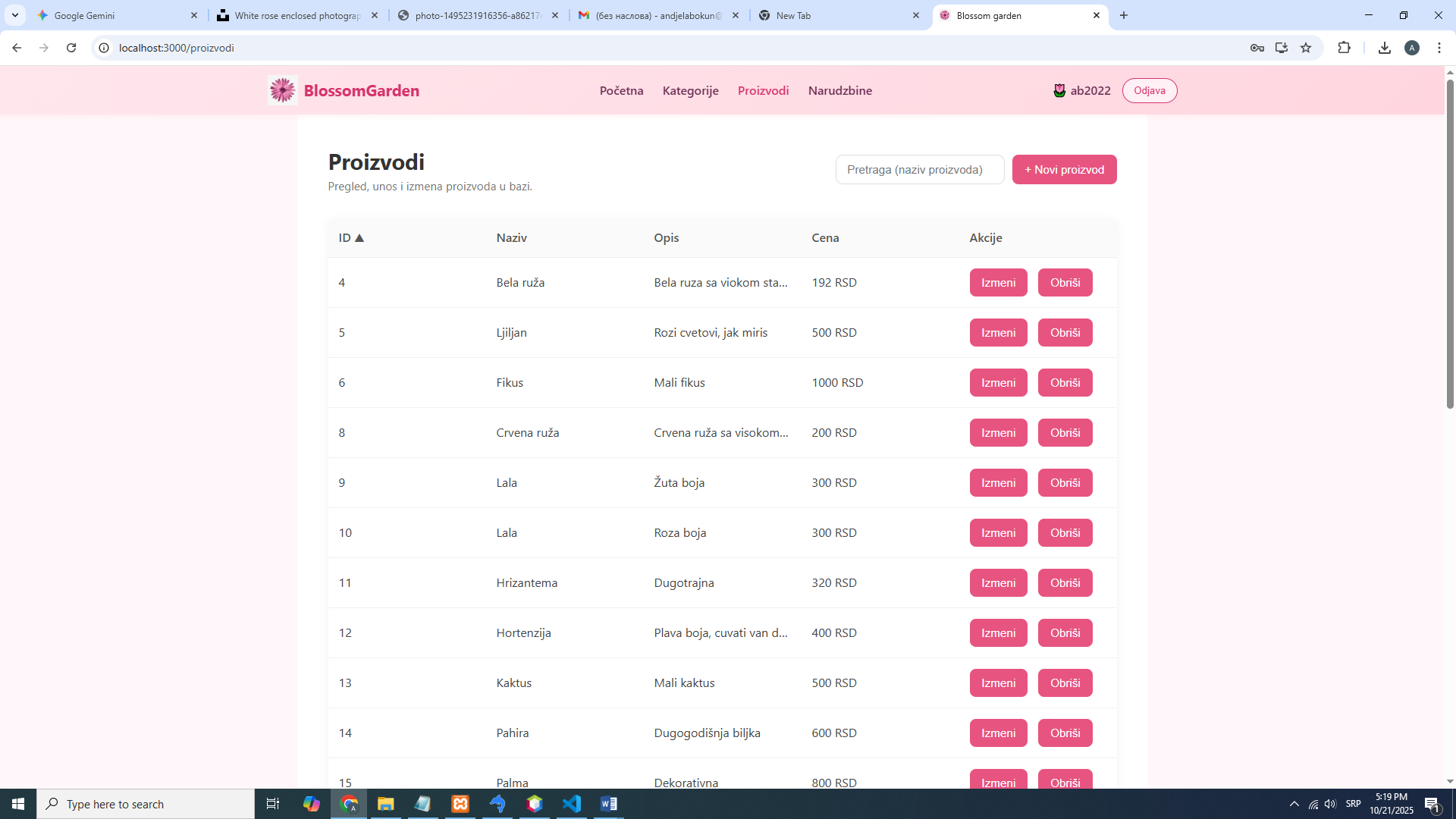
Slika 6 – Kategorije kojima pristupa klijent



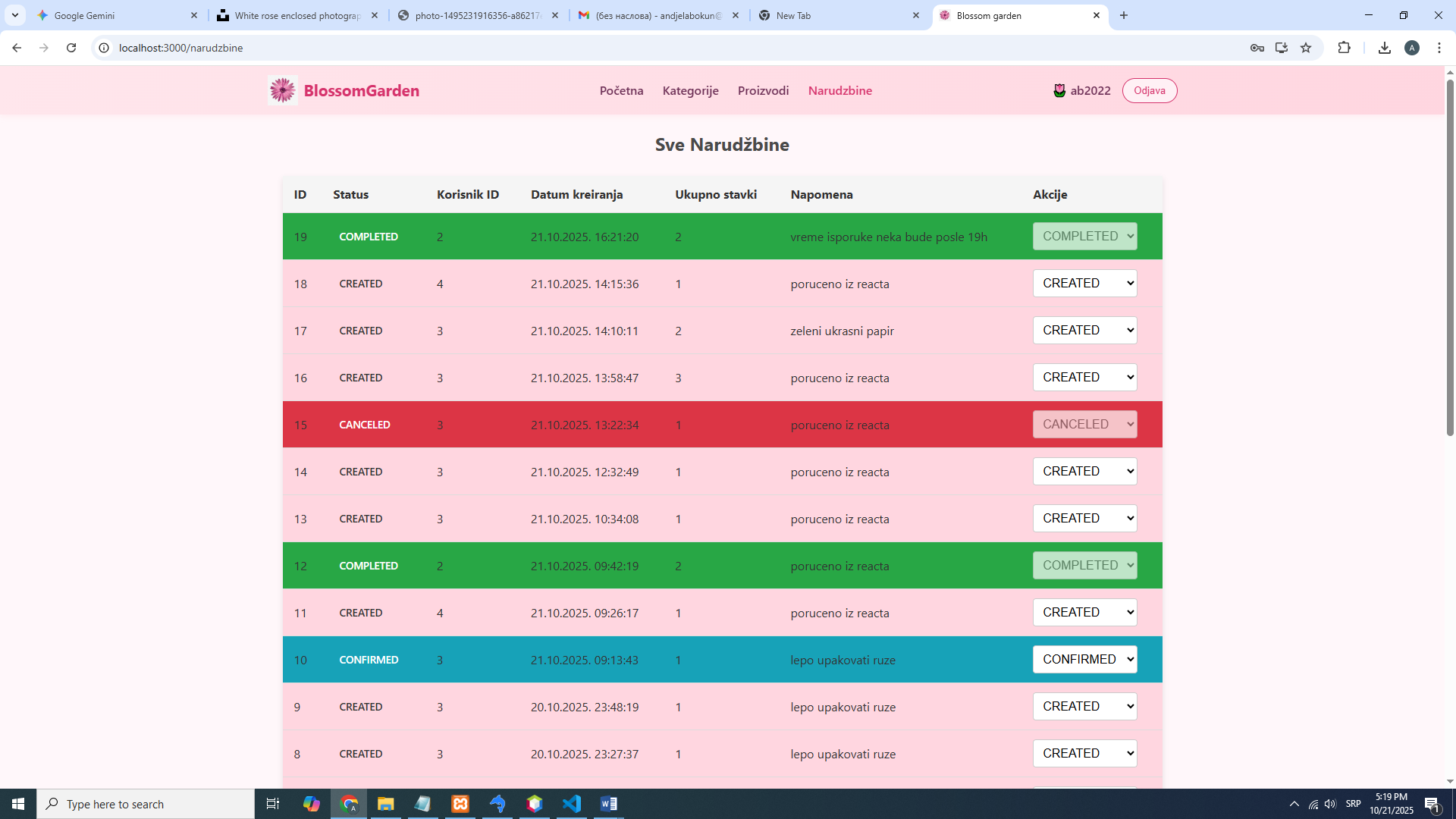
Slika 8 – Korpa nakon dodavanja proizvoda



Slika 9 – Stranica za upravljanje kategorijama - cvećar

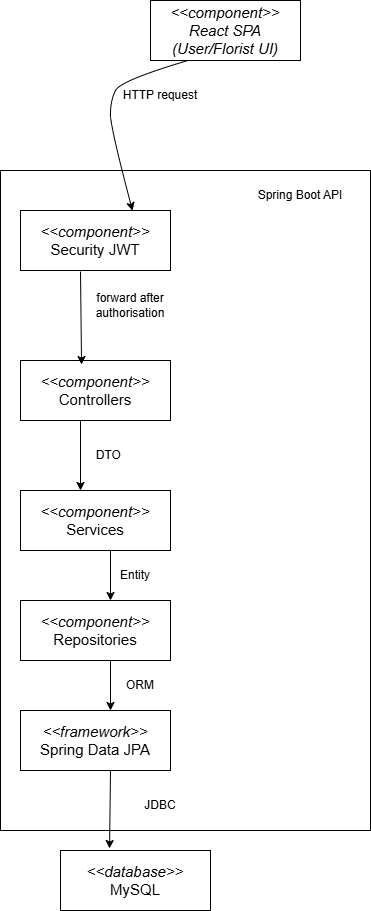


Slika 10 – Stranica za upravljanje proizvodima - cvećar



Slika 11 – Stranica za upravljanje narudžbinama - cvećar

# Arhitektura projekta

Aplikacija je organizovana po standardnom troslojnom obrascu. Na vrhu je web/sigurnosni sloj sa REST kontrolerima i JWT filtrom koji proverava kredencijale i prosleđuje validne zahteve odgovarajućim kontrolerima. Kontroleri su tanki i samo mapiraju HTTP pozive na servisne metode. U servisnom sloju je poslovna logika: validacija podataka, izračunavanje ukupnih iznosa narudžbina i orkestracija nad više repozitorijuma. Servisi rade sa DTO objektima prema klijentu, a preko mapper-a ih prevode u JPA entitete. Repozitorijumi su implementirani kroz Spring Data JPA i zaduženi su isključivo za pristup bazi, čime se postiže jasna separacija odgovornosti i lako testiranje. Baza podataka je MySQL, a entiteti su mapirani uz pomoć JPA/Hibernate-a. React klijent (Florist/User) komunicira isključivo sa REST API-jem, pa se UI i backend mogu razvijati i skalirati nezavisno. Ovakva podela olakšava održavanje, dodavanje novih funkcionalnosti (npr. novi tipovi aranžmana ili izveštaji) i zamenu bilo kog sloja bez promena u ostalim delovima sistema.

Slika 12 - arhitektura

# Objašnjenje frontend koda

Aplikacija je izrađena u Reactu i u ovom delu projekta biće opisane komponente koje su kreirane. React nam omogućava da interfejs gradimo pomoću komponenti – manjih, nezavisnih delova koda koji prikazuju određeni deo stranice (npr. dugme, forma, kartica). Svaka komponenta ima svoj izgled i ponašanje, i može da se koristi više puta u različitim delovima aplikacije.

U Reactu često koristimo hook-ove, koji su posebne funkcije pomoću kojih možemo da radimo sa stanjem i životnim ciklusom komponente. Na primer, useState koristimo da čuvamo podatke unutar komponente, a useEffect da reagujemo na promene (npr. da učitamo podatke sa servera).

Posebno se trudimo da pravimo reusable komponente, tj. one koje mogu da se ponovo koriste na više mesta bez potrebe da pišemo isti kod više puta. To olakšava održavanje i čini aplikaciju preglednijom i bržom za razvoj.

### App.js

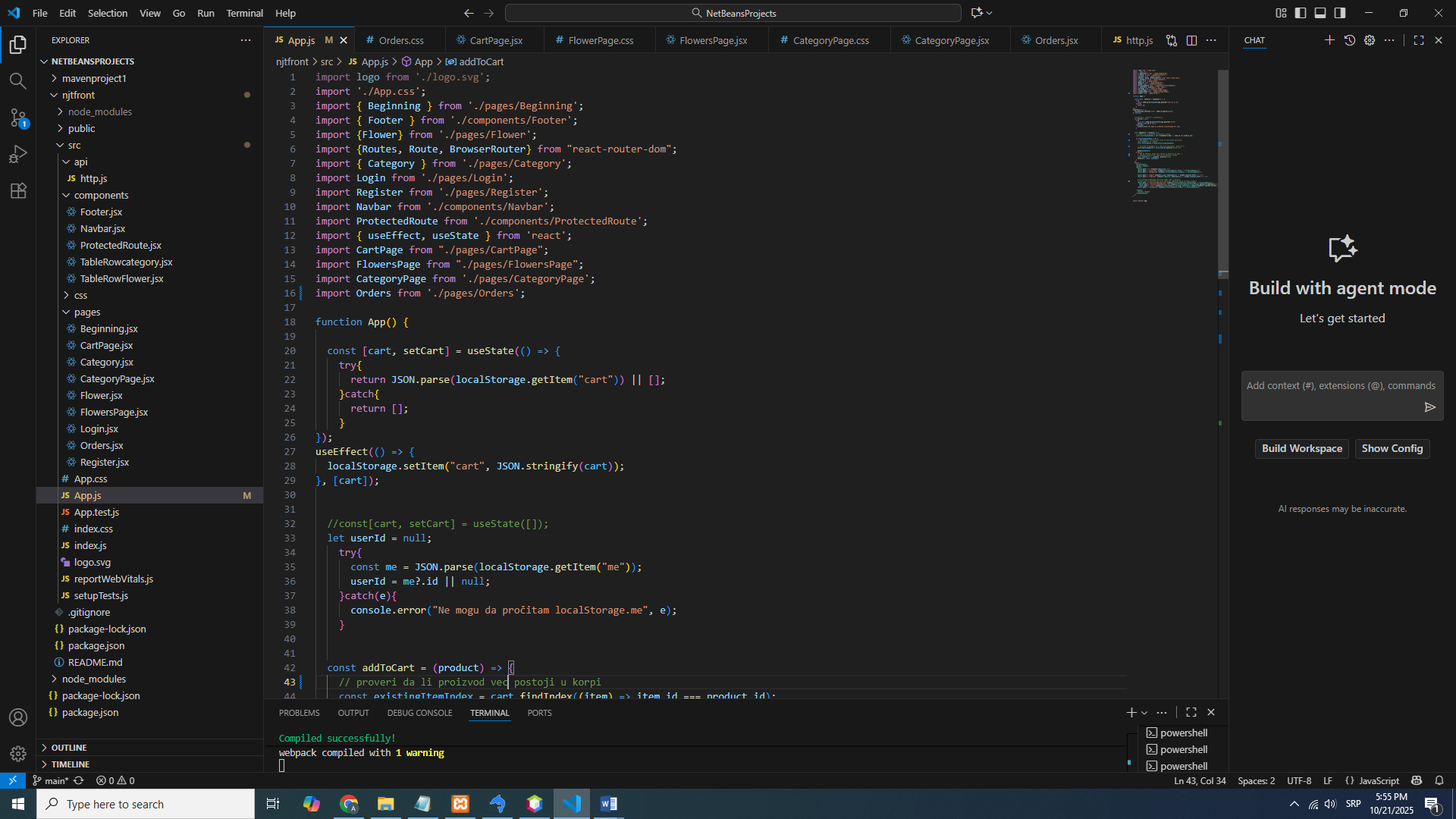
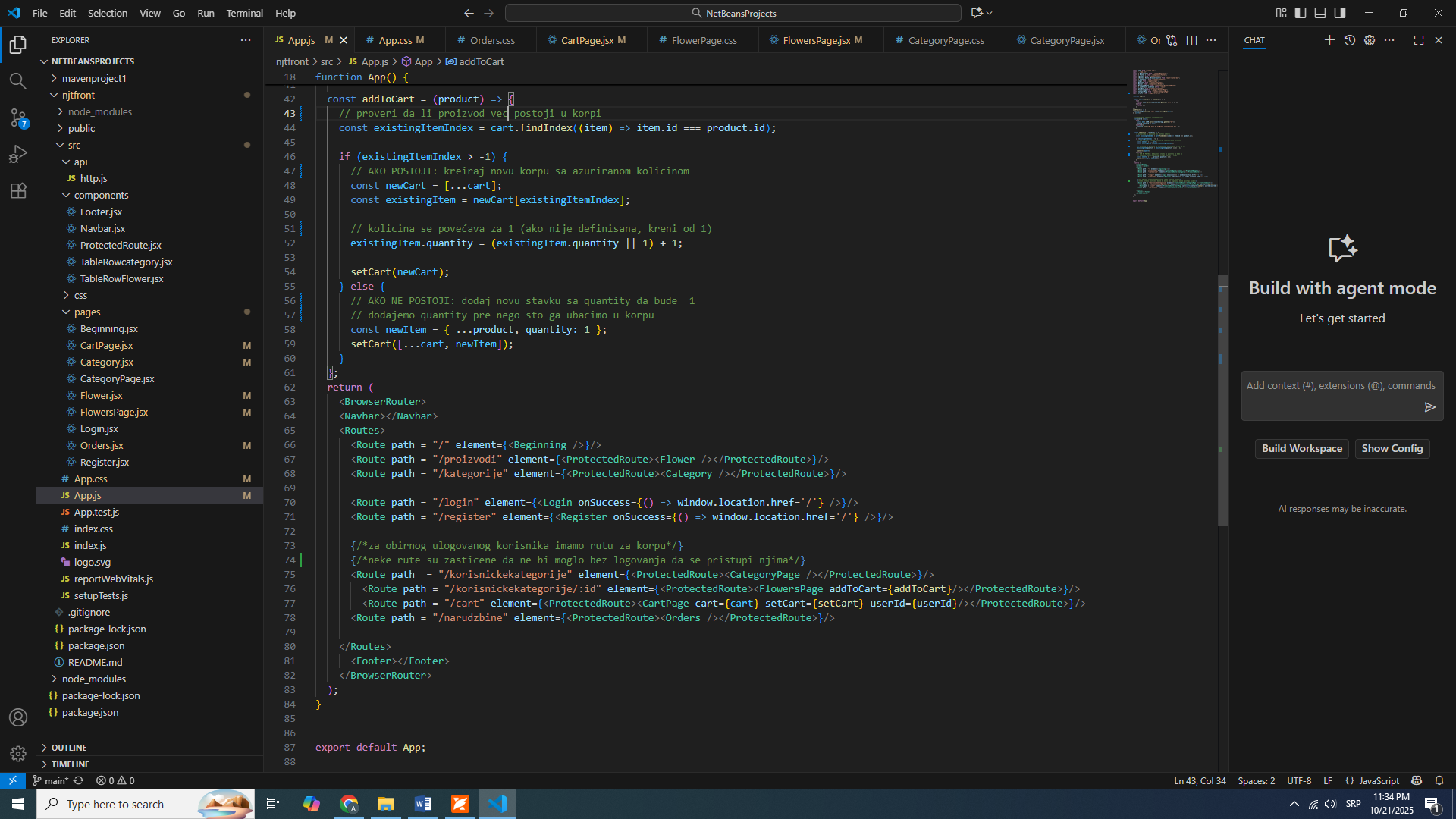
Fajl App.js predstavlja centralnu komponentu React aplikacije, funkcionišući kao "stub" celokupne aplikacije. Njegova primarna svrha je upravljanje stanjem aplikacije (state management) i definisanjem ruta (routing) za navigaciju između različitih delova aplikacije.

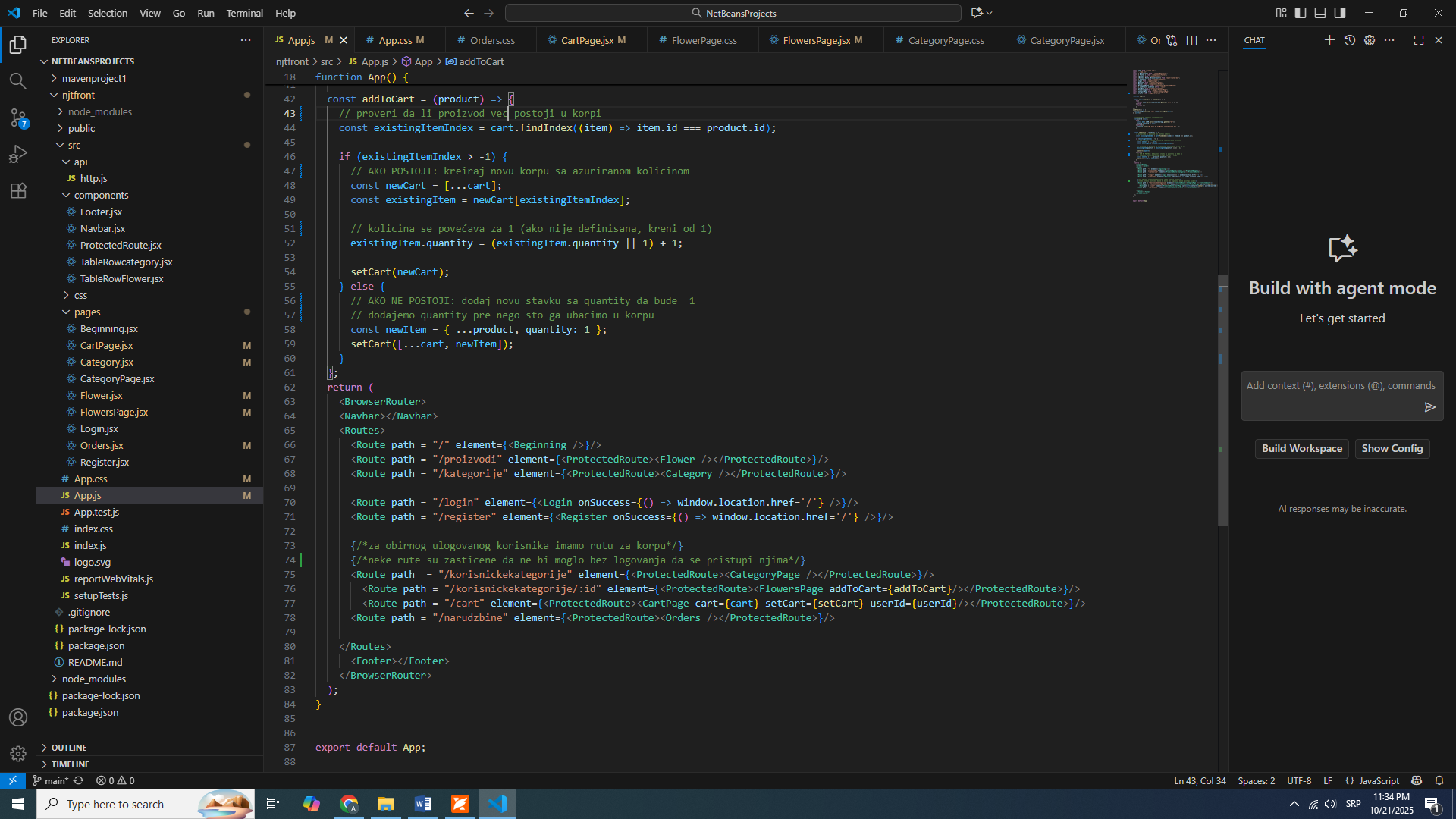
Na početku fajla se uvoze sve neophodne biblioteke i komponente, uključujući: React i useState, useEffect hookovi, react-router-dom. Komponenta App inicijalizuje stanje za korisničku korpu: useState za cart koristi se za skladištenje trenutnog sadržaja korpe. Inicijalno stanje se pokušava učitati iz lokalne memorije pretraživača (localStorage) pod ključem "cart". Ako učitavanje ne uspe ili podatak ne postoji, korpa se inicijalizuje kao prazan niz ([]). UseEffect *hook* osigurava da se svaki put kada se stanje korpe (cart) promeni, njena nova vrednost automatski sačuva u localStorage. Ovo omogućava da sadržaj korpe ostane zapamćen čak i nakon osvežavanja stranice.

Aplikacija pokušava da pročita ID ulogovanog korisnika (userId) iz localStorage ključa "me". Ovaj ID se koristi u komponentama koje zavise od ulogovanog korisnika (npr. na stranici korpe).

Definisana je funkcija addToCart koja služi za dodavanje proizvoda. Kada se proizvod doda, funkcija proverava da li on već postoji u korpi. ako postoji povećava se samo svojstvo quantity (količina) za 1. Ako ne postoji proizvod se dodaje u korpu sa inicijalnom količinom quantity: 1. Ova funkcija se prosleđuje komponentama nižeg reda (FlowersPage) kako bi mogle direktno da manipulišu stanjem korpe.

Unutar return bloka, App.js koristi BrowserRouter, Routes i Route za postavljanje strukture navigacije aplikacije. Stalni elementi: Komponente Navbar i Footer su pozicionirane izvan Routes, što znači da će biti vidljive na svakoj stranici. Svaki Route mapira određenu URL putanju (npr. /, /login, /cart) ka komponenti koja treba da se prikaže na toj adresi. Zaštićene rute (ProtectedRoute) osiguravaju da se ovim stranicama može pristupiti samo ako je korisnik prethodno prijavljen (autorizovan).





### Početna stranica

Fajl Beginning.jsx definiše glavnu komponentu početne stranice (Home Page) aplikacije, koja je prva stranica koju korisnik vidi. Njena primarna funkcija je da predstavi aplikaciju, istakne ključne proizvode i kategorije, i omogući osnovnu navigaciju. Komponenta je namerno *bezstanja* u smislu eksternih podataka, lokalno stanje (useState) se koristi isključivo za kontrolu *frontend* interaktivnosti kao što je upravljanje selekcijom filtera ("sve", "poklon", itd.), omogućavajući dinamičko prikazivanje podskupa proizvoda iz niza featured. Stranica je podeljena na tri dela. Hero Sekcija (<section className="hero">) sadrži upečatljiv naslov i kratak podnaslov, uz listu istaknutih pogodnosti (stats blok – npr. 30+ cvetova, 45 min dostave).

Vizuelni element u *hero* sekciji realizovan je kroz jednostavnu CSS animaciju i React useEffect hook. UseEffect postavlja interval (tajmer) koji automatski menja stanje index svakih 3000 ms. Ovo obezbeđuje laganu, neometajuću animaciju (slideshow) koja dinamički menja *cover* sliku, poboljšavajući vizuelni utisak bez dodatnog opterećenja performansi.

Sekcija izdvojenih kategorija i dugmad za filtriranje (chips): Omogućavaju korisniku da na jednostavan način pretražuje proizvode po ključnom atributu tip (poklon, dekoracija, proslave).

Navigacija: Svaka kartica sadrži dugme "Pogledaj ponudu" koje koristi useNavigate za programsko preusmeravanje korisnika na specifičnu, zaštićenu rutu (/korisnickekategorije/:id), vodeći ga direktno ka detaljnoj ponudi izabrane kategorije.

import React from 'react'

import '../css/Beginning.css';

import { useState } from "react";

import { useEffect } from "react";

import { useNavigate } from "react-router-dom";

export const Beginning = () => {

  const navigate = useNavigate();

    const categories = [

  {name: "Lale", img: "https://images.unsplash.com/photo-1602992191575-59a2c7bd0ae1?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=689",},

  {name: "Kaktus", img: "https://images.unsplash.com/photo-1683688412173-148e0ba57c19?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=735",},

  {name: "Pampas", img: "https://images.unsplash.com/photo-1666425865764-862f7ea60cf1?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=735",},

  {name: "Hrizantema buket", img: "https://plus.unsplash.com/premium\_photo-1713823800827-4c10d4d37585?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=687",},

  {name: "Pahira", img: "https://images.unsplash.com/photo-1597055181187-1f9b726e2d66?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=1964",},

];

    const featured = [

    {

      id: 5,name: "Rezano cveće",rating: 4.9,povod: "Savršeno za poklon i dekoraciju doma",cena: "od 300 RSD po cvetu",tip: "poklon",img: "https://images.unsplash.com/photo-1578972497170-bfc780c65f65?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=686",},

    {

      id: 7,name: "Buketi",rating: 4.8,povod: "Rođendani, godišnjice, pokloni iz srca",cena: "od 1200 RSD",tip: "poklon",img: "https://images.unsplash.com/photo-1660549071389-962f436e1b5f?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=765",},

    {

      id: 8,name: "Saksijsko cveće",rating: 4.7,povod: "Dugotrajan poklon za kuću ili kancelariju",cena: "od 800 RSD",tip: "dekoracija",img: "https://plus.unsplash.com/premium\_photo-1672998159540-0a3f849fe3c6?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=687",},

    {

      id: 6,name: "Dekorativno i suvo cveće",rating: 4.6,povod: "Idealno za uređenje enterijera i proslave",cena: "od 1000 RSD",tip: "proslave",img: "https://plus.unsplash.com/premium\_photo-1676383427728-5e6df7edcf02?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=713",},

  ];

  const [filter, setFilter] = useState("sve");

  const filtered =

    filter === "sve"

      ? featured

      : featured.filter((item) => item.tip === filter);

      const slike = [

  "https://images.unsplash.com/photo-1563241527-3004b7be0ffd?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=687",

  "https://images.unsplash.com/photo-1608982216701-a9ab65b4a3a2?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=805",

  "https://plus.unsplash.com/premium\_photo-1667870034632-134c46ef9a47?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=764",

  "https://plus.unsplash.com/premium\_photo-1670426501265-3cd3d4a30f8f?ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&q=80&w=687",

];

const [index, setIndex] = useState(0);

  useEffect(() => {

    const interval = setInterval(() => {

      setIndex((prev) => (prev + 1) % slike.length);

    }, 3000); // menja sliku svakih 3 sekunde

    return () => clearInterval(interval);

  }, []);

  return (

    <>

    <section className="hero">

      <div className="container hero-grid">

        <div className="hero-copy">

          <h1>Pošalji cveće onima koje voliš, brzo i sa stilom </h1>

          <p>Odaberi savršen poklon, dodaj poruku i obraduj nekog danas.</p>

          {/\*

          <form className="hero-search" onSubmit={(e) => e.preventDefault()}>

            <input

              placeholder="Unesi povod/kategoriju"

              className="hero-input"

            />

            <button className="btn btn-primary">Pronađi kategoriju</button>

          </form>

          \*/}

          <div className="stats">

            <div>

              <strong>30+</strong>

              <span>Različitih cvetova</span>

            </div>

            <div>

              <strong>45 min</strong>

              <span>Prosečno vreme dostave</span>

            </div>

            <div>

              <strong>Besplatna dostava</strong>

              <span>Za porudžbine preko 2500 RSD</span>

            </div>

          </div>

        </div>

        <div className="hero-art">

          <div className="hero-slideshow">

            <img src={slike[index]} alt="Buket" className="hero-slideshow-img" />

            </div>

          </div>

      </div>

    </section>

    <section id="categories" className="section">

  <div className="container">

    <div className="section-head">

      <h2>Proizvodi</h2>

      <a className="link" href="#ponuda">Izdvojeni proizvodi</a>

    </div>

    <div className="grid cats">

      {categories.map((c) => (

        <a key={c.name} className="cat" href="#ponuda">

          <img src={c.img} alt={c.name} />

          <span>{c.name}</span>

        </a>

      ))}

    </div>

  </div>

</section>

      <section id="featured" className="section alt">

      <div className="container">

        <div className="section-head">

          <h2>Izdvojene kategorije</h2>

        </div>

        {/\* Dugmići za filtere \*/}

        <div className="filters">

          <button

            className={`chip ${filter === "sve" ? "active" : ""}`}

            onClick={() => setFilter("sve")}

          >

            Sve

          </button>

          <button

            className={`chip ${filter === "poklon" ? "active" : ""}`}

            onClick={() => setFilter("poklon")}

          >

            Za poklon

          </button>

          <button

            className={`chip ${filter === "dekoracija" ? "active" : ""}`}

            onClick={() => setFilter("dekoracija")}

          >

            Za dekoraciju

          </button>

          <button

            className={`chip ${filter === "proslave" ? "active" : ""}`}

            onClick={() => setFilter("proslave")}

          >

             Za proslave

          </button>

        </div>

        {/\* Kartice \*/}

        <div className="grid rest">

          {filtered.map((r) => (

            <article className="rest-card" key={r.id}>

              <div className="thumb">

                <img src={r.img} alt={r.name} />

                <div className="overlay">

                  <span>{r.cena}</span>

                </div>

              </div>

              <div className="content">

                <h3>{r.name}</h3>

                <p className="povod">{r.povod}</p>

                <div className="meta">

                  <span className="rating">⭐ {r.rating}</span>

                  <button className="btn tiny" onClick={() => navigate(`/korisnickekategorije/${r.id}`)}>Pogledaj ponudu</button>

                </div>

              </div>

            </article>

          ))}

        </div>

      </div>

    </section>

    </>

  )

}

### Stranica za registraciju

Komponenta Register.jsx predstavlja samostalni modul odgovoran za registraciju novih korisnika i njihovu automatsku prijavu na platformu. Dizajnirana je da bude efikasna i da osigura bezbedan prenos korisničkih kredencijala ka backend servisu.

Komponenta koristi React hookove za upravljanje svim podacima i statusima povezanim sa procesom registracije:

* form State: Skladišti unos korisnika za korisničko ime, email i lozinku. Korišćenjem kontrolisanih komponenti (controlled components), svaka promena u polju za unos odmah ažurira odgovarajuće stanje form.
* Statusi aplikacije: loading prati da li je zahtev za registraciju trenutno u toku, dok error i ok služe za prikaz povratnih informacija korisniku (poruke o grešci ili uspehu).

Centralna funkcionalnost je sadržana u asinhronoj funkciji handleSubmit, koja se aktivira podnošenjem forme:

1. Registracija: prvi HTTP POST poziv šalje korisničke podatke na rutu /auth/register.
2. Automatska prijava: u cilju poboljšanja korisničkog iskustva, nakon uspešne registracije, automatski se šalje drugi POST poziv na rutu /auth/login. Time se eliminiše potreba da korisnik ručno unosi podatke za prijavu odmah nakon kreiranja naloga.
3. Skladištenje autentifikacije: odgovor dobijen prijavom sadrži autorizacioni token i podatke o korisniku (user). Ovi ključni podaci se skladište u localStorage pretraživača pod ključevima "token" i "me". Ovo je neophodno za očuvanje statusa prijave i autorizaciju budućih zahteva ka zaštićenim rutama.
4. Završetak toka: po uspešnom završetku, poziva se funkcija onSuccess (koja je prosleđena kao *prop (App.js)*), a njena tipična uloga je preusmeravanje korisnika na glavnu stranicu aplikacije.

Interfejs je vizuelno sveden i fokusiran na cilj. Dinamički se prikazuju poruke o grešci (auth-alert) ili uspehu (auth-success). Dugme za podnošenje forme je onesposobljeno (disabled={loading}) dok je zahtev u toku, sprečavajući višestruko slanje podataka. U podnožju komponente, obezbeđena je jednostavna navigacija ka stranici za prijavu putem linka (/login).

import React, { useState } from "react";

import http from "../api/http";

import "../css/auth.css";

export default function Register({onSuccess}) {

  const [form, setForm] = useState({ username: "", email: "", password: "" });

  const [loading, setLoading] = useState(false);

  const [ok, setOk] = useState("");

  const [error, setError] = useState("");

  async function handleSubmit(e) {

    e.preventDefault();

    setLoading(true);

    setError("");

    setOk("");

    try {

      await http.post("/auth/register", form);

      const loginRes = await http.post("/auth/login", {

        username: form.username,

        password: form.password,

      });

      localStorage.setItem("token", loginRes.data.token);

      localStorage.setItem("me", JSON.stringify(loginRes.data.user));

      setOk("Account created successfully!");

      if(typeof onSuccess === "function") onSuccess(loginRes.data.user);

    } catch (err) {

      setError(err.response?.data?.message || "Registration failed");

    } finally {

      setLoading(false);

    }

  }

  return (

    <div className="auth-wrap">

      <div className="auth-card">

        <h2>Napravite profil</h2>

        <p className="muted">Pridružite nam se i nastavite sa razgledanjem!</p>

        {error && <div className="auth-alert">{error}</div>}

        {ok && <div className="auth-success">{ok}</div>}

        <form onSubmit={handleSubmit} className="auth-form">

          <div className="field">

            <label>Username</label>

            <input

              type="text"

              value={form.username}

              onChange={(e) => setForm({ ...form, username: e.target.value })}

              required

            />

          </div>

          <div className="field">

            <label>Email</label>

            <input

              type="email"

              value={form.email}

              onChange={(e) => setForm({ ...form, email: e.target.value })}

              required

            />

          </div>

          <div className="field">

            <label>Password</label>

            <input

              type="password"

              value={form.password}

              onChange={(e) => setForm({ ...form, password: e.target.value })}

              required

              minLength={6}

            />

          </div>

          <button className="btn-primary" disabled={loading}>

            {loading ? "Creating..." : "Create account"}

          </button>

        </form>

        <div className="auth-footer">

          <span className="muted">Već imate profil?</span>

          <a href="/login">Log in</a>

        </div>

      </div>

    </div>

  );

}

### Stranica za login

Fajl Login.jsx definiše React komponentu koja služi za prijavu postojećih korisnika u aplikaciju. Njegova primarna svrha je da sigurno prikupi korisničko ime i lozinku, pošalje ih serveru radi validacije, i po uspešnoj proveri, uspostavi autentifikovanu sesiju za korisnika. Ovaj modul je strukturalno i funkcionalno sličan modulu za registraciju

Komponenta koristi useState za praćenje promenljivih ključnih za formu i komunikaciju:

* form: Objekat koji skladišti trenutne vrednosti polja username i password.
* loading: Boolean vrednost koja se postavlja na true dok se čeka odgovor sa servera. Koristi se za vizuelno označavanje procesa prijave i onemogućavanje dugmeta.
* error: Skladišti poruku o grešci ukoliko prijava ne uspe (npr. pogrešno korisničko ime ili lozinka).

Funkcija handleSubmit je asinhrona i sadrži jezgro logike za autentifikaciju:

1. Slanje Kredencijala: Šalje se HTTP POST zahtev ka backend ruti /auth/login, noseći sa sobom podatke iz form stanja.
2. Uspešna Autentifikacija: Ako server vrati uspešan odgovor (što uključuje autorizacioni token i podatke o korisniku):
   1. Skladištenje Tokena: res.data.token se čuva u localStorage pod ključem "token". Ovaj token je neophodan za autorizaciju svih budućih zahteva ka zaštićenim API rutama.
   2. Skladištenje Korisničkih Podataka: Podaci o korisniku res.data.user se serijalizuju u JSON formatu i čuvaju u localStorage pod ključem "me".
   3. Pozivanje onSuccess: Poziva se *callback* funkcija onSuccess (koja je prosleđena iz App.js i obično preusmerava korisnika na glavnu stranicu).
3. Obrada Grešaka: Ako dođe do greške greška se hvata u bloku catch, a poruka o grešci (err.response?.data?.message) se postavlja u stanje error i prikazuje korisniku.

Komponenta prikazuje jednostavnu, fokusiranu formu za prijavu. Poruke o grešci se dinamički prikazuju koristeći stil auth-alert. U podnožju komponente (auth-footer), obezbeđen je link "Registrujte se" koji korisnika preusmerava na rutu /register, olakšavajući navigaciju za one koji još uvek nemaju nalog. Dugme za prijavu menja tekst i status zavisno od stanja loading.

import React, { useState } from "react";

import http from "../api/http";

import "../css/auth.css";

export default function Login({onSuccess}) {

  const [form, setForm] = useState({ username: "",  password: "" });

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState("");

  async function handleSubmit(e) {

    e.preventDefault();

    setLoading(true);

    setError("");

     try {

      const res = await http.post("/auth/login", form);

      //ocekujemo token

      localStorage.setItem("token", res.data.token);

      localStorage.setItem("me", JSON.stringify(res.data.user));

      if(typeof onSuccess === "function") onSuccess(res.data.user);

    } catch (err) {

      setError(err.response?.data?.message || "Loging ing failed");

    } finally {

      setLoading(false);

    }}

    return (

    <div className="auth-wrap">

      <div className="auth-card">

        <h2>Ulogujte se</h2>

        <p className="muted">Pridružite nam se i nastavite sa razgledanjem!</p>

        {error && <div className="auth-alert">{error}</div>}

        <form onSubmit={handleSubmit} className="auth-form">

          <div className="field">

            <label>Username</label>

            <input

              type="text"

              value={form.username}

              onChange={(e) => setForm({ ...form, username: e.target.value })}

              required

            />

          </div>

          <div className="field">

            <label>Password</label>

            <input

              type="password"

              value={form.password}

              onChange={(e) => setForm({ ...form, password: e.target.value })}

              required

              minLength={6}

            />

          </div>

          <button className="btn-primary" disabled={loading}>

            {loading ? "Loging in..." : "Log in"}

          </button>

        </form>

        <div className="auth-footer">

          <span className="muted">Nemate profil?</span>

          <a href="/register">Registrujte se</a>

        </div>

      </div>

    </div>

  );

}

### Stranica za proizvode – cvećar

Fajl Flower.jsx definiše React komponentu koja služi kao administratorska stranica za upravljanje proizvodima (cvećem) u bazi podataka. Ovo je ključni modul za *backend* operacije, omogućavajući cvećaru pregled, pretragu, sortiranje, dodavanje, izmenu i brisanje artikala.

Što se tiče inicijalizacije komponenta koristi useState za praćenje svih promenljivih neophodnih za rad administrativnog interfejsa:

* data: Niz koji skladišti sve proizvode (cveće) učitano iz baze.
* categories: Niz koji sadrži sve dostupne kategorije.
* loading: Boolean zastavica koja pokazuje status učitavanja podataka.
* q: Stanje za čuvanje unosa za pretragu po nazivu proizvoda.
* sort: Objekat koji definiše kolonu po kojoj se sortira (by) i smer sortiranja (dir, "asc" ili "desc").
* showmodal I editing: Stanja koja kontrolišu prikaz modala za unos/izmenu i ID/objekat proizvoda koji se trenutno menja.
* form: Objekat koji sadrži podatke za novi proizvod ili izmenu postojećeg .

Kada se komponenta prvi put učita, useEffect pokreće asinhronu funkciju fetchData. Ova funkcija koristi Promise.all za istovremeno slanje dva HTTP Get zahteva:

* Učitavanje liste proizvoda sa rute /flower.
* Učitavanje liste kategorija sa rute /category. Ovaj mehanizam minimizira vreme učitavanja i postavlja početne vrednosti za data I categories.

Što se tiče filtriranja lista proizvoda se dinamički filtrira na osnovu unosa u polju q. Sa druge strane funkcija toggleSort menja stanje sort. Ako se klikne na istu kolonu, smer sortiranja se menja (asc u desc i obrnuto); u suprotnom, sortiranje kreće od asc za novu kolonu. Sama funkcija filtriranja implementira logiku za sortiranje numeričkih kolona (id I price) i tekstualnih kolona.

Modal je komponenta koja podržava sve ključne CRUD (*Create, Read, Update, Delete*) operacije:

* Dodavanje/Izmena (Save): Funkcija handleSave se pokreće podnošenjem forme u modalu.
* Payload Priprema: Pre slanja, vrednosti se konvertuju: price u broj, a categoryId u null ako nije izabrana kategorija, ili u broj.
* Logika PUT/POST: Ako je definisano stanje editing, šalje se PUT zahtev za ažuriranje na /flower/flower{id}. U suprotnom, šalje se POST zahtev za kreiranje novog proizvoda na /flower. Stanje data se ažurira novim ili izmenjenim proizvodom.
* Brisanje (Delete): Funkcija handleDelete šalje DELETE zahtev na /flower{id} nakon potvrde korisnika. Nakon uspešnog brisanja, proizvod se uklanja iz stanja data.

import React, { useEffect, useState } from "react";

import http from "../api/http";

import "../css/Flower.css";

import { TableRowFlower } from "../components/TableRowFlower";

export const Flower = () => {

  const [data, setData] = useState([]);

  const [categories, setCategories] = useState([]);

  const [loading, setLoading] = useState(true);

  const [q, setQ] = useState("");

  const [sort, setSort] = useState({ by: "id", dir: "asc" });

  const [showModal, setShowModal] = useState(false);

  const [editing, setEditing] = useState(null);

  const [form, setForm] = useState({

    name: "",

    description: "",

    price: "",

    image\_url: "",

    category\_id: "",

  });

  useEffect(() => {

    const fetchData = async () => {

      try {

        const [flowerRes, categoryRes] = await Promise.all([

          http.get("/flower"),

          http.get("/category"),

        ]);

        setData(flowerRes.data);

        setCategories(categoryRes.data);

      } catch (error) {

        console.error("Greška pri učitavanju podataka", error);

      } finally {

        setLoading(false);

      }

    };

    fetchData();

  }, []);

  const toggleSort = (col) => {

    setSort((s) =>

      s.by === col ? { by: col, dir: s.dir === "asc" ? "desc" : "asc" } : { by: col, dir: "asc" }

    );

  };

  const filtered = [...data]

    .filter((f) => f.name?.toLowerCase().includes(q.toLowerCase()))

    .sort((a, b) => {

      const va = a[sort.by];

      const vb = b[sort.by];

      const numCols = ["id", "price"];

      if (numCols.includes(sort.by)) {

        const na = Number(va) || 0,

          nb = Number(vb) || 0;

        return sort.dir === "asc" ? na - nb : nb - na;

      }

      return sort.dir === "asc"

        ? String(va ?? "").localeCompare(String(vb ?? ""))

        : String(vb ?? "").localeCompare(String(va ?? ""));

    });

const handleOpenModal = (flower = null) => {

    if (flower) {

        setEditing(flower);

        const categoryId = flower.categoryId !== null ? String(flower.categoryId) : "";

        setForm({

            name: flower.name,

            description: flower.description,

            price: flower.price,

            imageUrl: flower.imageUrl || "",

            categoryId: categoryId,

        });

    } else {

    }

    setShowModal(true);

};

  const handleCloseModal = () => {

    setShowModal(false);

    setEditing(null);

  };

const handleSave = async (e) => {

    e.preventDefault();

    const payload = {

      ...form,

      price: Number(form.price),

      imageUrl: form.imageUrl,

      categoryId: form.categoryId === "" ? null : Number(form.categoryId),

    };

    try {

      if (editing) {

        // azuriranje

        const res = await http.put(`/flower/${editing.id}`, payload);

        setData((prev) =>

          prev.map((f) => (f.id === editing.id ? res.data : f))

        );

        alert("Cvet je uspešno IZMENJEN!");

      } else {

        // kreiranje novog cveta

        const res = await http.post("/flower", payload);

        setData((prev) => [...prev, res.data]);

        alert("Novi cvet je uspešno DODAT!");

      }

      handleCloseModal();

    } catch (e) {

      alert(e?.response?.data?.message || e.message);

    }

};

  const handleDelete = async (id) => {

    if (!window.confirm("Da li sigurno želiš da obrišeš proizvod?")) return;

    try {

      await http.delete(`/flower/${id}`);

      setData((prev) => prev.filter((r) => r.id !== id));

    } catch (e) {

      alert(e?.response?.data?.message || e.message);

    }

  };

  return (

    <div className="container admin-wrap">

      <header className="admin-head">

        <div>

          <h1>Proizvodi</h1>

          <p className="muted">Pregled, unos i izmena proizvoda u bazi.</p>

        </div>

        <div className="row-gap">

          <input

            className="input"

            placeholder="Pretraga (naziv proizvoda)"

            value={q}

            onChange={(e) => setQ(e.target.value)}

          />

          <button className="btn primary" onClick={() => handleOpenModal()}>

            + Novi proizvod

          </button>

        </div>

      </header>

      {loading ? (

        <p>Učitavanje...</p>

      ) : (

        <table className="table">

          <thead>

            <tr>

              <th onClick={() => toggleSort("id")}>

                ID {sort.by === "id" && (sort.dir === "asc" ? "▲" : "▼")}

              </th>

              <th>Naziv</th>

              <th>Opis</th>

              <th onClick={() => toggleSort("price")}>

                Cena {sort.by === "price" && (sort.dir === "asc" ? "▲" : "▼")}

              </th>

              <th>Akcije</th>

            </tr>

          </thead>

          <tbody>

            {filtered.map((f) => (

              <TableRowFlower

                key={f.id}

                flower={f}

                categories={categories}

                onUpdate={() => handleOpenModal(f)}

                onDelete={() => handleDelete(f.id)}

              />

            ))}

          </tbody>

        </table>

      )}

      {/\* Modal \*/}

      {showModal && (

        <div className="modal-overlay">

          <div className="modal-content">

            <h2>{editing ? "Izmeni proizvod" : "Dodaj novi proizvod"}</h2>

            <form onSubmit={handleSave} className="modal-form">

              <input

                className="input"

                placeholder="Naziv"

                value={form.name}

                onChange={(e) => setForm({ ...form, name: e.target.value })}

                required

              />

              <input

                className="input"

                placeholder="Opis"

                value={form.description}

                onChange={(e) =>

                  setForm({ ...form, description: e.target.value })

                }

              />

              <input

                className="input"

                placeholder="Cena (RSD)"

                type="number"

                value={form.price}

                onChange={(e) => setForm({ ...form, price: e.target.value })}

              />

              <input

                className="input"

                placeholder="URL slike"

                value={form.imageUrl}

                onChange={(e) =>

                  setForm({ ...form, imageUrl: e.target.value })

                }

              />

              {/\* padajuca lista za kategoriju \*/}

              <select

                className="input"

                value={form.categoryId}

                onChange={(e) =>

                  setForm({ ...form, categoryId: e.target.value })

                }

                required

              >

                <option value="">Izaberite kategoriju</option>

                {categories.map((c) => (

                  <option key={c.id} value={c.id}>

                    {c.name}

                  </option>

                ))}

              </select>

              <div className="modal-buttons">

                <button className="btn primary" type="submit">

                  Sačuvaj

                </button>

                <button className="btn" type="button" onClick={handleCloseModal}>

                  Otkaži

                </button>

              </div>

            </form>

          </div>

        </div>

      )}

    </div>

  );

};

### Stranica za narudžbine – cvećar

Fajl Orders.jsx definiše ključnu komponentu za administraciju i pregled svih narudžbina u sistemu, pozicioniranu na ruti /narudzbine, koja je zaštićena komponentom ProtectedRoute. Ova komponenta koristi useState za praćenje liste narudžbina (orders), statusa učitavanja (loading) i eventualnih grešaka (error). Nakon inicijalnog renderovanja, useEffect pokreće asinhrono dohvatanje podataka sa backend rute /orders. Primljeni podaci se odmah sortiraju tako da se najnovije narudžbine (koristeći createdAt vremensku oznaku) prikazuju na vrhu liste, čime se obezbeđuje prioritetno rukovanje.

Korisnički interfejs prikazuje naslov "Sve Narudžbine" i, ukoliko postoje narudžbine, predstavlja ih u detaljnoj tabelarnoj strukturi. Za svaku narudžbinu su vidljive ključne informacije, uključujući ID narudžbine, Status, ID korisnika, formatirani Datum kreiranja, ukupan broj stavki, te eventualna Napomena.

Centralna administrativna funkcionalnost je implementirana kroz funkciju handleStatusChange, koja se pokreće promenom vrednosti u padajućem meniju u koloni "Akcije". Ova funkcija prvo zahteva konfirmaciju putem dijaloga, a zatim šalje PATCH HTTP zahtev na rutu /orders/{orderId}/status sa novim statusom kao query parametrom. Nakon uspešnog odgovora, stanje narudžbina (orders) se lokalno ažurira kako bi se novi status odmah reflektovao u tabeli. Status se može menjati između dostupnih opcija ("CREATED", "CONFIRMED", "COMPLETED", "CANCELED"), pri čemu je manipulacija statusom onemogućena za narudžbine koje su već označene kao 'COMPLETED' ili 'CANCELED'. Na ovaj način, Orders.jsx služi kao efikasan kontrolni panel za administraciju protoka narudžbina.

import React, { useEffect, useState } from "react";

import http from "../api/http";

import "../css/Orders.css";

export default function Orders() {

  const [orders, setOrders] = useState([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState(null);

  // fja za dohvatanje narudzbina

  const fetchOrders = async () => {

    try {

      setLoading(true);

      const res = await http.get("/orders");

      // najnovije narudzbine na pocetku

      const sortedOrders = res.data.sort(

        (a, b) => new Date(b.createdAt) - new Date(a.createdAt)

      );

      setOrders(sortedOrders);

      setError(null);

    } catch (err) {

      console.error("Greška pri učitavanju narudžbina:", err);

      setError("Neuspelo učitavanje narudžbina.");

    } finally {

      setLoading(false);

    }

  };

  useEffect(() => {

    fetchOrders();

  }, []);

  // fja za promenu statusa

  const handleStatusChange = async (orderId, newStatus) => {

    if (!window.confirm(`Da li ste sigurni da želite da promenite status narudžbine #${orderId} u ${newStatus}?`)) {

      return;

    }

    try {

      await http.patch(`/orders/${orderId}/status?status=${newStatus}`);

      setOrders((prevOrders) =>

        prevOrders.map((order) =>

          order.id === orderId ? { ...order, status: newStatus } : order

        )

      );

    } catch (err) {

      console.error("Greška pri ažuriranju statusa:", err);

      alert("Neuspela promena statusa narudžbine.");

    }

  };

  // pomocna funkcija za formatiranje datuma

  const formatDateTime = (dateTime) => {

      const options = {

          year: 'numeric',

          month: '2-digit',

          day: '2-digit',

          hour: '2-digit',

          minute: '2-digit',

          second: '2-digit'

      };

      return new Date(dateTime).toLocaleDateString('sr-RS', options);

  };

  if (loading) return <div className="loading">Učitavanje narudžbina...</div>;

  if (error) return <div className="error">{error}</div>;

  const statuses = ["CREATED", "CONFIRMED", "COMPLETED", "CANCELED"]; // moguci statusi

  return (

    <div className="orders-wrap">

      <h2>Sve Narudžbine</h2>

      {orders.length === 0 ? (

        <p>Trenutno nema narudžbina.</p>

      ) : (

        <table className="orders-table">

          <thead>

            <tr>

              <th>ID</th>

              <th>Status</th>

              <th>Korisnik ID</th>

              <th>Datum kreiranja</th>

              <th>Ukupno stavki</th>

              <th>Napomena</th>

              <th>Akcije</th>

            </tr>

          </thead>

          <tbody>

            {orders.map((order) => (

              <tr key={order.id} className={`status-${order.status.toLowerCase()}`}>

                <td>{order.id}</td>

                <td>

                  <span className={`status-badge status-${order.status.toLowerCase()}`}>

                    {order.status}

                  </span>

                </td>

                <td>{order.userId}</td>

                <td>{formatDateTime(order.createdAt)}</td>

                <td>{order.items.length}</td>

                <td>{order.napomena}</td>

                <td>

                  <select

                    value={order.status}

                    onChange={(e) => handleStatusChange(order.id, e.target.value)}

                    disabled={order.status === 'COMPLETED' || order.status === 'CANCELED'} // ne dozvoli promenu ako je zavrsena/otkazana

                  >

                    {statuses.map((s) => (

                      <option key={s} value={s}>

                        {s}

                      </option>

                    ))}

                  </select>

                </td>

              </tr>

            ))}

          </tbody>

        </table>

      )}

    </div>

  );

}

### Stranice kojima pristupa Korisnik - kategorije, proizvodi

Moduli CategoryPage.jsx i FlowersPage.jsx zajedno čine ključni tok za prikaz ponude i interakciju sa korpom, namenjen ulogovanom korisniku. Komponenta CategoryPage.jsx služi kao prvi korak, prikazujući pregled svih dostupnih kategorija proizvoda. Koristeći useEffect, ona asinhrono uzima kompletnu listu kategorija sa backend rute /category. Nakon učitavanja, stranica prikazuje naslov i podnaslov, te responzivnu mrežu kartica, od kojih svaka predstavlja jednu kategoriju sa vizuelno atraktivnom slikom koja je mapirana iz objekta categoryImages. Klikom na bilo koju karticu, komponenta koristi useNavigate za preusmeravanje korisnika na specifičnu, zaštićenu rutu /korisnickekategorije/:id, čime započinje pregled proizvoda te kategorije.

Nakon navigacije, aktivira se komponenta FlowersPage.jsx, čija je primarna uloga da prikaže cveće unutar izabrane kategorije i omogući korisniku kupovinu. Ova komponenta koristi useParams da bi iz URL-a izvukla jedinstveni ID kategorije, koji se koristi u dva odvojena asinhrona poziva. Prvi poziv uzima listu proizvoda iz rute /flower/category/:id, dok drugi, zaseban poziv, služi samo za uzimanje naziva kategorije sa rute /category/:id, koji se zatim koristi za kreiranje dinamičnog naslova na stranici. Glavni interfejs prikazuje mrežu kartica cveća, gde svaka kartica sadrži sliku, naziv, opis i cenu proizvoda. Ključna interaktivna funkcionalnost je dugme "Dodaj u korpu", koje poziva funkciju handleAddToCart. Ova funkcija koristi addToCart prop (prosleđen iz App.js) za ažuriranje globalnog stanja korpe, a zatim privremeno prikazuje popup poruku ("Uspešno dodato u korpu!") koja se automatski skriva nakon 2000 milisekundi, pružajući trenutnu vizuelnu potvrdu korisniku o uspešnom dodavanju artikla. Ukoliko u kategoriji nema proizvoda, korisnik dobija odgovarajuću poruku.

return (

    <div className="categories-page">

      <h2 className="page-title">Naše kategorije</h2>

      <p className="page-subtitle">Odaberite povod i pronađite savršen buket!</p>

      <div className="categories-grid">

        {categories.length > 0 ? (

          categories.map((category) => (

            <div

              key={category.id}

              className="category-card"

              onClick={() => navigate(`/korisnickekategorije/${category.id}`)}

            >

              <div className="category-image-wrapper">

                <img

                    src={categoryImages[category.id.toString()] || "/flowers/default.jpg"}

                    alt={category.name}

                    className="category-image"

                />

              </div>

              <div className="category-info">

                <h3>{category.name}</h3>

                <p>{category.description || "Otkrijte više cvetnih aranžmana"}</p>

              </div>

            </div>

          ))

        ) : (

          <p className="no-categories">Nema dostupnih kategorija 🌸</p>

        )}

      </div>

    </div>

  );

Proizvodi:

import React, { useEffect, useState } from "react";

import { useParams } from "react-router-dom";

import http from "../api/http";

import "../css/FlowerPage.css";

export default function FlowersPage({ addToCart }) {

  const { id } = useParams();

  const [flowers, setFlowers] = useState([]);

  const [categoryName, setCategoryName] = useState("");

  const [catName, setCatName] = useState("");

  const [showPopup, setShowPopup] = useState(false);

  useEffect(() => {

    const fetchFlowers = async () => {

      try {

        const res = await http.get(`/flower/category/${id}`);

        setFlowers(res.data);

        if (res.data.length > 0) setCategoryName(res.data[0].categoryName);

      } catch (err) {

        console.error("Greška pri učitavanju cveća:", err);

      }

    };

    fetchFlowers();

  }, [id]);

  useEffect(() => {

    const fetchCategoryName = async () => {

      try {

        const response = await http.get(`/category/${id}`);

        setCatName(response.data.name);

      } catch (error) {

        console.error("Greška pri učitavanju naziva kategorije:", error);

      }

    };

    fetchCategoryName();

  }, [id]);

  const handleAddToCart = (flower) => {

  addToCart(flower); // koristi funkciju iz App.js

  setShowPopup(true);

  setTimeout(() => setShowPopup(false), 2000);

};

  if (!flowers.length) {

    return (

      <div className="flowers-page">

        <h2 className="flowers-title">Trenutno nema proizvoda u ovoj kategoriji.</h2>

      </div>

    );

  }

  return (

    <div className="flowers-page">

      <h2 className="flowers-title">

        Cveće iz kategorije:{" "}

        <span className="category-name">{catName}</span>

      </h2>

      <div className="flowers-grid">

        {flowers.map((flower) => (

          <div key={flower.id} className="flower-card">

            <img

              src={

                flower.imageUrl ||

                "https://images.unsplash.com/photo-1608538465319-7e7f3c7adf79?auto=format&fit=crop&w=800&q=80"

              }

              alt={flower.name}

              className="flower-image"

            />

            <div className="flower-info">

              <h3>{flower.name}</h3>

              <p>{flower.description}</p>

              <div className="flower-bottom">

                <span className="flower-price">{flower.price} RSD</span>

                <button

                  className="add-to-cart-btn"

                  onClick={() => handleAddToCart(flower)}

                >

                  Dodaj u korpu

                </button>

              </div>

            </div>

          </div>

        ))}

      </div>

      {/\* popup poruka \*/}

      {showPopup && (

        <div className="popup-message">🌷 Uspešno dodato u korpu!</div>

      )}

    </div>

  );

}

### Stranica za kreiranje narudžbine – korisnik

Fajl CartPage.jsx definiše ključnu komponentu koja služi kao korisnička korpa za kupovinu i finalna tačka za slanje narudžbine (checkout). Ova komponenta je zaštićena rutom u App.js i prima tri ključna propa iz globalnog stanja: trenutni sadržaj cart, funkciju za ažuriranje stanja korpe setCart, i userId ulogovanog korisnika.

Glavna svrha komponente je da omogući korisniku pregled, modifikaciju i slanje izabrane robe. Za ovo su definisane tri pomoćne funkcije: calculateTotal koristi reduce metodu da precizno izračuna ukupan iznos korpe sabiranjem cene svakog proizvoda pomnožene s njegovom količinom. UpdateQuantity omogućava korisniku da promeni količinu (quantity) određenog proizvoda, ažurirajući stanje korpe putem setCart. Konačno, removeItem dozvoljava korisniku da izbaci stavku iz korpe filtriranjem niza.

Korisnički interfejs prikazuje kompletan sadržaj korpe u tabelarnom formatu, gde su za svaki proizvod vidljivi naziv, slika, cena i količina, uz kontrole za povećanje/smanjenje količine i uklanjanje. U dnu stranice nalazi se Sekcija Sumarnog Pregleda, koja sadrži polje za Napomenu (note state) koje omogućava korisniku da unese specifične detalje isporuke. Ključna funkcionalnost je funkcija createOrder, koja se pokreće klikom na dugme za slanje narudžbine.

Funkcija createOrder prvo proverava da li je korpa prazna, a zatim konstruiše Data Transfer Objekat (DTO) koji sadrži userId, unetu napomenu (sa fallback vrednošću "poruceno iz reacta" ako je polje prazno) i mapiran niz items sa samo esencijalnim podacima (flowerId, quantity). Nakon slanja POST zahteva ruti /order, komponenta prikazuje poruku o uspehu, briše sadržaj korpe pozivom setCart([]), i preusmerava korisnika na stranicu sa narudžbinama (/narudzbine) koristeći useNavigate, čime se uspešno završava proces kupovine.

import React from "react";

import "../css/Crat.css";

import { useNavigate } from "react-router-dom";

import http from "../api/http";

import  {  useState } from 'react'

const CartPage = ({ cart, setCart, userId }) => {

    // Stanje za napomenu

    const [note, setNote] = useState("");

    // Funkcija za proracun ukupnog iznosa korpe

    const calculateTotal = () => {

        return cart.reduce((sum, item) => sum + (item.price \* (item.quantity || 1)), 0);

    };

    const updateQuantity = (id, newQty) => {

        if (newQty < 1) return;

        setCart(

            cart.map((c) => c.id === id ? { ...c, quantity: newQty } : c)

        );

    };

    const removeItem = (id) => {

        setCart(cart.filter((c) => c.id !== id));

    };

    const createOrder = async () => {

        if (cart.length === 0) return;

        const orderNote = note.trim() === "" ? "poruceno iz reacta" : note.trim(); //logika za napomenu

        const dto = {

            userId,

            napomena: orderNote,

            items: cart.map((c) => ({

                flowerId: c.id,

                quantity: c.quantity || 1,

                unitPrice: c.price,

            })),

        };

        try {

            await http.post("/orders", dto);

            alert("NARUDŽBINA KREIRANA!");

            setCart([]);

            setNote("");

        } catch (err) {

            console.error("GRESKA prilikom kreiranja narudžbine:", err);

            alert("Došlo je do greške. Proverite konzolu.");

        }

    };

    // Prikaz

    return (

        <div className="cart-page-container">

            <h2>Moja Korpa</h2>

            {cart.length === 0 ? (

                <p className="empty-cart-message">🛒 Vaša korpa je trenutno prazna. Dodajte cveće da biste naručili!</p>

            ) : (

                <div className="cart-content-wrapper">

                    {/\* LEVA KOLONA: Lista stavki u korpi \*/}

                    <div className="cart-items-list">

                        {cart.map((c) => (

                            <div key={c.id} className="cart-item-card">

                                {/\* Slika proizvoda  \*/}

                                <div className="item-details">

                                    <h4 className="item-name">{c.name}</h4>

                                    <span className="item-price-per-unit">{c.price} RSD / kom</span>

                                </div>

                                <div className="qty-controls">

                                    <button onClick={() => updateQuantity(c.id, (c.quantity || 1) - 1)}>-</button>

                                    <input

                                        type="number"

                                        min="1"

                                        value={c.quantity || 1}

                                        onChange={(e) => updateQuantity(c.id, parseInt(e.target.value))}

                                        className="qty-input"

                                    />

                                    <button onClick={() => updateQuantity(c.id, (c.quantity || 1) + 1)}>+</button>

                                </div>

                                <div className="item-subtotal">

                                    Ukupno: \*\*{(c.price \* (c.quantity || 1)).toFixed(2)} RSD\*\*

                                </div>

                                <button onClick={() => removeItem(c.id)} className="btn-delete">

                                    &times;

                                </button>

                            </div>

                        ))}

                    </div>

                    {/\* DESNA KOLONA: Rezime narudzbine \*/}

                    <div className="order-summary-box">

                        <h3>Rezime narudžbine</h3>

                        {/\* Polje za napomenu \*/}

                        <div className="napomena-field">

                            <label htmlFor="order-note">Napomena za narudžbinu:</label>

                            <textarea

                                id="order-note"

                                rows="3"

                                value={note}

                                onChange={(e) => setNote(e.target.value)}

                                placeholder="Npr. Isporuka na adresu: Ulica cvetova 5, Vreme isporuke: posle 17h..."

                            ></textarea>

                        </div>

                        <div className="summary-row">

                            <span>Broj stavki:</span>

                            <span>{cart.length}</span>

                        </div>

                        <div className="summary-row total-row">

                            <span>Ukupna cena:</span>

                            <span>{calculateTotal().toFixed(2)} RSD</span>

                        </div>

                        <button

                            onClick={createOrder}

                            className="btn-order-submit"

                        >

                            <i className="fas fa-shopping-bag"></i> Pošalji narudžbinu ({calculateTotal().toFixed(2)} RSD)

                        </button>

                    </div>

                </div>

            )}

        </div>

    );

};

export default CartPage;

### Komponente NavBar I Footer

Oba modula, Navbar.jsx i Footer.jsx, predstavljaju stalne (perzistentne) elemente korisničkog interfejsa, što znači da su vidljivi na svakoj stranici aplikacije jer su pozicionirani izvan bloka Routes u fajlu App.js.

Navbar.jsx – Navigacioni Kontroler

Komponenta Navbar.jsx služi kao dinamički navigacioni kontroler aplikacije. Njena primarna funkcija je da odredi i prikaže odgovarajuće navigacione linkove i akcije na osnovu statusa autentifikacije i uloge korisnika.

Na početku komponente, uzimaju se ključni podaci iz localStorage: token za proveru da li je korisnik ulogovan (isAuth), i user objekat za utvrđivanje njegove role (koja može biti "FLORIST" za administratora ili "USER" za običnog korisnika). Sam navigacioni interfejs se deli na tri sekcije:

* Logo sekcija: Prikazuje ime brenda ("BlossomGarden") i link ka početnoj stranici (/).
* Centralni linkovi: Sadrže nezaštićeni link "Početna" (/), a ostali linkovi se prikazuju uslovno (conditional rendering): Ako je korisnik cvećar (role === "FLORIST"), prikazuju se administrativne rute kao što su "Kategorije", "Proizvodi" i "Narudzbine". Ako je korisnik običan korisnik (role === "USER"), prikazuju se linkovi specifični za kupovinu: "Cveće" (/korisnickekategorije) i "Korpa" (/cart).
* Desna sekcija – Akcije: Ovaj blok se takođe uslovno renderuje. Ako korisnik nije ulogovan (!isAuth), prikazuju se dugmad za Log in i Register. Ako je korisnik ulogovan, prikazuje se poruka dobrodošlice ("Korisnik") i dugme "Log out". Funkcija handleLogout je implementirana direktno u ovom fajlu i zadužena je za brisanje tokena, user podataka (me) i sadržaja korpe (cart) iz localStorage, nakon čega preusmerava korisnika na početnu stranicu.

Footer.jsx – Podnožje Aplikacije

Komponenta Footer.jsx predstavlja statični element na dnu svake stranice, osiguravajući dosledan vizuelni identitet i pružajući esencijalne informacije. Glavni sadržaj podnožja obuhvata ponovljeni prikaz naziva brenda ("BlossomGarden"), centralnu moto poruku ("“Svaki cvet je poruka, mi je samo nežno upakujemo.”"), i obaveznu informaciju o autorskim pravima. Godina za autorska prava (new Date().getFullYear()) se dinamički računa u runtime-u, čime se osigurava da je podatak uvek ažuran. Iako su u kodu ostali zakomentarisani neki linkovi, u trenutnoj implementaciji služi kao čisto vizuelni i informativni blok.

import React from 'react'

export const Footer = () => {

  return (

    <footer id="contact" className="footer">

      <div className="container foot">

        <div className="brand">

          {/\*<div className="dot"></div>\*/}

          <span>BlossomGarden</span>

        </div>

        <p className="moto">

          “Svaki cvet je poruka, mi je samo nežno upakujemo.”

        </p>

        <small>© {new Date().getFullYear()} BlossomGarden</small>

      </div>

    </footer>

  )

}

import React from "react";

import { Link, NavLink, useNavigate } from "react-router-dom";

import "../css/Navbar.css";

import { CgProfile } from "react-icons/cg";

import { IoIosLogIn } from "react-icons/io";

export default function Navbar() {

  const navigate = useNavigate();

  const token = localStorage.getItem("token");

  const user = JSON.parse(localStorage.getItem("me") || "null");

  const role = user?.role; // može biti "FLORIST" ili "USER"

  const isAuth = !!token;

  function handleLogout() {

    localStorage.removeItem("token");

    localStorage.removeItem("me");

    localStorage.removeItem("cart");

    navigate("/");

  }

  return (

    <header className="navbar">

      <div className="nav-inner">

        {/\* Logo sekcija \*/}

        <div className="nav-left">

          <Link to="/" className="logo">

            <img src="/flower.png" alt="logo" className="logo-img" />

            <span className="brand-name">BlossomGarden</span>

          </Link>

        </div>

        {/\* Linkovi \*/}

          <nav className="nav-links">

          <NavLink to="/" className="nav-item">Početna</NavLink>

          {/\* Ako je cvecar \*/}

          {isAuth && role === "FLORIST" && (

           <>

            <NavLink to="/kategorije" className="nav-item">Kategorije</NavLink>

            <NavLink to="/proizvodi" className="nav-item">Proizvodi</NavLink>

            <NavLink to="/narudzbine" className="nav-item">Narudzbine</NavLink>

            </>

           )}

           {/\* Ako je  korisnik \*/}

           {isAuth && role === "USER" && (

           <>

            <NavLink to="/korisnickekategorije" className="nav-item">Cveće</NavLink>

             <NavLink to="/cart" className="nav-item">Korpa</NavLink>

          </>

            )}

          </nav>

        {/\* Desna sekcija – login/register ili user info \*/}

        <div className="nav-actions">

          {!isAuth ? (

            <>

              <Link to="/login" className="btn-login">

                <i className="fa fa-user"></i><IoIosLogIn /> Log in

              </Link>

              <Link to="/register" className="btn-register">

                <i className="fa fa-user-plus"></i><CgProfile /> Register

              </Link>

            </>

          ) : (

            <>

              <span className="welcome">🌷 {user?.username || "Korisnik"}</span>

              <button className="btn-logout" onClick={handleLogout}>

                Odjava

              </button>

            </>

          )}

        </div>

      </div>

    </header>

  );

}

### Axios - povezivanje sa backendom

Axios je JavaScript biblioteka za HTTP zahteve (GET/POST/PUT/DELETE) koju koristimo na front-endu da pričamo sa backend API-jem. Radi na bazi promisa (kao i fetch), ali daje praktičniji API: vraća odgovor u formatu response.data, automatski parsira JSON i ima centralnu podršku za “presretače” (interceptore).

Mi ga biramo jer nam pojednostavljuje ponavljajuće stvari: podesimo baseURL jednom, pa svi pozivi idu relativno; u request interceptoru ubacimo JWT token u Authorization header pre svakog zahteva; u response interceptoru možemo jedinstveno da obradimo greške (npr. 401 → logout). Time čistimo komponente i izbegavamo copy-paste.

Mi najčešće pravimo svoju Axios instancu (npr. http) sa baseURL i interceptorima. To je naš “HTTP sloj” koji sve komponente dele. Ako sutra promenimo adresu API-ja ili način autentikacije, menjamo samo tu instancu – ostatak aplikacije ostaje netaknut. Za instalaciju bismo rekli “npm install axios”, a zatim koristimo instancu kroz pozive tipa “pošalji POST na /auth/login sa kredencijalima i uzmi data iz odgovora”. Ukratko: Axios nam daje čist, dosledan i centralizovan način da radimo sa HTTP-om u React aplikaciji.

import axios from "axios";

const http = axios.create({baseURL:"http://localhost:8080/api", headers: {"Content-Type": "application/json"}, });

//ako u lokalnoj mem imas token uzmi ga i dodaj u karticu za autorizaciju (kao kod postmana)

http.interceptors.request.use((config) => {

    const token = localStorage.getItem("token");

     //if (!config.url.includes("/auth")) {

    if (token) config.headers.Authorization = `Bearer ${token}`;

    return config;

});

export default http;

# Github

Kompletan kod projekta objavljen je na GitHub-u u javnom repozitorijumu: <https://github.com/andjelabokun/mavenproject1>. Repozitorijum sadrži backend (Spring Boot) i frontend (React) deo aplikacije, zajedno sa konfiguracionim fajlovima (npr. application.properties, package.json), DTO/entitetima, kontrolerima, servisima i repozitorijumima, kao i komponentama i stranicama za Florist i User dashboard.

# Zaključak

Implementirana aplikacija ostvaruje ključne ciljeve: jednostavan i brz rad cvećara nad kategorijama, proizvodima i narudžbinama kao i pouzdano kreiranje novih narudžbina od strane korisnika. Primena Spring Boot-a omogućila je brzo podizanje stabilnog REST API-ja sa jasnim granicama odgovornosti i dobrim praksama (autokonfiguracija, JPA repozitorijumi, validacija), dok je React obezbedio modularan i responzivan korisnički interfejs koji se lako razvija i održava. Predložena arhitektura olakšava testiranje, praćenje i nadogradnju sistema, pa su dodavanje novih funkcionalnosti (npr. napredna statistika, kompleksniji izveštaji, dodatne uloge) i eventualna skaliranja moguća bez većih promena u osnovi projekta.

# Literatura

1. VMware, Inc. (2024). Spring Boot Reference Documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/
2. Meta Platforms, Inc. (2024). React Documentation. <https://react.dev/learn>

1. IBM. (n.d.). *Java Spring Boot*. IBM Think. [https://www.ibm.com/think/topics/java- spring-boot](https://www.ibm.com/think/topics/java-spring-boot)
2. Simplilearn. (n.d.). *What is Angular?* [https://www.simplilearn.com/tutorials/angular- tutorial/what-is-angular](https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular)