

# **Programsko inženjerstvo ak.god 2025./2026.**

Sveučilište u Zagrebu  
Fakultet elektrotehnike i računarstva

## **Room-Rently**

Tim: <TG03.2>

Ime tima: OpenReservations

Članovi tima: Jakov Zekić – Josip Mrakovčić – Karlo Živković – Mateo Cerčić –  
Nino Strčić – Noa Rešetar

Nastavnik: Vlado Sruk

Asistent: Miljenko Krhen

Demos: Ivo Gabud

## SADRŽAJ

<b>1. OPIS PROJEKTNOG ZADATKA.....</b>	<b>6</b>
<b>ROOM-RENTLY – MODERNIZACIJA TURIZMA U JEDNOJ APLIKACIJI .....</b>	<b>6</b>
UVOD.....	6
CILJ PROJEKTA .....	6
PROBLEMATIKA I MOTIVACIJA .....	6
POTENCIJALNA KORIST PROJEKTA .....	6
POSTOJEĆA SLIČNA RJEŠENJA .....	7
ŠKUP KORISNIKA .....	7
MOGUĆNOST PRILAGODBE I DALJINJEG RAZVOJA.....	7
OPSEG PROJEKTNOG ZADATKA .....	8
ZAKLJUČAK .....	8
<b>2. ANALIZA ZAHTJEVA .....</b>	<b>8</b>
<b>FUNKCIONALNI ZAHTJEVI .....</b>	<b>8</b>
<b>NEFUNKCIONALNI ZAHTJEVI.....</b>	<b>12</b>
<b>DIONICI SUSTAVA.....</b>	<b>14</b>
AKTORI I NJIHOVI FUNKCIONALNI ZAHTJEVI.....	15
ZAKLJUČAK .....	15
<b>3. SPECIFIKACIJA ZAHTJEVA SUSTAVA.....</b>	<b>15</b>
<b>OBRASCI UPORABE .....</b>	<b>15</b>
<b>DIJAGRAMI OBRAZACA UPORABE.....</b>	<b>16</b>
<b>OPIS OBRAZACA UPORABE .....</b>	<b>18</b>
<b>SEKVENCIJSKI DIJAGRAMI .....</b>	<b>21</b>
<b>PROVJERA UKLJUČENOSTI FUNKCIONALNOSTI U OBRASCE UPORABE .....</b>	<b>24</b>
<b>4. ARHITEKTURA I DIZAJN SUSTAVA .....</b>	<b>24</b>
CILJ POGLAVLJA .....	24
OPIS ARHITEKTURE.....	24
OBRAZLOŽENJE ODABIRA ARHITEKTURE .....	26
ORGANIZACIJA SUSTAVA NA VISOKOJ RAZINI .....	26
ORGANIZACIJA APLIKACIJE .....	26
KLJUČNE KOMPONENTE.....	28
INTERAKCIJA IZMEĐU FRONTENDA I BACKENDA.....	28
<b>BAZA PODATAKA .....</b>	<b>29</b>
OPIS TABLICA .....	29
ER DIJAGRAM BAZE PODATAKA .....	33
<b>DIJAGRAM RAZREDA.....</b>	<b>34</b>
<b>A. POPIS LITERATURE.....</b>	<b>35</b>

BAZA PODATAKA.....	35
FRONTEND .....	35
<b>B. PRIKAZ AKTIVNOSTI GRUPE.....</b>	<b>35</b>
—ZAPISNIK 10.10.2025.— .....	35
—ZAPISNIK 11.10.2025.— .....	35
—ZAPISNIK KRHEN 15.10.2025.— .....	36
—ZAPISNIK 24.10.2025.— .....	37
—ZAPISNIK KRHEN 30.10.2025.— .....	37
—ZAPISNIK 9.11.2025.— .....	38
<b>TABLICA AKTIVNOSTI .....</b>	<b>38</b>
<b>C. DNEVNIK PROMJENA DOKUMENTACIJE .....</b>	<b>40</b>
TABLICA.....	40
<b>DOKUMENTACIJA O OBJAVLJIVANJU APLIKACIJE NA JAVNOM POSLUŽITELJU .....</b>	<b>41</b>
SAŽETAK .....	41
PROMJENE I PROBLEMI TIJEKOM RAZVOJA .....	41
<b>PRIPREMA BACKENDA .....</b>	<b>41</b>
<b>PRIPREMA FRONTENDA .....</b>	<b>43</b>
<b>DEPLOYMENT BACKENDA .....</b>	<b>43</b>
KORAK 1: POSTAVLJANJE VARIJABLI OKRUŽENJA .....	43
KORAK 2: PAKIRANJE APLIKACIJE .....	43
KORAK 3: IZRADA DOCKER SLIKE .....	43
KORAK 4: DIJELJENJE DOCKER SLIKE NA DOCKER HUBU .....	44
KORAK 5: POSTAVLJANJE NA RENDER.COM .....	44
KORAK 1: AŽURIRANJE URL-A BACKENDA U FRONTENDU .....	44
KORAK 2: PRIPREMA FRONTENDA ZA PRODUCTION .....	44
KORAK 3: DEPLOYMENT FRONTENDA NA NETLIFY.....	45
KORAK 4: AŽURIRANJE BACKENDA NA RENDERU .....	45
KORAK 5: KONFIGURACIJA ENVIRONMENT VARIJABLI NA NETLIFY .....	45
<b>DEPLOYMENT BAZE PODATAKA.....</b>	<b>45</b>
KORAK 1: STVARANJE SUPABASE RAČUNA .....	45
KORAK 2: STVARANJE BAZE PODATAKA.....	45
KORAK 3: SPAJANJE SUPABASE BAZE I BACKENDA NA RENDER-U .....	45
KORAK 4: PROVJERA BAZE.....	46
<b>TIJEK IMPLEMENTIRANJA BACKEND.....</b>	<b>46</b>
<b>BACKEND — SPRING BOOT IMPLEMENTACIJA.....</b>	<b>46</b>
CILJ .....	46
POKRETANJE BACKEND APLIKACIJE .....	47
POSTAVLJANJE I POKRETANJE FRONTENDA .....	47
STRUKTURA BACKEND DIJELA PROJEKTA .....	48

PERSON MODEL (MODEL/PERSON.JAVA).....	48
PERSON CONTROLLER (CONTROLLER/PERSONCONTROLLER.JAVA) .....	49
PERSONREPO (REPOSITORY/PERSONREPO.JAVA) .....	51
SECURITY KONFIGURACIJA (CONFIG/SECURITYCONFIG.JAVA) .....	53
DATOTEKA APPLICATION.PROPERTIES.....	55
CJELOKUPNI TIJEK AUTENTIFIKACIJE (FRONTEND ↔ BACKEND) .....	55
ZAKLJUČAK .....	55
<b>TIJEK IMPLEMENTIRANJA BAZE PODATAKA .....</b>	<b>56</b>
<b>POČETAK INICIJALIZIRANJA BAZE PODATAKA (INSTALACIJE) .....</b>	<b>56</b>
<b>RELACIJE.....</b>	<b>56</b>
RELACIJA UNIT.....	57
<b>REPOZITORIJI .....</b>	<b>62</b>
PERSONREPO .....	62
UNITREPO .....	63
UNITRESERVATIONREPO .....	63
UNITIMGREPO.....	63
UNITCONTROLLER.....	65
UNITRESERVATIONCONTROLLER.....	67
UNITIMGCONTROLLER.....	68
<b>TIJEK IMPLEMENTIRANJA FRONTEND .....</b>	<b>69</b>
<b>POČETAK PROJEKTA(INSTALACIJE).....</b>	<b>69</b>
<b>POČETNA STRANICA(LANDING SCREEN) .....</b>	<b>70</b>
REACT + CSS ANIMACIJA .....	70
<b>DODAVANJE OSTALIH STRANICA TE NJIHOVIH RUTA .....</b>	<b>75</b>
<b>GLAVNA STRANICA.....</b>	<b>76</b>
DODAVANJE NAVIGACIJSKE TRAKE .....	76
IZRADA ZAGLAVLJA .....	84
KOMPONENTE KOJE SE NALAZE NA SREDINI GLAVNE STRANICE .....	102
KOMPONENTE PODNOŽJA STRANICE .....	114
<b>STRANICA LIST .....</b>	<b>120</b>
KOMPONENTA SEARCHITEM KOJA SE KORISTI NA STRANICI LIST .....	129
<b>STRANICA HOTEL .....</b>	<b>138</b>
UVEĆANI PRIKAZ SLIKA TE POMICANJE PO SLIKAMA .....	138
GOOGLE MAPS .....	141
CSS STIL.....	142
<b>STRANICA FORM ZA UNOS I UREĐIVANJE SMJEŠTAJNIH JEDINICA .....</b>	<b>149</b>
<b>APARTMENTFORM CSS .....</b>	<b>156</b>

<b>KOMPONENTA RENTALUNITS.JSX.....</b>	<b>161</b>
<b>RENTALUNITS.CSS .....</b>	<b>164</b>
<b>TIJEK IMPLEMENTIRANJA FULLSTACK .....</b>	<b>167</b>
<b>DODAVANJE CRUD OPERACIJA ZA SMJEŠTAJNE JEDINICE .....</b>	<b>167</b>
POVEZIVANJE SUBMITA I BACKENDA/BAZE.....	167
POVEZIVANJE FRONTEND PRIKAZA SMJEŠTAJNIH JEDINICA S BAZOM.....	169
DELETE FUNKCIONALNOST NA LISTI JEDINICA .....	178
UPDATE FUNKCIONALNOST ZA JEDINICE.....	178
UREĐIVANJE /ADMIN PAGEA (SITNI FIXOVI) .....	182
<b>USER LISTA NA ADMIN PAGEU.....</b>	<b>187</b>
PODIJELITI ADMIN PAGE NA UNIT I USERS TABOVE.....	187
FUNKCIONALNOST BRISANJA KORISNIKA .....	198
<b>NETLIFY BUGOVI I RESPONZIVNOST ADMIN I FORM PAGEA .....</b>	<b>199</b>
NETLIFY BUGOVI.....	199
<b>IZJAVA O KORIŠTENJU ALATA UMJETNE INTELIGENCIJE .....</b>	<b>204</b>

# 1. OPIS PROJEKTNOG ZADATKA

## ROOM-RENTLY – MODERNIZACIJA TURIZMA U JEDNOJ APLIKACIJI

### UVOD

**Room-Rently** je web aplikacija razvijena s ciljem modernizacije procesa rezervacije i upravljanja smještajem unutar jednog hotela ili objekta.

Ideja projekta je stvoriti **jednostavan, pouzdan i pregledan sustav** koji povezuje goste s osobljem hotela te olakšava svakodnevno poslovanje kroz digitalizaciju rezervacija i evidencija.

Aplikacija omogućuje **brzu i sigurnu rezervaciju soba**, jednostavno upravljanje dostupnošću te automatsko vođenje statistike i izvještaja.

Room-Rently tako predstavlja digitalnu alternativu klasičnim načinima vođenja rezervacija, uz naglasak na **učinkovitost, preglednost i profesionalno korisničko iskustvo**.

### CILJ PROJEKTA

Glavni cilj **Room-Rently** sustava je **pojednostaviti i unaprijediti način** na koji hotel ili iznajmljivač upravlja svojim smještajem i rezervacijama.

Aplikacija gostima omogućuje jednostavnu pretragu i rezervaciju soba, dok osoblju pruža pregledan alat za upravljanje smještajnim jedinicama i praćenje zauzeća.

**Osnovne ideje projekta su:** - digitalizirati proces rezervacija unutar hotela ili objekta,  
- smanjiti mogućnost grešaka i preklapanja termina,  
- poboljšati komunikaciju između gostiju i osoblja,  
- omogućiti analitički uvid u poslovanje i zauzeće soba.

### PROBLEMATIKA I MOTIVACIJA

Mnogi manji hoteli i privatni iznajmljivači još uvijek koriste ručne metode rezervacija putem telefona, e-maila ili bilježnica, što često dovodi do zabuna, dvostrukih rezervacija i gubitka informacija.

Globalne platforme poput Booking.com-a ili Airbnb-a nude napredna rješenja, ali su često prekompleksna, uz visoke provizije i nepotrebne funkcionalnosti za objekte koji žele samostalno upravljati svojim kapacitetima.

**Room-Rently** je zamišljen kao **interni sustav za jedan hotel**, koji omogućuje potpunu kontrolu nad rezervacijama i smještajem bez posrednika i dodatnih troškova.

### POTENCIJALNA KORIST PROJEKTA

#### ZA GOSTE

- Jednostavna pretraga i online rezervacija soba.
- Brza i sigurna potvrda rezervacije putem e-maila.
- Mogućnost prijave putem Google računa bez dodatne registracije.

---

#### ZA HOTEL / IZNAJMLJIVAČA

- Centralizirano upravljanje svim sobama i rezervacijama.
- Automatska evidencija zauzeća i dolazaka gostiju.
- Izrada izvještaja i statistika o poslovanju (u PDF ili Excel formatu).
- Manje administracije i bolja organizacija recepcionskog osoblja.

---

#### ZA ADMINISTRATORE SUSTAVA

- Kontrola pristupa i ovlasti korisnika.
- Uvid u sve rezervacije i korisničke aktivnosti.
- Generiranje analitičkih izvještaja o radu sustava.

#### POSTOJEĆA SLIČNA RJEŠENJA

Na tržištu postoje brojni sustavi za hotelsko poslovanje (PMS – Property Management System), no **Room-Rently** se razlikuje po: - jednostavnom i modernom sučelju prilagođenom manjim objektima, - potpunoj neovisnosti o vanjskim platformama, - lokalnoj prilagodbi i podršci, - mogućnosti integracije s Google Maps API-jem i lokalnim platnim servisima.

Ovakav pristup omogućuje hotelima i iznajmljivačima da imaju **vlastiti sustav rezervacija**, bez dodatnih troškova i provizija.

#### SKUP KORISNIKA

U sustavu **Room-Rently** definirane su tri osnovne korisničke uloge:

- **Korisnik / Gost** – može pregledavati dostupne sobe, izvršiti rezervaciju, primiti potvrdu putem e-maila te upravljati vlastitim rezervacijama.
- **Vlasnik objekta** – upravlja smještajnim jedinicama (unos, uređivanje, brisanje), pregledava rezervacije, zauzeće i izvještaje o poslovanju.
- **Administrator** – ima puni pristup sustavu, nadzire sve korisnike i rezervacije, dodjeljuje ovlasti te održava stabilnost i sigurnost aplikacije.

#### MOGUĆNOST PRILAGODE I DALJNJE RAZVOJA

Sustav je razvijen modularno, što omogućuje jednostavno proširenje funkcionalnosti.

U budućnosti se može nadograditi s: - mobilnom aplikacijom za goste i osoblje, - dinamičnim određivanjem cijena na temelju potražnje,

- integracijom s pametnim bravama (IoT) za automatsku prijavu gostiju,
- naprednim AI preporukama i analitikom poslovanja.

## OPSEG PROJEKTNOG ZADATKA

Projekt obuhvaća razvoj web aplikacije koja omogućuje:

- prijavu korisnika i upravljanje korisničkim ulogama,
- pregled, unos i uređivanje smještajnih jedinica,
- online rezervaciju i slanje potvrda putem e-maila,
- prikaz statistika zauzeća i izvještaja,
- responzivan dizajn i prikaz lokacije hotela putem Google Maps servisa.

## ZAKLJUČAK

**Room-Rently** donosi modernizaciju hotelskog poslovanja kroz jednostavnu, učinkovitu i prilagodljivu aplikaciju.

Kao interni sustav za jedan hotel ili iznajmljivača, omogućuje digitalizaciju rezervacija, bolju organizaciju i profesionalniji pristup gostima bez potrebe za vanjskim platformama ili provizijama.

## 2. ANALIZA ZAHTJEVA

### FUNKCIONALNI ZAHTJEVI

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
F-001	Sustav omogućuje autentifikaciju korisnika putem OAuth2 protokola (Google Sign-In).	Visok	Korisnik se može prijaviti i registrirati putem Google računa te uspješno pristupiti sustavu.
F-002	Sustav razlikuje tri uloge korisnika : Gost, Voditelj hotela i Administrator.	Visok	Nakon prijave, korisniku se dodjeljuje odgovarajuća uloga i prikazuju funkcionalnosti prema ovlastima.
F-003	Middleware sloj provjerava korisničku ulogu	Visok	Neovlašteni korisnici nemaju pristup zaštićenim funkcijama.

	pristup API rutama.		
F-004	Sustav omogućuje unos, izmjenu, dohvati i brisanje smještajnih jedinica (CRUD operacije).	Visok	Voditelj hotela može uspješno unositi i uređivati smještajne jedinice.
F-005	Svaka smještajna jedinica sadrži atribute: naziv, opis, broj kreveta, cijenu, tip, dostupnost i dodatne usluge.	Visok	Podaci se ispravno prikazuju i spremaju u bazu.
F-006	Sustav validira unos podataka (npr. cijena > 0, broj kreveta ≥ 1).	Visok	Pogrešni podaci ne mogu biti spremljeni.
F-007	API omogućuje filtriranje smještajnih jedinica prema cijeni, tipu i dostupnosti.	Srednji	Rezultati pretrage odgovaraju odabranim filtima.

F-008	Sustav omogućuje stvaranje , pregled, izmjenu i otkazivanje rezervacija (CRUD operacije ).	Visok	Gost može uspješno rezervirati i otkazati smještaj.
F-009	Sustav provjerava dostupnost smještaja i sprječava preklapanje termina.	Visok	Nije moguće izvršiti rezervaciju za već zauzet termin.
F-010	Sustav generira potvrdu rezervacije u PDF formatu i šalje e-mail korisniku .	Visok	Nakon potvrde rezervacije, korisnik prima e-mail s PDF dokumentom.
F-011	Sustav omogućuje pregled i izvoz statistike o zauzeću, gostima i popularnosti usluga.	Srednji	Administrator i voditelj hotela mogu generirati izvještaje u PDF, XLSX ili XML formatu.
F-012	Sustav omogućuje prikaz smještaj	Srednji	Lokacija smještaja ispravno se prikazuje na mapi.

	a na interaktivnoj karti putem Google Maps integracije.		
F-013	Sustav prikazuje dinamičnu naslovnu stranicu s tražilicom, kategorijama i istaknutim smještajima.	Srednji	Glavna stranica prikazuje ažurirane podatke i omogućuje pretragu.
F-014	Gost može putem tražilice filtrirati ponudu po lokaciji, datumu, cijeni i broju osoba.	Visok	Pretraga prikazuje relevantne rezultate.
F-015	Sustav šalje e-mail obavijesti o novim rezervacijama i promjenama statusa.	Visok	Korisnici dobivaju pravovremene obavijesti putem e-maila.

## NEFUNKCIONALNI ZAHTJEVI

### ZAHTJEVI PERFORMANSI

ID zahtjeva	Opis	Prioritet
NF-1.1	Stranica treba učitati u razumnom roku, maksimalno do 10 sekundi.	Visok
NF-1.2	Sustav mora podržavati više istovremenih korisnika bez usporavanja rada.	Visok
NF-1.3	Pretraga smještajnih jedinica mora se izvršiti unutar nekoliko sekundi.	Srednji

### ZAHTJEVI SIGURNOSTI

ID zahtjeva	Opis	Prioritet
NF-2.1	Prijava i registracija provode se putem sigurnog OAuth2 protokola.	Visok
NF-2.2	Komunikacija između klijenta i poslužitelja mora se odvijati putem HTTPS protokola.	Visok
NF-2.3	Samo ovlašteni korisnici imaju pristup administracijsko m dijelu sustava.	Visok
NF-2.4	Podaci o korisnicima i rezervacijama moraju biti šifrirani i zaštićeni od neovlaštenog pristupa.	Visok

---

#### ZAHTJEVI UPOTREBLJIVOSTI

ID zahtjeva	Opis	Prioritet
NF-3.1	Sučelje aplikacije mora biti jednostavno i pregledno.	Visok
NF-3.2	Sustav mora biti responzivan i ispravno se prikazivati na svim uređajima.	Visok
NF-3.3	Navigacija mora biti intuitivna i laka za korištenje.	Visok

---

#### ZAHTJEVI POUZDANOSTI I DOSTUPNOSTI

ID zahtjeva	Opis	Prioritet
NF-4.1	Sustav treba biti dostupan korisnicima bez čestih prekida rada.	Visok
NF-4.2	Podaci o rezervacijama moraju se redovito arhivirati.	Visok
NF-4.3	U slučaju greške, sustav se mora brzo oporaviti i vratiti u stabilno stanje.	Visok

---

#### ZAHTJEVI ODRŽAVANJA I PROŠIRIVOSTI

ID zahtjeva	Opis	Prioritet
NF-5.1	Sustav mora biti razvijen modularno radi lakšeg dodavanja novih funkcionalnosti.	Srednji
NF-5.2	Programska dokumentacija mora biti uredna i ažurna.	Visok
NF-5.3	Kód treba biti pisan prema dogovorenim	Visok

	standardima i dobro komentiran.	
--	---------------------------------	--

---

#### ZAHTEVI KOMPATIBILNOSTI

ID zahtjeva	Opis	Prioritet
NF-6.1	Aplikacija mora raditi na najčešće korištenim preglednicima (Chrome, Firefox, Edge, Opera GX).	Visok
NF-6.2	Sustav mora podržavati izvoz podataka u PDF, XML i XLSX formatu.	Visok
NF-6.3	Integracija s Google Maps servisom mora omogućiti prikaz lokacije hotela.	Visok

---

#### ZAHTEVI SKALABILNOSTI

ID zahtjeva	Opis	Prioritet
NF-7.1	Sustav mora omogućiti dodavanje novih korisnika i smještajnih jedinica bez pada performansi.	Visok
NF-7.2	Baza podataka mora biti pripremljena za povećanje količine zapisa u budućnosti.	Srednji

#### DIONICI SUSTAVA

Dionik	Opis uloge
Gost	Krajnji korisnik koji pretražuje i rezervira smještaj.
Voditelj hotela	Upravlja smještajnim jedinicama, pregledavaju rezervacije i statistiku.
Administrator	Nadgleda cijeli sustav, kontrolira korisnike i osigurava sigurnost i stabilnost rada.
Razvojni tim	Odgovoran za implementaciju, održavanje i daljnji razvoj

	aplikacije.
--	-------------

## AKTORI I NJIHOVI FUNKCIONALNI ZAHTJEVI

ID aktora	Uloga i opis	Funkcionalnosti
A-1	<b>Gost (inicijator)</b>	Registracija i prijava putem Google Sign-In (F-001), pregled i filtriranje smještaja (F-007, F-014), rezervacija i otkazivanje (F-008, F-009), primanje e-mail potvrda (F-010).
A-2	<b>Voditelj hotela (inicijator)</b>	Upravljanje smještajnim jedinicama (F-004, F-005, F-006), pregled i potvrda rezervacija (F-008, F-009), generiranje izvještaja (F-011).
A-3	<b>Administrator (sudionik)</b>	Upravljanje korisnicima i nadzor nad sustavom (F-002, F-003, F-011, F-015), pristup svim podacima i statistici.

## ZAKLJUČAK

Analiza zahtjeva obuhvaća sve funkcionalne i nefunkcionalne potrebe Room-Rently sustava.

Funkcionalni zahtjevi definiraju osnovne procese i interakcije korisnika sa sustavom, dok nefunkcionalni zahtjevi određuju standarde kvalitete, sigurnosti i izvedbe koji su nužni za pouzdan i skalabilan rad aplikacije.

## 3. SPECIFIKACIJA ZAHTJEVA SUSTAVA

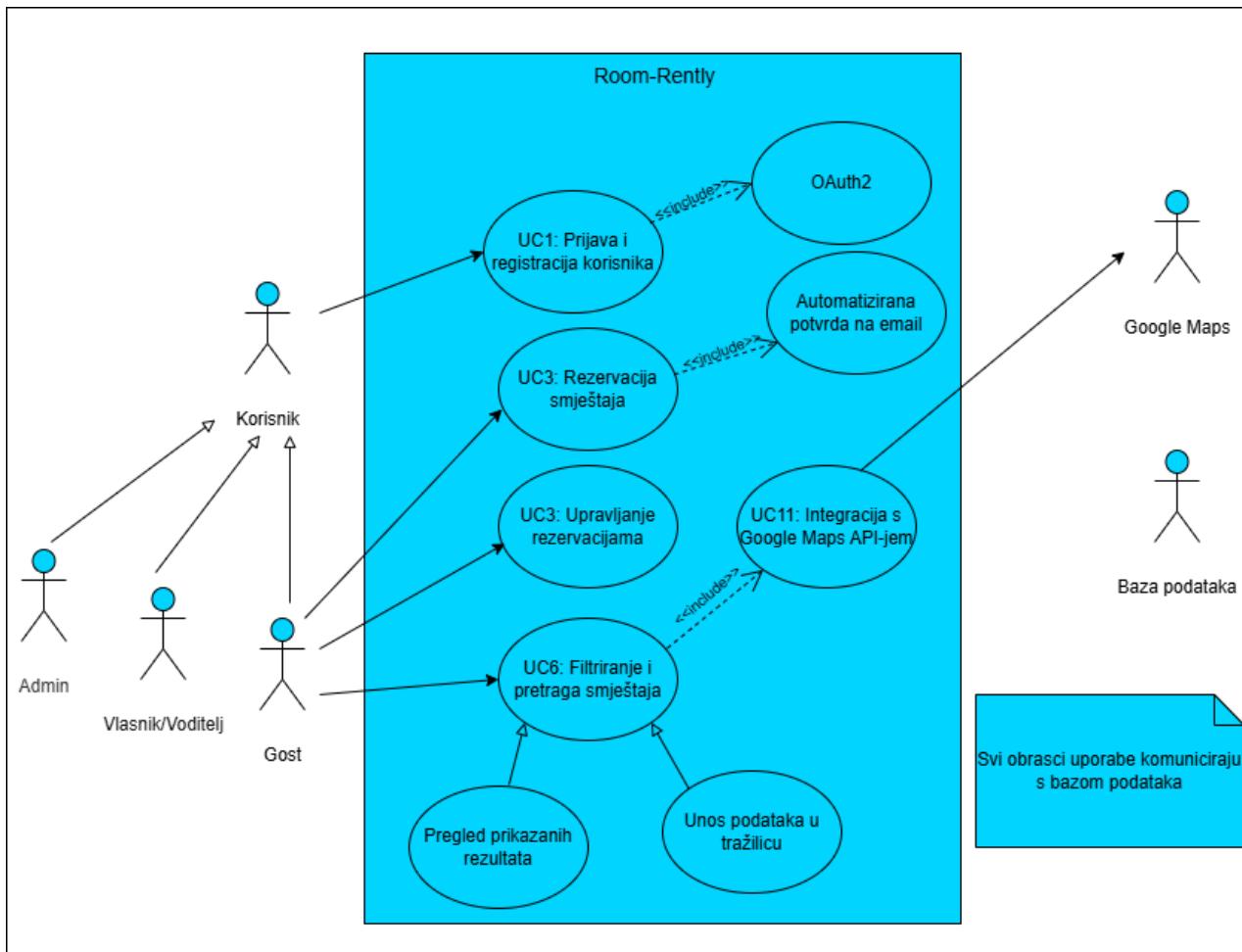
### OBRASCI UPORABE

Kategorija	Use Caseovi
<b>1. Visokorazinski dijagram obrazaca uporabe cijelog sustava</b>	UC1: Prijava i registracija korisnika UC2: Upravljanje smještajem UC3: Upravljanje rezervacijama UC4: Generiranje i slanje potvrda rezervacije UC5: Pregled izvještaja i statistike
<b>2. Dijagram obrazaca uporabe za ključne funkcionalnosti</b>	UC6: Filtriranje i pretraga smještaja UC7: Upravljanje dostupnošću i terminima UC8: Upravljanje korisnicima UC9: Slanje e-mail obavijesti UC10: Optimizacija zauzeća smještaja
<b>3. Dijagram obrazaca uporabe za korisničke uloge</b>	UC1: Prijava i registracija korisnika UC2: Upravljanje smještajem (Administrator) UC3: Upravljanje rezervacijama (Gost) UC5: Pregled izvještaja i statistike (Vlasnik/Voditelj) UC7: Upravljanje dostupnošću i terminima (Vlasnik/Voditelj) UC8: Upravljanje korisnicima (Administrator)
<b>4. Dijagram obrazaca uporabe za osnovne poslovne procese</b>	UC3: Upravljanje rezervacijama UC4: Generiranje i slanje potvrda rezervacije UC5: Pregled izvještaja i statistike UC10: Optimizacija zauzeća smještaja
<b>5. Dijagram obrazaca uporabe za kritične sustave i</b>	UC1: Prijava i registracija korisnika UC4: Generiranje i

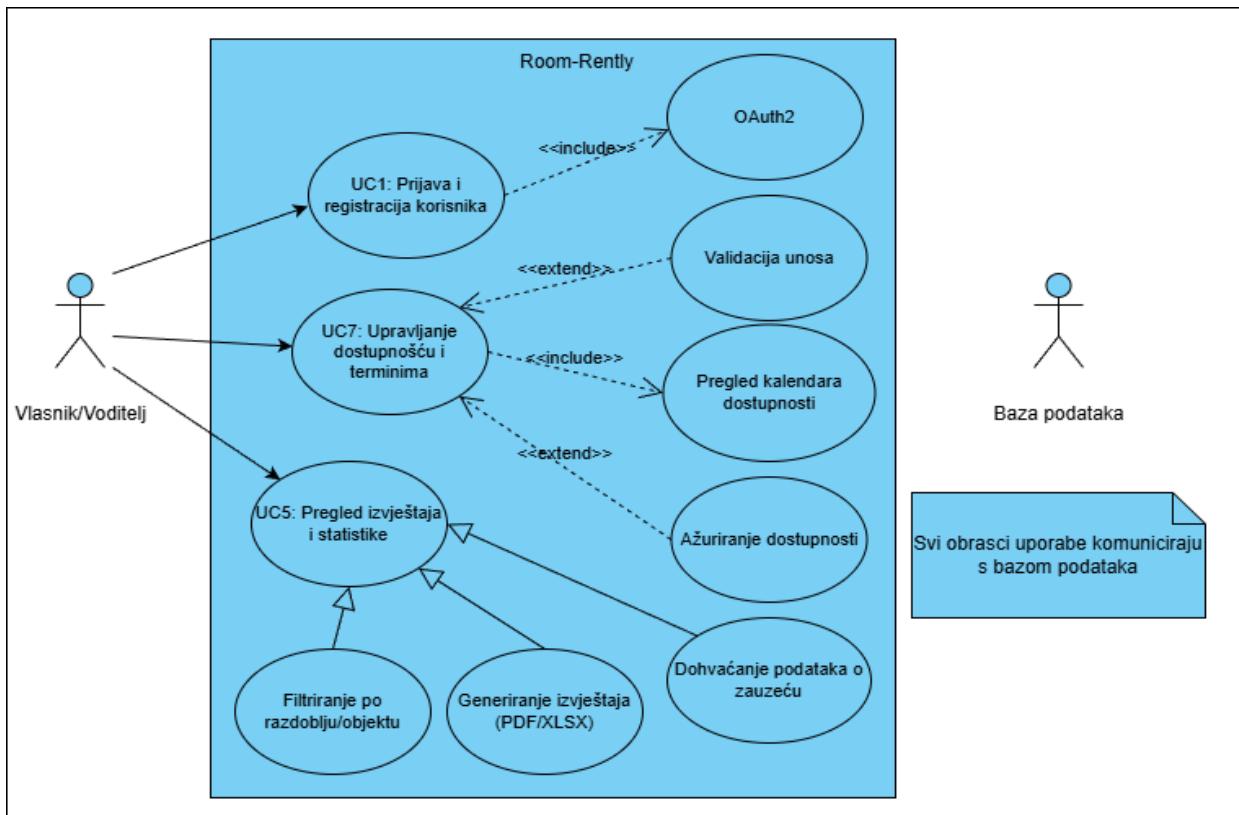
Kategorija	Use Caseovi
integracije	slanje potvrda rezervacije UC11: Integracija s Google Maps API-jem

## DIJAGRAMI OBRAZACA UPORABE

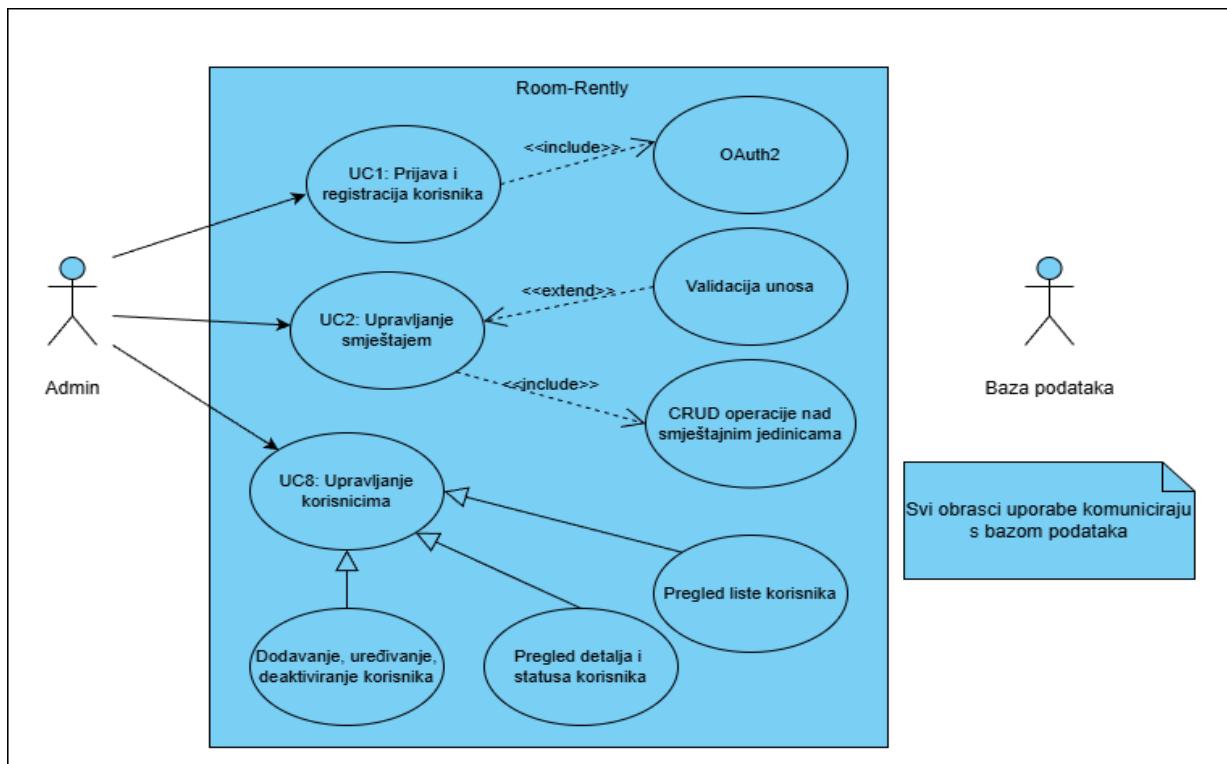
### DIJAGRAM OBRAZACA UPORABE OPĆENITO



Slika 3.1 Dijagram obrazaca uporabe općenito



Slika 3.2 Funkcionalnosti Vlasnika/Voditelja



Slika 3.3 Funkcionalnosti administratora

## OPIS OBRAZACA UPORABE

---

### UC1: PRIJAVA I REGISTRACIJA KORISNIKA

**Glavni sudionik:** Korisnik (Gost, Vlasnik/Voditelj, Administrator)

**Cilj:** Omogućiti korisniku siguran pristup sustavu putem OAuth2 (Google Sign-In).

**Sudionici:** Korisnik, Sustav, Google API

**Preduvjet:** Korisnik posjeduje Google račun.

**Opis osnovnog tijeka:** 1. Korisnik otvara početnu stranicu i bira opciju "Prijava" ili "Registracija" (F-001).

2. Sustav otvara OAuth2 autentifikacijski prozor.

3. Korisnik potvrđuje prijavu svojim Google računom.

4. Sustav provjerava postoje li podaci u bazi i kreira novi račun ako ne postoji.

5. Korisniku se dodjeljuje uloga (Gost, Vlasnik/Voditelj, Administrator) i preusmjerava ga na pripadajuće sučelje (F-002).

**Moguća odstupanja:**

- Ako autentifikacija ne uspije, korisnik dobiva obavijest o grešci i može pokušati ponovo.

- Ako Google API nije dostupan, sustav nudi privremenu lokalnu prijavu.

---

### UC2: UPRAVLJANJE SMJEŠTAJEM

**Glavni sudionik:** Administrator

**Cilj:** Omogućiti unos, izmjenu, brisanje i pregled smještajnih jedinica.

**Sudionici:** Administrator, Sustav

**Preduvjet:** Administrator je prijavljen u sustav.

**Opis osnovnog tijeka:** 1. Administrator otvara panel "Smještajne jedinice" (F-004).

2. Sustav prikazuje popis postojećih smještajnih jedinica.

3. Administrator dodaje novu jedinicu i unosi podatke (naziv, opis, broj kreveta, cijena, tip, usluge) (F-005).

4. Sustav validira unos (F-006) i pohranjuje ga u bazu.

5. Administrator može uređivati ili brisati postojeće jedinice.

**Moguća odstupanja:**

- Ako unos nije valjan, sustav javlja poruku o pogrešci.

- Ako jedinica ima aktivne rezervacije, brisanje nije moguće.

---

### UC3: UPRAVLJANJE REZERVACIJAMA

**Glavni sudionik:** Gost

**Cilj:** Omogućiti stvaranje, izmjenu i otkazivanje rezervacija.

**Sudionici:** Gost, Sustav

**Preduvjet:** Gost je prijavljen i odabrao smještaj.

**Opis osnovnog tijeka:** 1. Gost odabire smještaj i klikne "Book Now" (F-008).

2. Sustav prikazuje formu za odabir datuma i unosa osobnih podataka.

3. Sustav provjerava dostupnost (F-009).

4. Ako je smještaj slobodan, rezervacija se pohranjuje u bazu.

5. Sustav generira potvrdu rezervacije i šalje e-mail korisniku (F-010).

**Moguća odstupanja:**

- Ako termin nije dostupan, sustav nudi alternativne datume.
- Ako korisnik ne potvrdi rezervaciju, ona se ne pohranjuje.

---

**UC4: GENERIRANJE I SLANJE POTVRDA REZERVACIJE****Glavni sudionik:** Sustav

**Cilj:** Automatski generirati i poslati potvrdu rezervacije e-mailom.

**Sudionici:** Sustav, Korisnik

**Preduvjet:** Rezervacija je uspješno kreirana.

**Opis osnovnog tijeka:** 1. Sustav prepoznaje novu potvrđenu rezervaciju (F-010).

2. Generira PDF dokument s detaljima (hotel, period, cijena, usluge).

3. Šalje e-mail korisniku s privitkom.

4. Pohranjuje potvrdu u bazu podataka.

**Moguća odstupanja:**

- Ako slanje e-maila ne uspije, sustav pokušava ponovno ili obavještava administratora.

---

**UC5: PREGLED IZVJEŠTAJA I STATISTIKE****Glavni sudionik:** Vlasnik/Voditelj

**Cilj:** Pregledati podatke o zauzeću, gostima i uslugama te izvoziti izvještaje.

**Sudionici:** Vlasnik/Voditelj, Sustav

**Preduvjet:** Korisnik ima ulogu Vlasnika/Voditelja.

**Opis osnovnog tijeka:** 1. Korisnik otvara modul "Statistika" (F-011).

2. Sustav dohvaća podatke o rezervacijama i zauzeću jedinica.

3. Korisnik filtrira izvještaj po periodu.

4. Sustav prikazuje rezultate i nudi izvoz u PDF/XLSX/XML.

**Moguća odstupanja:**

- Ako nema dostupnih podataka, sustav prikazuje obavijest.

---

**UC6: FILTRIRANJE I PRETRAGA SMJEŠTAJA****Glavni sudionik:** Gost

**Cilj:** Pretražiti smještaj prema odabranim kriterijima.

**Sudionici:** Gost, Sustav

**Preduvjet:** Gost se nalazi na glavnoj stranici sustava.

**Opis osnovnog tijeka:** 1. Gost unosi lokaciju, datume i broj osoba u tražilicu (F-007, F-014).

2. Sustav filtrira dostupne jedinice.

3. Gost pregledava prikazane rezultate.

**Moguća odstupanja:**

- Ako nema rezultata, sustav prikazuje poruku o nedostupnosti smještaja.

---

**UC7: UPRAVLJANJE DOSTUPNOŠĆU I TERMINIMA**

**Glavni sudionik:** Voditelj hotela

**Cilj:** Ažurirati dostupnost smještajnih jedinica.

**Sudionici:** Voditelj, Sustav

**Preduvjet:** Voditelj je prijavljen.

**Opis osnovnog tijeka:** 1. Voditelj otvara kalendar dostupnosti (F-009).

2. Odabire datume i označava zauzete termine.

3. Sustav sprema promjene u bazu.

**Moguća odstupanja:**

- Ako termin ima postojeću rezervaciju, izmjena nije dopuštena.

---

#### UC8: UPRAVLJANJE KORISNICIMA

**Glavni sudionik:** Administrator

**Cilj:** Upravljati korisničkim računima i pristupom sustavu.

**Sudionici:** Administrator, Sustav

**Preduvjet:** Administrator je prijavljen.

**Opis osnovnog tijeka:** 1. Administrator otvara listu korisnika (F-002, F-003).

2. Pregledava detalje i statuse korisnika.

3. Dodaje, uređuje ili deaktivira korisnike.

**Moguća odstupanja:**

- Ako korisnik ima aktivne rezervacije, sustav blokira brisanje.

---

#### UC9: SLANJE E-MAIL OBAVIJESTI

**Glavni sudionik:** Sustav

**Cilj:** Obavijestiti korisnike o promjenama rezervacija.

**Sudionici:** Sustav, Korisnik

**Preduvjet:** Postoji događaj koji zahtjeva obavijest.

**Opis osnovnog tijeka:** 1. Sustav detektira događaj (potvrda, promjena, otkaz).

2. Generira e-mail s relevantnim informacijama (F-015).

3. Šalje obavijest korisniku.

**Moguća odstupanja:**

- Ako slanje ne uspije, sustav ponavlja pokušaj.

---

#### UC10: OPTIMIZACIJA ZAUZEĆA SMJEŠTAJA

**Glavni sudionik:** Sustav

**Cilj:** Predložiti optimalan raspored rezervacija radi smanjenja praznih dana.

**Sudionici:** Sustav, Voditelj hotela

**Preduvjet:** Postoje aktivne rezervacije.

**Opis osnovnog tijeka:** 1. Sustav analizira raspored rezervacija (F-009).

2. Predlaže optimizirane datume zauzeća.

3. Voditelj može prihvati prijedlog i ažurirati kalendar.

### Moguća odstupanja:

- Ako nema alternativnih termina, sustav zadržava postojeći raspored.

## UC11: INTEGRACIJA S GOOGLE MAPS API-JEM

**Glavni sudionik:** Sustav

**Cilj:** Prikazati lokaciju smještaja na interaktivnoj karti.

**Sudionici:** Sustav, Korisnik, Google Maps API

**Preduvjet:** Smještaj ima spremljene koordinate.

**Opis osnovnog tijeka:** 1. Korisnik otvara stranicu smještaja (F-012).

2. Sustav dohvata koordinate iz baze.

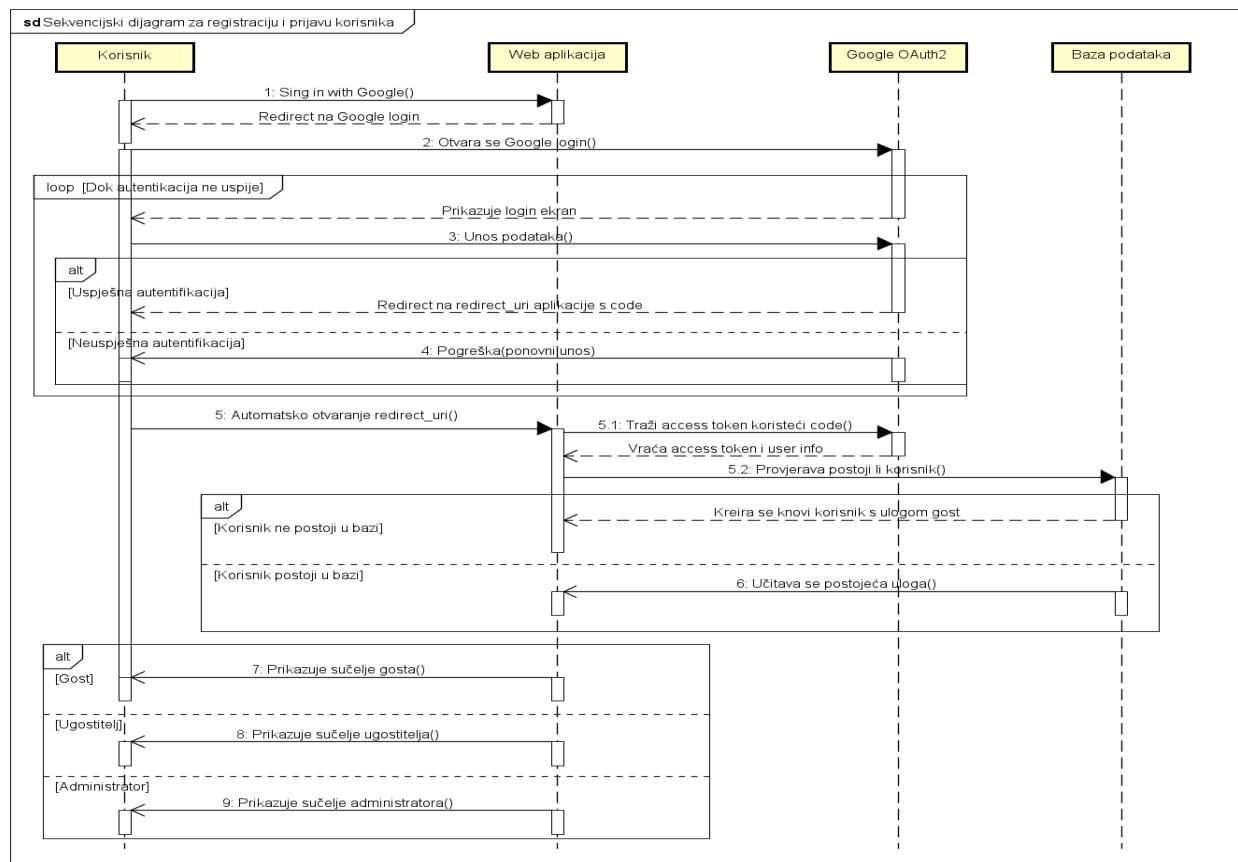
3. Prikazuje lokaciju putem Google Maps API integracije.

### Moguća odstupanja:

- Ako API ne odgovara, prikazuje se statična slika karte.

## SEKVENCIJSKI DIJAGRAMI

### SEKVENCIJSKI DIJAGRAM ZA REGISTRACIJU I PRIJAVU KORISNIKA (UC1)



Slika 3.4 Sekvencijski dijagram za registraciju i prijavu korisnika

Ovaj sekvencijski dijagram prikazuje proces registracije i prijave korisnika putem Google OAuth2 autentifikacije. U njemu sudjeluju četiri glavna sudionika (lifeline-a): korisnik, Web aplikacija, Google OAuth2 i baza podataka.

---

#### KORISNIK POKREĆE PRIJAVU

Korisnik na web aplikaciji klikne "Sign in with Google" te ga aplikacija preusmjerava (redirect) na Googleovu stranicu za prijavu.

---

#### GOOGLE LOGIN STRANICA

Otvara se Google login ekran u kojem korisnik unosi svoje Google podatke za prijavu (email, lozinku). Ovaj dio je unutar petlje (loop) te ako autentifikacija ne uspije, korisnik ponovno unosi podatke dok ne uspije.

---

#### AUTENTIFIKACIJA (ALT GRANA)

Dijagram ima mogućnosti uspješne i neuspješne autentifikacije. Kod uspješne autentifikacije Google vraća aplikaciji autorizacijski kod (authorization code) putem redirect\_uri, dok se kod neuspješne autentifikacije korisniku prikazuje poruka o grešci te nudi ponovno pokušavanje.

---

#### DOBIVANJE ACCESS TOKENA I KORISNIČKIH INFORMACIJA

Kada se redirect\_uri automatski otvorи, web aplikacija šalje zahtjev Google OAuth2 servisu odnosno traži access token koristeći code. Google vraća access token i informacije o korisniku (npr. ime, email).

---

#### PROVJERA KORISNIKA U BAZI PODATAKA

Web aplikacija provjerava postoji li korisnik u bazi podataka. Ako korisnik ne postoji, aplikacija kreira novi korisnički zapis i dodjeljuje mu početnu ulogu "Gost", a ako korisnik već postoji, učitavaju se njegovi postojeći podaci i uloga.

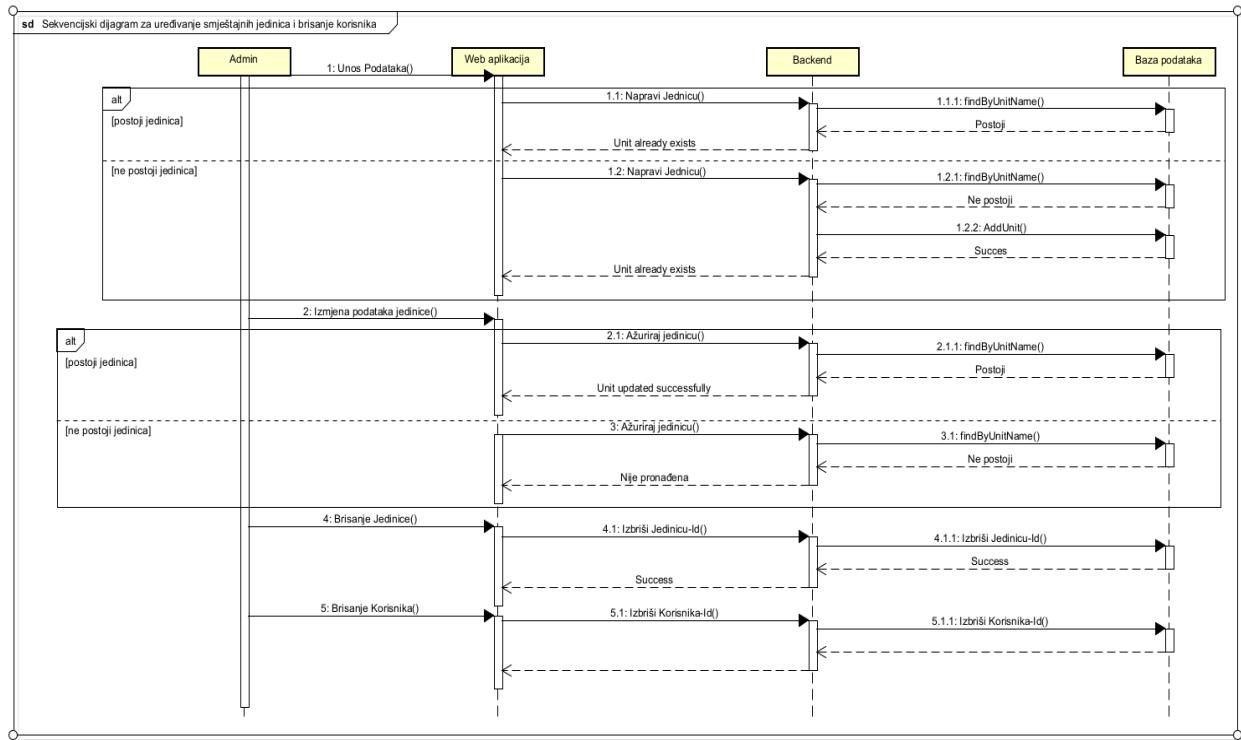
---

#### PRIKAZ KORISNIČKOG SUČELJA PREMA ULOZI

Dijagram dalje koristi alt blok da prikaže različite scenarije ovisno o ulozi korisnika. Za svakog korisnika, ovisno o tome je li gost, ugostitelj ili administrator postoji odgovarajuće sučelje.

---

#### SEKVENCIJSKI DIJAGRAM ZA UPRAVLJANJE SMJEŠTAJNIH JEDINICA I BRISANJE KORISNIKA (UC2, UC8)



Slika 3.5 Sekvenčni dijagram za upravljanje smještajnih jedinica i brisanje korisnika

Ovaj sekvenčni dijagram prikazuje procese dodavanja, ažuriranja i uklanjanja smještajnih jedinica te brisanja korisnika. U dijagramu sudjeluju četiri glavna sudionika (lifeline-a): Admin, Web aplikacija, Backend i Baza podataka.

#### ADMIN POKREĆE AKCIJU

Admin u web aplikaciji odabire jednu od dostupnih akcija: \* dodavanje smještajne jedinice \* ažuriranje postojeće smještajne jedinice \* uklanjanje smještajne jedinice \* brisanje korisnika Web aplikacija prikazuje odgovarajuću formu ili potvrdu akcije.

#### PROVJERA POSTOJANJA NAZIVA (ALT GRANA — SAMO ZA DODAVANJE I AŽURIRANJE SMJEŠTAJA)

Kada admin unese podatke za novu smještajnu jedinicu ili ažurira postojeću, web aplikacija šalje zahtjev Backend-u. Backend zatim provjerava u Bazi podataka postoji li smještajna jedinica s istim nazivom.

#### DODAVANJE I AŽURIRANJE SMJEŠTAJNE JEDINICE

U slučaju da naziv ne postoji ili se nije mijenjao, Backend upisuje podatke u Bazu: \* kod dodavanja: spremi novi zapis smještajne jedinice \* kod ažuriranja: modificira postojeće podatke Nakon uspješne operacije Baza podataka vraća potvrdu Backend-u, koji prosljeđuje Web aplikaciji poruku o uspjehu.

#### UKLANJANJE SMJEŠTAJNE JEDINICE

Web aplikacija šalje zahtjev Backend-u, koji zatim upućuje naredbu Bazi podataka za brisanje zapisa. Nakon završetka operacije, Backend šalje Web aplikaciji poruku o uspješnom brisanju.

## BRISANJE KORISNIKA

Proces je isti kao kod brisanja smještajne jedinice. Admin odabire korisnika kojeg želi izbrisati. Zeatim Web aplikacija šalje zahtjev Backend-u te Backend šalje naredbu Bazi podataka kako bi izbrisala korisnika. Nakon uspjeha, Web aplikacija prikazuje poruku adminu.

## PROVJERA UKLJUČENOSTI FUNKCIONALNOSTI U OBRASCE UPORABE

Funkcionalni zahtjev	Obrazac uporabe
F-001	UC1
F-002	UC1, UC8
F-003	UC8
F-004	UC2
F-005	UC2
F-006	UC2
F-007	UC6
F-008	UC3
F-009	UC3, UC7, UC10
F-010	UC4, UC9
F-011	UC5
F-012	UC11
F-013	UC6
F-014	UC6
F-015	UC9

## 4. ARHITEKTURA I DIZAJN SUSTAVA

### CILJ POGLAVLJA

Cilj ovog poglavlja je pružiti jasan, sažet i strukturiran pregled arhitekture sustava Room-Rently, uz razrađivanje ključnih komponenata i njihovih međusobnih odnosa. Dokumentacija omogućuje svim sudionicima projekta potpuno razumijevanje arhitekture, podjele odgovornosti i načina funkcioniranja sustava kao cjeline.

### OPIS ARHITEKTURE

#### STIL ARHITEKTURE

Room-Rently je razvijen koristeći **klijent-poslužitelj arhitekturu** s jasno odvojenim frontend i backend slojevima.

- **Frontend:** Web aplikacija (React.js) odgovorna za interakciju s korisnikom.
- **Backend:** HTTP endpoint servis (Spring Boot) koji upravlja poslovnom logikom, autentifikacijom i pristupom podacima.

#### - Razlozi odabira:

- Jednostavnost implementacije i održavanja
- Jasna podjela odgovornosti između korisničkog sučelja i poslovne logike
- Skalabilnost za buduće proširenje (mobilna aplikacija, integracija IoT uređaja)

---

#### PODSUSTAVI

1. **Autentifikacija i autorizacija** – upravljanje prijavom, autentifikacijom i pristupom sustavu prema ulogama (Gost, Voditelj hotela, Administrator).
2. **Upravljanje smještajem** – CRUD operacije nad smještajnim jedinicama, validacija i filtriranje.
3. **Upravljanje rezervacija** – kreiranje, izmjena i otkazivanje rezervacija, provjera dostupnosti i sprječavanje preklapanja termina.
4. **Notifikacije i potvrde** – automatsko generiranje PDF potvrda i slanje e-mail obavijesti korisnicima.
5. **Statistika i izvještaji** – pregled i izvoz podataka o zauzeću, gostima i popularnosti usluga.

---

#### PRESLIKAVANJE NA RADNU PLATFORMU

- Sustav se može implementirati na **lokalne servere ili cloud infrastrukturu** (npr. AWS, Azure) radi skalabilnosti i lakše dostupnosti.
- Cloud rješenje omogućuje jednostavno horizontalno skaliranje u slučaju povećanog broja korisnika i rezervacija.

---

#### SPREMIŠTA PODATAKA

- **Relacijska baza podataka (PostgreSQL)** za pohranu korisničkih podataka, smještajnih jedinica i rezervacija.
- Svaka glavna komponenta može imati vlastitu bazu radi izolacije podataka i sigurnosti.

---

#### MREŽNI PROTOKOLI

- **HTTPS** za siguran prijenos podataka između klijenta i poslužitelja.
- **HTTP endpoint komunikacija** između frontend i backend slojeva, gdje frontend šalje zahtjeve na definirane HTTP rute, a backend vraća odgovarajuće odgovore (JSON za podatke, PDF ili statične datoteke po potrebi).

---

#### GLOBALNI UPRAVLJAČKI TOK

1. Korisnik šalje zahtjev putem web aplikacije.
2. Backend provjerava autentifikaciju i autorizaciju.

- Sustav obrađuje zahtjev, komunicira s bazom podataka i generira odgovor.
- Rezultat se šalje korisniku u obliku prikaza na web sučelju, PDF potvrde ili e-mail obavijesti.

---

#### SKLOPOVSKOPROGRAMSKI ZAHTJEVI

- Server: minimalno 2 CPU jezgre, 4 GB RAM.
- Podržani OS: Windows Server.
- Web preglednici: Chrome, Firefox, Edge, Opera GX.

#### OBRAZLOŽENJE ODABIRA ARHITEKTURE

---

#### PRINCIPI OBLIKOVANJA

- Visoka kohezija** – funkcionalnosti su grupirane prema podsustavima.
- Niska povezanost** – frontend i backend su odvojeni, olakšava održavanje i nadogradnju.
- Fleksibilnost i skalabilnost** – modularna arhitektura omogućuje buduće proširenje (mobilna aplikacija, IoT integracija).
- Sigurnost** – autentifikacija putem OAuth2 i HTTPS protokol.

---

#### RAZMATRANE ALTERNATIVE

#### ORGANIZACIJA SUSTAVA NA VISOKOJ RAZINI

- Klijent-poslužitelj:** Frontend (React) komunicira s backend HTTP endpoint servisom.
- Baza podataka:** PostgreSQL relacijska baza za sve ključne podatke.
- Datotečni sustav:** Pohrana PDF potvrda rezervacija i primitaka.
- Grafičko sučelje:** Web aplikacija povezana s backendom putem HTTP endpoint komunikacije, responzivna i intuitivna.

#### ORGANIZACIJA APLIKACIJE

---

#### FRONTEND I BACKEND SLOJEVI

- Frontend:** prezentacijski sloj, prikazuje podatke korisniku, šalje zahtjeve backendu.
- Backend:** poslovna logika, kontrola pristupa, obrada podataka i komunikacija s bazom.

---

#### MVC ARHITEKTURA

Arhitektura sustava Room-Rently temelji se na MVC (Model–View–Controller) obrascu, koji omogućuje jasno odvajanje podataka, poslovne logike i korisničkog sučelja. Ovakav pristup olakšava razvoj, održavanje i nadogradnju aplikacije jer se promjene u jednom sloju ne odražavaju izravno na druge.

---

## MODEL

Model sloj predstavlja temelj aplikacije, zadužen za upravljanje podacima i implementaciju poslovnih pravila. U ovom sloju definiraju se klase koje odgovaraju entitetima u bazi podataka primjerice korisnik, rezervacija, smještajna jedinica, kategorija i dodatna usluga. Model komunicira s PostgreSQL bazom podataka te omogućuje sigurno dohvaćanje, pohranu i ažuriranje informacija.

---

## VIEW

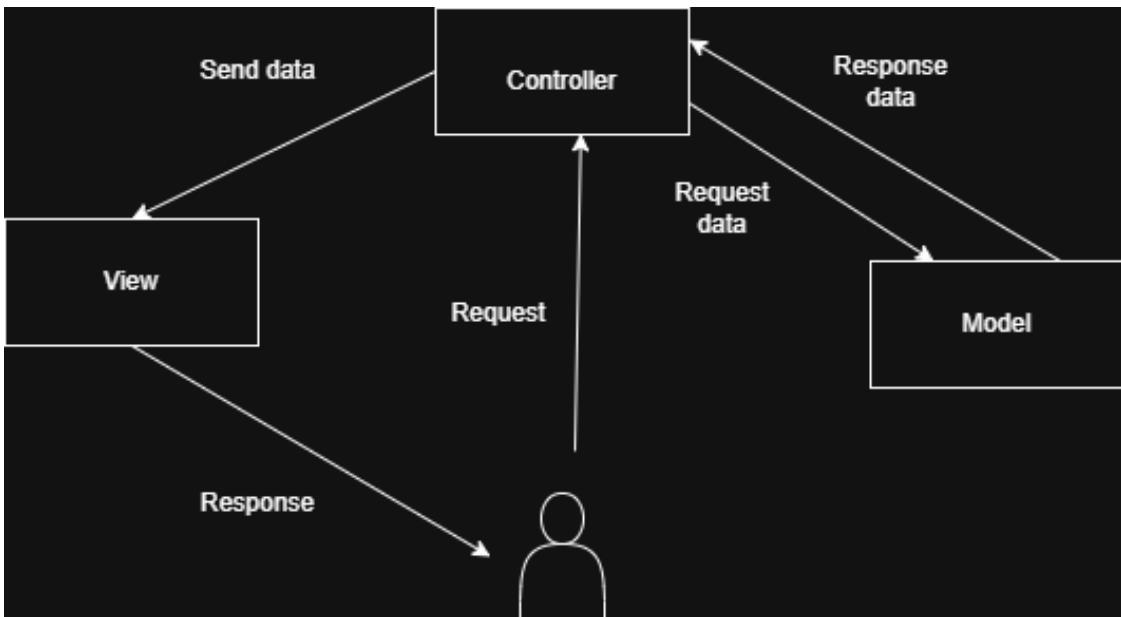
View sloj odgovoran je za vizualni prikaz podataka korisniku i interakciju s aplikacijom. U sustavu Room-Rently ovu ulogu preuzima frontend aplikacija izrađena pomoću React okvira. Frontend koristi React Router za navigaciju između različitih stranica i komponenti aplikacije, što omogućuje jednostraničnu strukturu s brzim i dinamičnim prijelazima bez ponovnog učitavanja stranice. Cilj View sloja je omogućiti korisnicima intuitivan pregled dostupnih soba, stvaranje i upravljanje novih jedinica te pregled profila bez potrebe za izravnim kontaktom s poslužiteljem ili bazom podataka.

---

## UPRAVLJAČ

Controller sloj djeluje kao posrednik između View i Model sloja. Njegova je zadaća primati korisničke zahtjeve s frontend aplikacije (npr. prijava, rezervacija, dodavanje nove jedinice,...), proslijediti ih servisima u model sloju te vraćati rezultate natrag View sloju. U sustavu Room-Rently, kontroleri su implementirani unutar Spring Boot okvira i definiraju REST API rute koje React frontend koristi za komunikaciju. Ovakva struktura omogućuje jasnu razdvojenost poslovne logike od prezentacijskog sloja, što znatno pojednostavljuje testiranje, održavanje i buduće nadogradnje aplikacije.

## MVC MODEL



Slika 4.1: Prikaz MVC modela sustava Room-Rently

## KLJUČNE KOMPONENTE

- Autentifikacija i autorizacija
- Upravljanje smještajem
- Upravljanje rezervacijama
- Notifikacije i potvrde
- Statistika i izvještaji
- Baza podataka – PostgreSQL
- Vanjski servisi – Google OAuth2, e-mail servisi, Google Maps API

## INTERAKCIJA IZMEĐU FRONTENDA I BACKENDA

- Frontend (React) komunicira s backendom (Spring Boot) putem HTTP zahtjeva poslanih pomoću Axios biblioteke. Axios omogućuje slanje GET, POST, PUT i DELETE zahtjeva prema REST API endpointima na backendu. Svi zahtjevi sadrže odgovarajuće zaglavje s autentifikacijskim tokenom (Authorization: Bearer ), čime se osigurava sigurna komunikacija i provjera korisničkog identiteta.
- Backend obrađuje zahtjeve unutar Spring Boot kontrolera (npr. PersonController.java), koji su mapirani na odgovarajuće rute pomoću anotacija poput @PostMapping, @GetMapping i

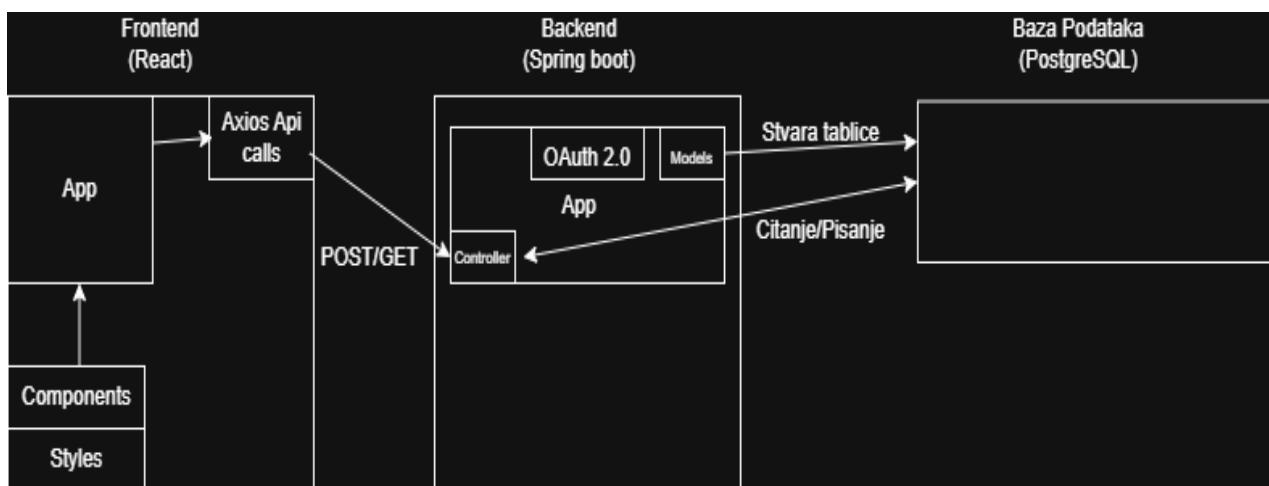
@RequestMapping. Backend zatim poziva servisni sloj i komunicira s bazom podataka kako bi vratio ili pohranio tražene informacije.

### PRIJAVA I AUTENTIFIKACIJA (OAUTH 2.0)

- Autentifikacija korisnika odvija se pomoću Google OAuth 2.0. Korisnik se prijavljuje putem Google računa, a Google API vraća ID token (JWT) koji se proslijeđuje backendu. Backend validira taj token pomoću sigurnosnog mehanizma OAuth 2.0 i: provjerava autentičnost korisnika, po potrebi pohranjuje korisnika u bazu podataka, i vraća status zahtjeva frontend aplikaciji.
- Na frontend strani, token se lokalno pohranjuje (localStorage) kako bi korisnik ostao prijavljen i nakon ponovnog učitavanja stranice.

### ZAŠTITA RUTA

- Na temelju prisutnosti i valjanosti tokena, frontend kontrolira pristup određenim komponentama aplikacije. Korisnici koji nisu prijavljeni ne mogu pristupiti zaštićenim rutama poput korisničkog profila ili stranice za rezervaciju smještaja. Ovo se ostvaruje pomoću React Router-a, koji preusmjerava korisnika na login stranicu ako token nije prisutan ili je nevažeći.



Slika 4.2 Dijagram organizacije aplikacije

### BAZA PODATAKA

Baza podataka predstavlja temelj našeg sustava jer nam omogućuje strukturiranu pohranu, dodavanje, promjenu i pretraživanje podataka te kontrolirani pristup. Temeljni element naše baze je entitet koji posjeduje skup svojstava ili atribute koji ga karakteriziraju. U našoj bazi definirali smo 4 entiteta za potrebe uspješnog vođenja evidencije poslovanja i upravljanja podacima. Ti atributi su redom: + Person + Unit + UnitImg + UnitReservation

### OPIS TABLICA

#### PERSON

Entitet Person sadrži sve važne informacije o korisniku aplikacije. Sadrži sljedeće atribute: id, email, is\_admin, is\_user, is\_owner, name.

Povezan je One-to-many vezom s entitetom UnitReservation, gdje se veza ostvaruje preko atributa id (Person može imati više rezervacija).

Funkcionalno, entitet Person predstavlja sve korisnike sustava i omogućuje razlikovanje uloga pomoću atributa is\_admin, is\_user i is\_owner. Rezervacije koje korisnik stvara ili upravlja pohranjuju se u tablicu UnitReservation.

Atribut	Tip podataka	Opis atributa
<b>id</b>	BIGINT	jedinstveni identifikator osobe (primarni ključ)
<b>email</b>	TEXT	jedinstvena email adresa osobe
<b>is_admin</b>	BOOLEAN	oznaka je li osoba administrator
<b>is_user</b>	BOOLEAN	oznaka je li osoba gost
<b>is_owner</b>	BOOLEAN	oznaka je li osoba vlasnik/voditelj
<b>name</b>	TEXT	ime osobe

---

## UNIT

Entitet Unit sadrži sve važne informacije o smještajnim jedinicama unutar aplikacije. Sadrži sljedeće atribute: id\_unit, price, num\_rooms, cap\_adults, cap\_children, num\_beds, rating, has\_parking, has\_wifi, has\_breakfast, has\_towels, has\_shampoo, has\_hair\_dryer, has\_heater, has\_air\_conditioning, apartment, unit\_name, location, main\_desc\_name, main\_desc\_content, sec\_desc\_name, sec\_desc\_content.

Entitet ima One-to-many veze s entitetima UnitImg i UnitReservation, gdje se veze ostvaruju preko atributa id\_unit.

Funkcionalno, entitet Unit omogućuje pohranu svih relevantnih podataka o smještajnim jedinicama, uključujući kapacitete, pogodnosti, detaljne opise i slike, te povezuje jedinicu s rezervacijama koje korisnici kreiraju u sustavu.

Atribut	Tip podataka	Opis atributa
<b>id_unit</b>	BIGINT	jedinstveni identifikator smještajne jedinice (primarni ključ)

<b>price</b>	INTEGER	cijena smještajne jedinice
<b>num_rooms</b>	INTEGER	broj soba u jedinici
<b>cap_adults</b>	INTEGER	smještajni kapacitet za odrasle osobe
<b>cap_children</b>	INTEGER	smještajni kapacitet za odrasle djecu
<b>num_beds</b>	INTEGER	broj kreveta u jedinici
<b>rating</b>	INTEGER	ocjena jedinice
<b>has_parking</b>	BOOLEAN	oznaka postoji li parking
<b>has_wifi</b>	BOOLEAN	oznaka postoji li Wi-Fi
<b>has_breakfast</b>	BOOLEAN	oznaka uključuje li doručak
<b>has_towels</b>	BOOLEAN	oznaka uključuju li se ručnici
<b>has_shampoo</b>	BOOLEAN	oznaka uključuje li šampon
<b>has_hair_dryer</b>	BOOLEAN	oznaka uključuje li sušilo za kosu
<b>has_heater</b>	BOOLEAN	oznaka uključuje li grijač
<b>has_air_conditioning</b>	BOOLEAN	oznaka uključuje li klima uređaj
<b>apartment</b>	BOOLEAN	oznaka je li jedinica apartman
<b>unit_name</b>	TEXT	naziv smještajne jedinice
<b>location</b>	TEXT	lokacija smještajne jedinice
<b>main_desc_name</b>	TEXT	naziv glavnog opisa jedinice
<b>main_desc_content</b>	TEXT	sadržaj glavnog opisa jedinice

<b>sec_desc_name</b>	TEXT	naziv sekundarnog opisa jedinice
<b>sec_desc_content</b>	TEXT	sadržaj sekundarnog opisa jedinice

---

## UNITRESERVATION

Entitet UnitReservation sadrži sve važne informacije o rezervacijama smještajnih jedinica unutar aplikacije. Sadrži sljedeće atribute: id\_unit\_reservation, start\_date, end\_date, status, person, unit.

Entitet ima Many-to-one veze s entitetima Person i Unit, gdje se veze ostvaruju preko atributa id (Person) i id\_unit (Unit).

Funkcionalno, entitet UnitReservation omogućuje pohranu svih rezervacija koje korisnici kreiraju u sustavu. Povezuje korisnika i smještajnu jedinicu te čuva podatke o datumu početka, završetka i statusu rezervacije, što omogućuje daljnju obradu poput generiranja potvrda i provjere zauzetosti jedinica.

Atribut	Tip podataka	Opis atributa
id_unit_reservation	BIGINT	jedinstveni identifikator rezervacije (primarni ključ)
start_date	DATE	datum početka rezervacije
end_date	DATE	datum završetka rezervacije
status	TEXT	status rezervacije
id	BIGINT	identifikator korisnika koji je kreirao rezervaciju (strani ključ prema Person)
id_unit	BIGINT	identifikator smještajne jedinice koja je rezervirana (strani ključ prema Unit)

---

## UNITIMG

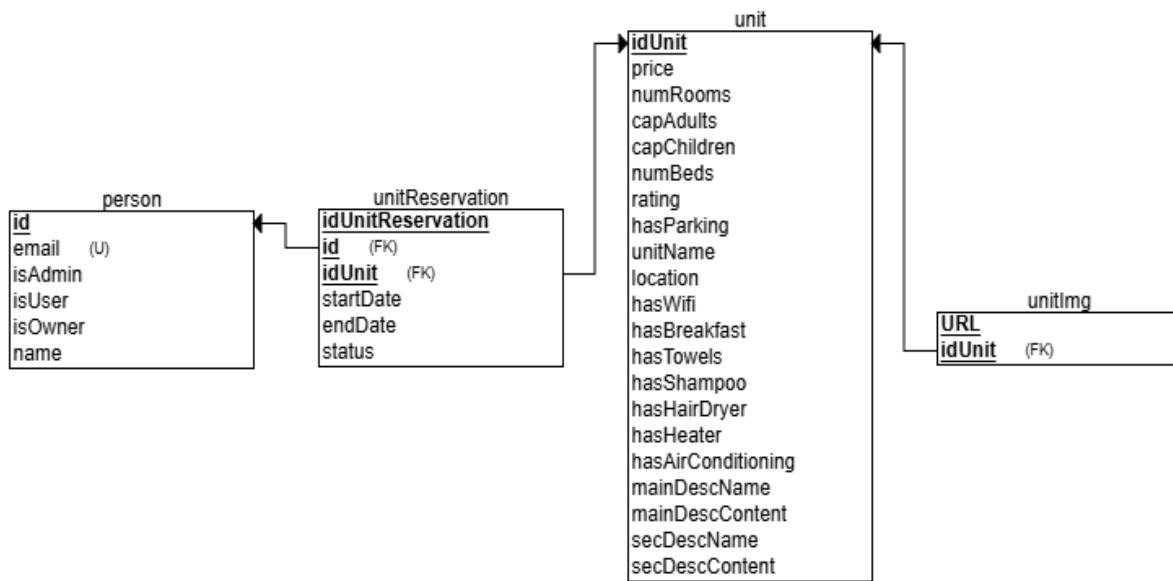
Entitet UnitImg sadrži sve važne informacije o slikama smještajnih jedinica unutar aplikacije. Sadrži sljedeće atribute: url, unit.

Entitet ima Many-to-one vezu s entitetom Unit, gdje se veza ostvaruje preko atributa id\_unit.

Funkcionalno, entitet UnitImg omogućuje pohranu URL-ova slika koje pripadaju određenoj smještajnoj jedinici.

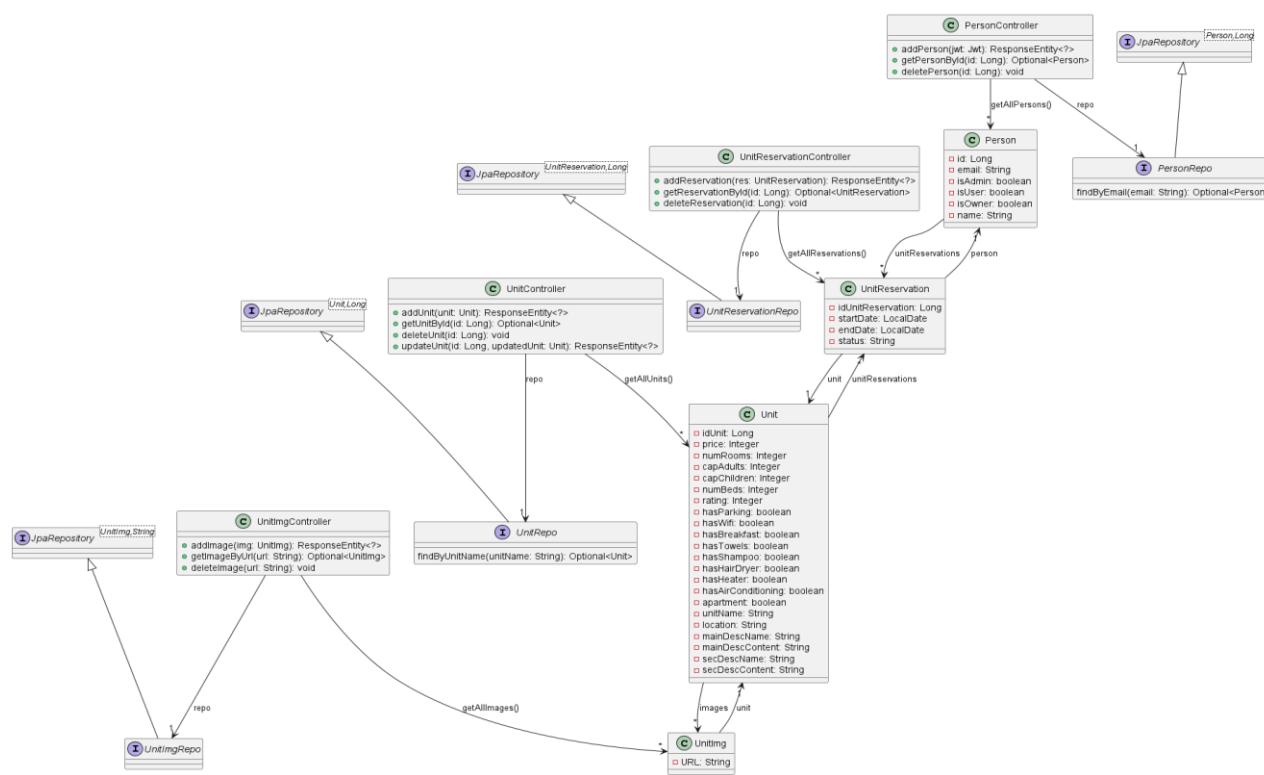
Atribut	Tip podataka	Opis atributa
url	TEXT	jedinstveni URL slike (primarni ključ)
id_unit	BIGINT	identifikator smještajne jedinice kojoj slika pripada (strani ključ prema Unit)

ER DIJAGRAM BAZE PODATAKA



Slika 4.3 ER Dijagram

## DIJAGRAM RAZREDA



Slika 4.4 Dijagram razreda

Dijagram prikazuje skup klasa koje čine osnovu sustava za upravljanje smještajem, rezervacijama i pripadajućim podacima o korisnicima i jedinicama smještaja.

Model korisnika predstavljen je klasom Person, koja sadrži osnovne informacije kao što su id, name, email, te boolean polja isAdmin, isUser i isOwner koja definiraju ulogu korisnika unutar sustava. Za pristup podacima o korisnicima koristi se PersonRepo, dok PersonController omogućuje dodavanje, dohvatanje i brisanje korisnika.

Smještajne jedinice modelirane su klasom Unit, koja sadrži detalje poput broja soba (numRooms), kapaciteta odraslih i djece (capAdults, capChildren), cijene (price) te obilježja kao što su dostupnost Wi-Fi-ja, parkinga, doručka i drugih pogodnosti. Za rad s podacima o smještajnim jedinicama koristi se UnitRepo, dok UnitController omogućuje dodavanje, dohvatanje i brisanje smještajne jedinice.

Rezervacije smještajnih jedinica modelirane su klasom UnitReservation, koja sadrži id, startDate, endDate i status rezervacije. Kontroler UnitReservationController omogućuje dodavanje, dohvatanje i brisanje rezervacija, dok UnitReservationRepo služi za rad s bazom podataka.

Slike smještajnih jedinica predstavljene su klasom UnitImg sa Stringom URL koji predstavlja relativnu putanju slike. UnitImgController omogućuje dodavanje, dohvatanje i brisanje slika, dok UnitImgRepo komunicira s bazom podataka.

Kontroleri sustava (UnitController, UnitReservationController, UnitImgController, PersonController) koriste pripadajuće repozitorije za dohvatanje i manipulaciju podacima. Svaki kontroler nudi metode za osnovne CRUD operacije, a repozitoriji nasljeđuju JpaRepository, što omogućuje jednostavan rad s bazom podataka.

Ovim modelom osigurana je jasna povezanost između korisnika, smještajnih jedinica i rezervacija. Svaka klasa je modularno odvojena, a korištenje repozitorija omogućuje jednostavno proširenje sustava, primjerice dodavanje novih vrsta rezervacija, dodatnih informacija o korisnicima ili proširenih značajki smještajnih jedinica.

## A. POPIS LITERATURE

### BAZA PODATAKA

- [Connect a PostgreSQL database to a Spring Boot Application Tutorial AnbuZ HobbieZ ## Backend](#)

### FRONTEND

- [React Booking | Reservation App UI Design for Beginners](#)
- [React Google Login](#)
- [Google Maps](#)
- [Launch Site Inspiration](#)

## B. PRIKAZ AKTIVNOSTI GRUPE

### —ZAPISNIK 10.10.2025.—

- Jakov pokazuje ideje frontend dizajna i funkcionalnosti koje je pronašao na Youtubeu, itd.
- šifre u bazi podataka treba hashirati nekim pouzdanim algoritmom
- Noa će napraviti glavnu dokumentaciju za funkcionalnosti
- koristit ćemo Trello za lakšu organizaciju, odite na i registrirajte se na Trellu imamo boardove za frontend, backend, baze podataka i organizaciju, kako koristiti (napisat ću neki tutorial u chat)
- trebamo Github posložiti za vođenje bilježaka o razvoju aplikacije
- svaki kod mora biti popraćen komentarima u kodu, draftom tijeka misli i problemima s kojima ste se susreli, nakon iz tog drafta napišete dokumentaciju svojeg rada koji će biti jasan nekome tko nije bio uz vas dok ste radili “posao” da može replicirati vaše korake i dobiti isti rezultat
- meet za funkcionalnosti projekta (SVI MORAJU BITI PRISUTNI)

ZA SUTRA @everyone - Noa će započeti dijagram baze podataka koji ćemo sutra pregledati da vidimo što još dodati i mijenjati - Nino i Mateo će pogledati OAuth2, Google Maps linking i slične funkcije da skupe informacije koje će koristiti kao smjernice kod implementacije - Karlo i Jakov će započeti dizajnirati stranice koje ćemo također sutra pregledati i prokomentirati - svi ćemo napisati ideje i todo kartice u TRELLO board što nam padne na pamet da treba napraviti za ubuduće (primjer: frontend zna da mora napraviti Početnu stranicu s get started buttonom i to će napisati u ideje pa prebaciti u todo kad dovrše karticu) - Josip će posložiti Github repo (s Wikiem, branchevima, README.md, itd.), TRELLO porediti da svi imaju potrebne liste, napisati howto za korištenje TRELLO-a i bilježenje napretka u razvoju koji ćemo pratiti

### —ZAPISNIK 11.10.2025.—

- prošli smo HOWTO kanal
- vizualno objašnjavali kako Github Desktop koristiti
- updateali Matea i Nina koji prošli sastanak nisu bili
- gledali napredak u frontendu koji je Jakov napravio
- Jakov mora zapisati svoju implementaciju u Trello karticu i draft
- gledali smo dijagram baze podataka koju je Noa napravio
- treba predati dokument s funkcionalnostima do kraja sljedećeg tjedna (imamo sastanak u srijedu za prokomentirat napisane funkcionalnosti)

DO SRIJEDE 15.10. @everyone - svaka ekipa će napisati u Trello Megkarticu funkcionalnosti i u opis kartice raspisati funkcionalnosti koje istraže online s prioritetima, imajte na umu da moramo to formatirati za predaju do kraja tjedna pa kvalitetno to istražite i zapišite - Jakov će zapisati i urediti svoje kartice na Trellu o onome što je implementirao kroz Youtube tutorial, također će linkati Youtube tutorial u komentare kartice - Jakov će raspisati draft što je radio konkretno kroz Youtube kanal kako bi mogao s Karlom napisati profesionalan final dokument sa snippetima i objašnjenjima koda - Jakov i Karlo će napisati final dokument koji je već na Githubu s markdown uređenjem teksta kako bi netko mogao replicirati što je Jakov implementirao, koristit će howto kako bi formatirali taj dokument u obliku problem/rješenje - Josip i tko bude voljan će ići na labos konzultacije u srijedu 15.10. kako bi saznao informacije o formatiranju dokumenta za predaju

---

#### —ZAPISNIK KRHEN 15.10.2025.—

##### **Pitanje: Chat na stranici?**

Bilo bi lijepo imati chat. Komentari pri rezervaciji barem. Zahtjevi. Međusobno nema potrebe chatat. Eventualno chat bot, ali ne treba.

##### **Pitanje: Funkcijske jesu dobro zapisane?**

Sruk wiki kao primjer.

##### **Pitanje: Prezentacije samo dokument ili uživo prezentirati?**

Rok srijeda 22.10. sljedeći termin kreće od 8 ako namjesti Krhen, prezentacija plus uživo pričati. Ono iz zahtjeva treba prezentirati na lijepi način da je interesantno, ne sve raspisati. Preza i govor.

##### **Pitanje: Github username Krhen, demos?**

mkKrhen, demos ivogabud

##### **Pitanje: Statistika?**

Prosjek rezervacija, broj dana, broj ljudi, od kud ljudi dolaze, traže li posebne usluge, ako 80% i više ljudi traže parking da staviš parking kao standardnu opciju, koliko sam imao praznog hoda usred sezone, kad sam imao najviše slobodnih soba. Staviti se u ulogu ugostitelja i razmišljati.

##### **Pitanje: Razlike u administratoru, vlasniku hotela i korisniku?** Ovisi kako zamislimo da ćemo to napraviti. Vlasnik može biti i admin. To dajemo hotelu JEDNOM.

**BITNO RADIMO APIKACIJU NE BOOKING COM NEGO ZA JEDAN HOTEL DI IMA VLASNIKA, ADMINA ITD.**

**PRIMJER RJEŠENJA:** <https://www.bluesunhotels.com/booking/hotel-elaphusa?adults=2&loyalty=true>

Korisnik ne bira broj sobe, samo pogled tipa prema moru itd.

---

#### —ZAPISNIK 24.10.2025.—

Prisutni: Noa, Mateo, Josip, Nino

- komentiramo kako treba detaljno pisati komentare na Trellu s checkistama i opisima kako bi se mogla pratiti aktivnost za završnu tablicu koju će možda trebati pisati (u srijedu 29.10. ćemo saznati od Krhena)
- treba pisati u komentarima na karticama datume kad je tko što završio kako se ne bi desilo da Trello iztrijpa i obriše aktivnosti + aktivnosti se jako nakrcaju jer svaki pomak kartice iz liste u listu, tipa Doing u Done se broji kao aktivnost
- gledamo bazu podataka koju je Noa napravio i komentiramo
- stavljamo Noin napredak na Github
- Noa i Mateo syncaju što su napravili kako bi Noa mogao na Github objaviti svoje

Za napraviti sljedeće - @Baza podataka do **nedjelje 26.10. (uključujući)** istražiti kako staviti na Github bazu podataka i napisati kako je mi drugi možemo koristiti - @Backend kad baza bude stavljena na Github treba nastaviti s pretvorbom frontenda u backend što znači da se većina toga povlači sa servera kao na labosu iz WEBa - @Baza podataka i @Backend trebaju komunicirati kako bi sve bilo syncano jedno s drugim i **oba tima rade na branchu bazepod!!!!!!** - @Baza podataka i @Backend trebaju do **srijede 29.10.** imati gotov login s Google Sign Inom, znači treba sve bitne informacije povući s OAuth2 i spremiti u bazu i te informacije se moraju moći funkcijom izvlačiti iz baze (**FUNKCIJE MOGU BITI NAPRAVLJENE IAKO NEMA FRONTEND DIJELA KOJIM IH SE POKREĆE**) - @Baza podataka i @Backend trebaju do **nedjelje 2.11.** postaviti sve funkcije za pretraživanje smještajnih jedinica (filtere i samo pretraživanje mora funkcionirati)

**JAKO BITNO** @everyone - **KOMUNICIRAJTE S DRUGIMA KAD RADITE, POGOTOVO S VAŠIM TIMOM.**

- **DOKUMENTIRAJTE SVE KONTINUIRANO I ISPRAVNO POGOTOVO U TRELLU I WIKI “TIJEK IMPLEMENTIRANJA X”, SVA AKTIVNOST U TRELLU ĆE BITI TEMELJ DONOŠENJA BODOVA POJEDINCU NA KRAJU PROJEKTA.**
- **BILO KAKAV PROBLEM PITAJTE KOLEGU IZ TIMA, AKO OBOJE NE ZNATE ILI SAMI RADITE PITAJTE @Voda/fullstack. POTRUDITE SE PRONAĆI ODGOVOR SAMI UZ CHATGPT, GOOGLE, ITD. PRIJE NEGO DRUGU OSOBU PITATE, ALI ČIM VIDITE DA PREVIŠE VREMENA TROŠITE NA TRAŽENJE BEZ NAPRETKA PITAJTE!**

#### —ZAPISNIK KRHEN 30.10.2025.—

**Pitanje: Slike u bazi podataka u byteovima ili samo putanja?** Odgovor: Putanja/URL uvijek u bazi podataka. Napraviti filesystem u kojem ćemo pohranjivati sve slike, zvukove, itd.

**Pitanje: Aktivnost Wiki page treba li aktivno zapisivati ili je okej na kraju prekopirati s Trella?** Odgovor: Ne vidi se kad se commita Wiki, okej je Trello i zapisujte kada tko što doda na Wiki jer se ne može pratiti. Napisati snapshot aktivnosti do međuispita na Wiki. Pišite i circa broj sati koliko ste što radili.

**Pitanje: Dijagrami za zahtjeve i arhitekturu sustava, možemo li ih na kraju zaljepiti kad budemo sigurni kako će izgledati sve ili treba aktivno ih nadograditi?** Odgovor: Trebalo bi raditi postepeno pa nadodavati nove elemente u dijagrame kako ih razrađujemo.

**Pitanje: Dokumentacija i prototip predaja prije međuispita?** Odgovor: Sve u Wiki samo mora biti gotovo do roka, ne moramo ekstra raditi pdf ili što već ako se ne promjeni do tada zahtjev. Moramo imati funkcionalnost kao upisivanje smještajne jedinice, da aplikacija otvorи prozor tipa i upisuješ informacije o jedinici u prozor i da se to spremi u bazu podataka i mora to iz baze aplikacija povući i prikazati tipa kao search smještajnih jedinica. Napraviti superusera/admina koji to može napraviti, napraviti fake google mail koji će biti default admin pa se može nadodavati druge.

**Pitanje: Bodovanje commitovi?** Odgovor: Vođa dodjeljuje bodove nakon prve predaje prototipa (prije međuispita) pa će neradnici biti kažnjeni prikladnim brojem bodova.

#### —ZAPISNIK 9.11.2025.—

- diskutiranje svega što smo dokumentirali
- Noa će zapisat što konkretno radi kod ne samo objasniti kod liniju po liniju (relacije stvaraju tablice u bazu podataka kad se backend spoji na bazu)
- treba zapisati u Trello sve dokumentirano i zapisati sate i datume kad je napravljeno
- gledali smo dokumentaciju backenda i dokumentaciju kako postaviti na poslužitelja
- Mateo će zapisati kako lokalno pokrenuti backend da funkcioniра s environment varijablama
- Josip će zapisati dokumentaciju postavljanja Supabase poslužitelja
- gledali smo dokumentaciju frontenda
- treba nadodati sve linkove od Youtube tutoriala u Tijek Implementiranja i u Popis Literature
- treba napisati da se koristio ChatGPT
- provjeriti sve dijagrame koje trebamo imati i koje već imamo
- raspodijelili smo dijagrame po osobama, dijagrame treba danas napraviti
- provjerili smo wiki naslove ako imamo sve

@everyone - trebate danas napraviti dijagrame koji su vam dodjeljeni na Trello stranici - provjerite sve dokumentacije - zapišite vrijeme uloženo i datume kad je napravljeno

#### TABLICA AKTIVNOSTI

##### Napomena:

Doprinosi u aktivnostima navedeni su u satima po članovima grupe po aktivnosti.

Potrebno je navesti koliko je sati koja osoba uložila u pojedinu komponentu.

Aktivnost	Josip Mrakovčić	Jakov Zekić	Karlo Živković	Noa Rešetar	Mateo Cerčić	Nino Strčić
Upravljanje	50	0	0	0	0	0.001

projektom						
Opis projektnog zadatka	2	0	0	0	0	0
Funkcionalni zahtjevi	2	3	3	3	3	0
Opis pojedinih obrazaca	4	0	0	2	0	0
Dijagram obrazaca	3	0	0	3	0	0
Sekvencijski dijagrami	0	0	3	0	3	0
Opis ostalih zahtjeva	1	2	2	0	0	0
Arhitektura i dizajn sustava	1	2	0	2	0	0
Baza podataka	0	0	0	5	0	0
Dijagram razreda	0	0	2	2	0	0
Dijagram stanja	0	0	0	0	0	0
Dijagram aktivnosti	0	0	0	0	0	0
Dijagram komponenti	0	2	0	0	0	0
Korištene tehnologije i alati	0	0	0	0	0	0
Ispitivanje programskog rješenja	3	2	0	2	0	0
Dijagram razmještaja	0	1	1	0	0	0
Upute za puštanje u pogon	4	23	73	18	8	0
Dnevnik sastajanja	7	0	0	0	0	0
Zaključak i budući rad	0	0	0	0	0	0
Popis literature	0	0	0	0	0	0
Izrada početne stranice	0	20	0	0	0	0
Izrada baze podataka	0	0	0	28	0	0
Spajanje s bazom podataka	2	0	0	5	5	0
Izrada prezentacije	12	0	0	0	0	0
Frontend	8	97	11	0	0	0
Backend	8	0	0	0	24	0
Hostanje na Cloud	4	0	0	0	6	0

## C. DNEVNIK PROMJENA DOKUMENTACIJE

Za praćenje promjena koristimo Trello workspace (<https://trello.com/w/roomrently>).

TABLICA

Rev.	Opis promjene/dodataka	Autori	Datum
0.1	Napravljen početni predložak dokumentacije	Josip Mrakovčić	10.10.2025
0.2	Kreirana Home stranica i prvi sadržaj	Josip Mrakovčić	10.10.– 20.10.2025
0.3	Dodana poglavlja: 1. Opis projektnog zadatka, 2. Analiza zahtjeva	Josip Mrakovčić	20.10.– 25.10.2025
0.4	Kreirana početna verzija poglavlja 3. Specifikacija zahtjeva sustava	Josip Mrakovčić	24.10.2025
0.5	Višestruka ažuriranja Home, 1. Opis projektnog zadatka, 2. Analiza zahtjeva	Josip Mrakovčić	24.10.– 31.10.2025
0.6	Kreirana početna verzija poglavlja 4. Arhitektura sustava	Josip Mrakovčić	30.10.2025
0.7	Ažuriranja pogl. 3 i 4 – više iteracija	Josip / Karlo / Noa / Jakov	31.10.– 07.11.2025
0.8	Kreirani dokumenti implementacije: Backend, Frontend, Baza podataka, FullStack	Josip / Mateo / Noa / Jakov	24.10.– 07.11.2025
0.9	Intenzivna ažuriranja implementacijskih poglavlja (Backend, Frontend, Baza, FullStack) – veliki broj nadogradnji	Mateo / Jakov / Noa / Josip	07.11.– 14.11.2025
0.95	Dodana „Dokumentacija o objavljivanju aplikacije na javnom poslužitelju“ i višestruka ažuriranja	Mateo / Josip	07.11.– 11.11.2025
1.0	Finalna konsolidacija svih wiki stranica, uređivanje strukture i priprema dokumentacije za predaju	Svi članovi tima	14.11.2025

Evidencija promjena sadrži popis promjena izvršenih tijekom cijelog životnog trajanja dokumentacije.

Svrha je praćenje napretka svake promjene na temelju njezina preispitivanja, odobrenja (ili odbijanja), provedbe i zaključenja.

Kvalitetan dnevnik promjena sadrži i datum promjene te procjenu njezina utjecaja na projekt u smislu rizika, vremena i troškova.

Promjene su vidljive svim dionicima projekta. U povijest se uključuju i odbačene promjene.

### PRAĆENJE PROMJENA

Sve navedene promjene mogu se detaljno pratiti kroz **povijest revizija (Revisions)** svakog pojedinog Wiki dokumenta.

GitHub automatski evidentira svaku izmjenu zajedno s autorom, datumom i pripadajućim commit ID-em, čime se omogućuje jasan uvid u tijek razvoja dokumentacije.

Pregledom revizija na svakom Wiki pageu dionicima je olakšano praćenje svih izmjena, kao i razumijevanje njihovog utjecaja na projekt — uključujući rizike, vrijeme i troškove.

Na taj način osigurava se potpuna transparentnost i mogućnost praćenja kompletne povijesti izmjena sve do revizije **1.0**.

## DOKUMENTACIJA O OBJAVLJIVANJU APLIKACIJE NA JAVNOM POSLUŽITELJU

Video koji je korišten za postavljanje (backend/frontend/baza online):

<https://www.youtube.com/watch?v=jBDTsf8jsEs&list=PLacy7j-N4DaT4yWf0v1FKDPK34lue2y3e&index=8>

### SAŽETAK

Ova dokumentacija opisuje **sve promjene i dodatke napravljene kako bi se cijeli Room-Rently projekt uspješno postavio na internet (backend + frontend + baza + Docker)**. Uključuje postupak inicijalizacije novog Spring Boot projekta radi Maven Wrappera, ručno dodavanje getter/setter metoda u **Unit** i **UnitImg** modele, kreiranje Dockerfile-a (uz korištenje ChatGPT-a), te dodavanje lokalnih **.env** varijabli za jednostavnije konfiguiriranje i deployment.

### PROMJENE I PROBLEMI TIJEKOM RAZVOJA

#### 1. Inicijalizacija novog Spring Boot projekta

- Razlog: trebalo je generirati **Maven Wrapper** (**mvnw**, **mvnw.cmd**) kako bi se projekt mogao graditi i pokretati konzistentno na strojevima koji možda nemaju globalni **mvn**.
- Ovo je učinjeno lokalno tijekom razvoja — nakon toga su wrapper datoteke dodane u repozitorij.

#### 2. Getteri i setteri u **Unit** i **UnitImg**

- Problem: u originalnim model klasama **Unit** i **UnitImg** nisu bili dostupni getteri i setteri.
- Rješenje: ručno su ubačeni odgovarajući **getX()** i **setX()** metode kako bi JPA i serijalizacija radili ispravno.

#### 3. Dockerfile

- Dockerfile je izrađen korištenjem **ChatGPT**.

### PRIPREMA BACKENDA

```
## Dockerfile ##

# Build stage
FROM maven:3.9.8-eclipse-temurin-21 AS build

WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline

COPY src ./src
```

```

RUN mvn clean package -DskipTests

# Run stage - using one of the alternatives
FROM eclipse-temurin:21-jre-alpine

WORKDIR /app
COPY --from=build /app/target/backend-0.0.1-SNAPSHOT.jar .

EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app/backend-0.0.1-SNAPSHOT.jar"]

```

Ovaj Dockerfile koristi multi-stage build pristup. To znači da se aplikacija najprije gradi u jednom privremenom container-u (sa Mavenom), a zatim se rezultat (JAR datoteka) prenosi u drugi, manji container za pokretanje. Ukratko kompajlira kod i gradi JAR.

**Napomena:** Dockerfile je generiran korištenjem ChatGPT-a. **Datum pristupa:** 2025-11-8 **Svrha:** Generiranje Dockerfile specifično prilagođenog mojoj Spring Boot aplikaciji.

## Konfiguracija `.env` datoteke (lokalno)

Za olakšanu konfiguraciju baze i frontenda kreirana je `.env` datoteka s lokalnim varijablama:

```

DATASOURCE_URL=jdbc:postgresql://localhost:5432/BazaPROGI
DATASOURCE_USER=postgres
DATASOURCE_PASSWORD=bazepodataka
FRONTEND_URL=http://localhost:3000

```

Ove varijable se koriste prilikom pokretanja backenda i prilagodbe postavki za povezivanje na bazu i frontend.

## Funkcija `frontendUrl` ##

Varijabla `frontendUrl` služi za dinamičko dohvaćanje URL-a frontenda iz ili `application.properties`, tako da se CORS pravila automatski prilagode okruženju. Time backend dopušta zahtjeve samo s definirane frontend adrese.

Koristi se u CORS konfiguraciji u funkciji `corsConfigurationSource` u klasi `SecurityConfig`:

```
config.setAllowedOrigins(List.of(frontendUrl));
```

## `application.properties` ##

```

spring.datasource.url=${DATASOURCE_URL}
spring.datasource.username=${DATASOURCE_USER}
spring.datasource.password=${DATASOURCE_PASSWORD}

spring.profiles.include=docker
frontend.url=${FRONTEND_URL}

```

**spring.profiles.include=docker** Aktivira dodatni Spring profil naziva docker, što omogućuje da se određene postavke učitaju samo kada se koristi Docker okruženje. **frontend.url=\${FRONTEND\_URL}** Postavlja URL frontend aplikacije. Vrijednost se ne upisuje ručno, već se čita iz environment varijable FRONTEND\_URL **spring.datasource.url=\${DATASOURCE\_URL}** Definira URL baze podataka. Umjesto statičke vrijednosti koristi se environment varijabla DATASOURCE\_URL **spring.datasource.username=\${DATASOURCE\_USER}** Korisničko ime za pristup bazi podataka, također preuzeto iz environment varijable. **spring.datasource.password=\${DATASOURCE\_PASSWORD}** Lozinka za pristup bazi podataka, preuzeta iz environment varijable radi sigurnosti

## PRIPREMA FRONTENDA

Kako bi frontend aplikacija znala s kojim backendom treba komunicirati, potrebno je dodati datoteku .env u frontend projekt s ovim sadržajem:

**REACT\_APP\_API\_URL=https://room-rently-backend.onrender.com**

Za pozivanje backend API-ja bilo gdje u frontend aplikaciji moguće je koristiti:

**`\${process.env.REACT\_APP\_API\_URL}`** Ova metoda dohvaća URL backenda iz .env datoteke, gdje je definiran parametar.

Nakon toga, backend se može pozivati korištenjem odgovarajućih HTTP metoda (GET, POST, PUT, DELETE, itd.). Primjer: `fetch(${process.env.REACT_APP_API_URL}/unit/all);`

## DEPLOYMENT BACKENDA

### KORAK 1: POSTAVLJANJE VARIJABLJI OKRUŽENJA

Prije pakiranja aplikacije, potrebno je konfigurirati njen connection string i druge postavke. To radimo postavljanjem varijabli okruženja direktno u terminalu.

```
$env:DATASOURCE_URL="jdbc:postgresql://localhost:5432/BazaPROGI"  
$env:DATASOURCE_USER="postgres"  
$env:DATASOURCE_PASSWORD="bazepodataka"  
$env:FRONTEND_URL="http://localhost:3000"
```

### KORAK 2: PAKIRANJE APLIKACIJE

Sada kada je konfiguracija postavljena, aplikaciju trebamo pretvoriti u izvršni JAR fajl.

```
./mvnw package
```

Ova naredba pokreće Maven alat za upravljanje projektom. Preuzima sve potrebne biblioteke. Kompajlira Java kod i pakira ga u JAR fajl unutar target direktorija. Ovaj JAR fajl je samostalna, izvršna verzija vaše aplikacije.

### KORAK 3: IZRADA DOCKER SLIKE

Sljedeći korak je umetanje naše aplikacije u “kontejner” pomoću Docker-a. Kontejner omogućuje da aplikacija radi dosljedno na bilo kojem računalu. Docker će slijediti upute u Dockerfileu, preuzeti službenu Java sliku, kopirati vaš JAR fajl unutar nje i konfigurirati je za pokretanje.

```
docker build -t demo-deployment .
```

#### KORAK 4: DIJELJENJE DOCKER SLIKE NA DOCKER HUBU

Kako bismo aplikaciju mogli pokrenuti na serveru (npr. Render.com), potrebno je našu Docker sliku postaviti na javni registry.

```
docker tag demo-deployment vase_dockerhub_ime/demo-deployment:latest
```

Zamijenite vase\_dockerhub\_ime sa svojim stvarnim Docker Hub korisničkim imenom. Sada je sliku moguće gurnuti na Docker Hub.

```
docker push vase_dockerhub_ime/demo-deployment:latest
```

Ova naredba uploada označenu sliku na Docker Hub registry

#### KORAK 5: POSTAVLJANJE NA RENDER.COM

Sada kada je vaša slika na Docker Hubu, možete je pokrenuti na bilo kojem servisu koji podržava Docker, kao što je Render. \* Otvorite Render.com dashboard i kreirajte novu Web Service uslugu. \* Odaberite opciju “Deploy an existing image from a registry”. \* U polje “Image URL” unesite putanju do vaše slike: vase\_dockerhub\_ime/demo-deployment:latest \* Render će preuzeti vašu sliku i pokrenuti je. Sada vam je backend aplikacija dostupna na javnoj URL adresi koju će vam Render dodijeliti.

```
# Deployment frontend # Slijediti ove korake za uspješno postavljanje frontenda na Netlify i ažuriranje konfiguracije backenda aplikacije na Renderu.
```

#### KORAK 1: AŽURIRANJE URL-A BACKENDA U FRONTENDU

Prvo je potrebno reći frontend aplikaciji gdje se nalazi backend servis na Renderu. \* Otvorit .env datoteku u root direktoriju vašeg frontend projekta. \* Ažurirajte varijablu koja sadrži URL backend servisa:

```
REACT_APP_API_URL=https://vas-backend-app.onrender.com
```

Frontend aplikacija mora znati točnu adresu backend servisa kako bi mogla slati zahtjeve za podacima.

#### KORAK 2: PRIPREMA FRONTENDA ZA PRODUCTION

Prije deploymenta, frontend aplikaciju trebamo pretvoriti u optimiziranu, production-ready verziju.

```
cd client  
npm run build
```

npm run build - ova naredba: \* Optimizira JavaScript kod \* Kompajlira CSS i assets \* Kreira build direktorij sa statičkim fajlovima spremnima za serviranje \* Povećava performanse aplikacije za produkcijsko okruženje

### KORAK 3: DEPLOYMENT FRONTENDA NA NETLIFY

- Otvorite netlify.com i prijavite se
- Kliknite na “Add new site” → “Deploy manually”
- Učitajte build folder:

Netlify će automatski preuzeti i hostati statičke fajlove iz build direktorija. Dobit ćete jedinstveni URL (npr. <https://vas-app.netlify.app>) gdje je vaš frontend sada dostupan.

### KORAK 4: AŽURIRANJE BACKENDA NA RENDERU

Sada kada je frontend deployan, trebamo reći backendu da dopusti zahtjeve s ove nove adrese.

- Otvorite vaš backend projekt na render.com
- U dashboardu pronađite “Environment Variables” sekciju
- Ažurirajte FRONTEND\_URL varijablu: FRONTEND\_URL=https://vas-app.netlify.app

### KORAK 5: KONFIGURACIJA ENVIRONMENT VARIJABLJI NA NETLIFY

Kako bi frontend znao gdje šaljati API zahtjeve, potrebno je postaviti backend URL kao environment varijablu.

Na Netlify dashboardu, idite na: \* “Site settings” → “Environment variables” \* Dodajte novu varijablu: REACT\_APP\_API\_URL=https://room-rently-backend.onrender.com

Netlify će injicirati ovu varijablu u vašu frontend aplikaciju pri buildanju. Frontend će moći pristupiti ovoj varijabli kroz process.env.REACT\_APP\_API\_URL. Ovo omogućava frontendu da dinamički prilagodi endpointove ovisno o okruženju.

## DEPLOYMENT BAZE PODATAKA

Slijedeći koraci će opisati način hostanja i spajanja baze podataka s backendom.

### KORAK 1: STVARANJE SUPABASE RAČUNA

- Otvorite Supabase.com
- Registrirajte se putem Github-a (ili alternativom)

### KORAK 2: STVARANJE BAZE PODATAKA

- Nakon što ste stvorili račun dobit ćete opciju postavljanja passworda i imena za vaš projekt
- Nazovite projekt po želji

### KORAK 3: SPAJANJE SUPABASE BAZE I BACKENDA NA RENDER-U

- Uđite u vaš projekt na Supabase stranici

- Pri vrhu će biti button Connect
- Odaberite metodu session pooler dobit ćete nešto ovako  
(`postgresql://postgres.gnvyeggmbgjxtrndogb:[YOUR-PASSWORD]@aws-1-eu-west-1.pooler.supabase.com:5432/postgres`)
- Odite na Render.com projekt, lijevo na Environment tab i editajte varijable
- iz session pooler urla vadite sljedeće podatke  
(`postgresql://DATASOURCE_USER:[DATASOURCE_PASSWORD]@DATASOURCE_URL`)
- za DATASOURCE\_URL stavite `jdbc: + postgresql://aws-1-eu-west-1.pooler.supabase.com:5432/postgres`
- za DATASOURCE\_USER `postgres.gnvyeggmbgjxtrndogb`
- za DATASOURCE\_PASSWORD vaš password koji ste postavili prilikom postavljanja projekta i baze

#### KORAK 4: PROVJERA BAZE

- Pokretanjem backenda bi se na Supabase stranici pod Tables trebale pojaviti generirane tablice
- Ulogirajte se na stranicu putem Google accounta DefaultAdmina
- Uđite na admin page kroz button vidljiv na /main pageu
- Probajte dodati novu smještajnu jedinicu
- na Supabase.com odite pod Table Editor i otvorite tablicu unit u kojoj bi trebala biti vaša nova smještajna jedinica

Dio dokumentacije i objašnjenja koda izrađen je uz pomoć **ChatGPT (OpenAI)** za potrebe strukturiranja i pojašnjavanja tijeka deploymenta projekta. **Datum pristupa:** 2025-11-8 **Svrha:** Bolje formuliranje i jezično uređivanje teksta deploymenta projekta.

Sav sadržaj dobiven uz pomoć ChatGPT-a je samostalno provjerен, analiziran i po potrebi izmijenjen od strane autora. Alat nije korišten za generiranje cjelevitih dijelova rada bez nadzora, niti za donošenje zaključaka umjesto autora. Ova uporaba je u skladu s Etičkim kodeksom i pravilnikom Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu o korištenju alata temeljenih na umjetnoj inteligenciji.

## TIJEK IMPLEMENTIRANJA BACKEND

### BACKEND — SPRING BOOT IMPLEMENTACIJA

CILJ

Backend aplikacija služi za autentifikaciju korisnika putem Google računa, pohranu podataka o korisnicima u bazu podataka i omogućavanje osnovnih CRUD operacija nad entitetom **Person**. Aplikacija je izrađena pomoću Spring Boot okvira, koristi Spring Security za zaštitu ruta te podržava autentifikaciju pomoću Google OAuth 2.0 tokena.

## POKRETANJE BACKEND APLIKACIJE

### KORACI ZA POKRETANJE SPRING BOOT BACKENDA

1. Otvori projekt u **IntelliJ IDEA**.
2. Provjeri da su instalirane sve ovisnosti (ako nije, u terminalu projekta pokreni naredbu **mvn clean install**).
3. Otvori glavnu klasu projekta, najčešće se nalazi u **src/main/java/springboot/backend/SpringApplication.java**.
4. Desnim klikom na datoteku **BackendApplication.java** odaberite **Run ‘BackendApplication’**.
5. Nakon pokretanja, backend će po defaultu biti dostupan na adresi <http://localhost:8080>.

Spring Boot automatski pokreće ugrađeni Tomcat server koji sluša zahtjeve i obrađuje API pozive definirane u **PersonController** klasi.

## POSTAVLJANJE I POKRETANJE FRONTENDA

### INSTALACIJA DODATNIH PAKETA

U React frontend direktoriju **client**, potrebno je instalirati dodatnu biblioteku **jwt-decode** koja omogućuje dekodiranje Google ID tokena:

**npm install jwt-decode**

Ova naredba se mora pokrenuti prije **npm start**, jer frontend koristi **jwtDecode()** funkciju kako bi lokalno prikazao korisničke podatke (npr. ime i e-mail) nakon prijave putem Googlea.

### POKRETANJE FRONTENDA

1. U terminalu idu u mapu **client**:

**cd client**

2. Ako prvi put pokrećeš projekt, instaliraj sve ovisnosti:

**npm install**

3. Pokreni aplikaciju:

**npm start**

4. Frontend će se otvoriti u pregledniku na adresi <http://localhost:3000>.

## STRUKTURA BACKEND DIJELA PROJEKTA

Backend je organiziran u sljedeće osnovne pakete:

- **controller** – sadrži REST kontroler **PersonController** koji definira API rute.
- **config** – sadrži konfiguraciju sigurnosti (**SecurityConfig**).
- **model** – sadrži JPA entitet **Person** koji predstavlja korisnika u bazi.
- **repository** – sadrži sučelje **PersonRepo** koje proširuje **JpaRepository** i omogućava komunikaciju s bazom podataka.

Pojedinačne komponente sustava — controller, model i repository — već su detaljno opisane u dokumentu draftBaze.md. U ovom poglavlju naglasak je stavljen na implementacijski aspekt cijelog kupa backend sustava, s fokusom na sigurnosnu konfiguraciju, integraciju s Google OAuth2 autentifikacijom i način pokretanja aplikacije.

## PERSON MODEL (MODEL/PERSON.JAVA)

### FUNKCIJA

Klasa **Person** predstavlja entitet korisnika u bazi podataka. Svaki red tablice **person** sadrži osnovne podatke o korisniku prijavljenom putem Google računa.

### KOD I OBJAŠNJENJE

```
@Entity
@Table(name = "person")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private boolean isAdmin;

    @Column(nullable = false)
    private boolean isUser;

    @Column(nullable = false)
```

```

private boolean isOwner;

@Column(nullable = false)
private String name;

public Person(String email, boolean isAdmin, boolean isUser, boolean isOwner, String name) {
    this.email = email;
    this.isAdmin = isAdmin;
    this.isUser = isUser;
    this.isOwner = isOwner;
    this.name = name;
}

@OneToMany(mappedBy = "person", cascade = CascadeType.ALL, orphanRemoval = true)
private List<ApartmentReservation> apartmentReservations;

@OneToMany(mappedBy = "person", cascade = CascadeType.ALL, orphanRemoval = true)
private List<RoomReservation> roomReservations;
}

```

## OBJAŠNJENJE

Dodan je konstruktor koji omogućuje stvaranje instance klase **Person** bez potrebe za navođenjem ID-a (koji se automatski generira u bazi podataka). Koristi se kada se novi korisnik dodaje putem backend metode **addPerson()** u **PersonController** klasi.

Parametri konstruktora predstavljaju osnovne atribute korisnika:

- email — jedinstvena e-mail adresa korisnika dohvaćena iz Google JWT tokena,
- isAdmin — označava ima li korisnik administratorske ovlasti,
- isUser — označava je li korisnik registriran kao standardni korisnik,
- isOwner — označava je li korisnik u ulozi vlasnika smještaja,
- name — ime korisnika dohvaćeno iz Google JWT tokena.

Ovaj konstruktor pojednostavljuje inicijalizaciju objekata prilikom registracije korisnika, omogućujući brzu i čitljivu izradu entiteta bez potrebe za dodatnim postavljanjem vrijednosti nakon stvaranja objekta.

## PERSON CONTROLLER (CONTROLLER/PERSONCONTROLLER.JAVA)

## FUNKCIJA

**PersonController** je REST kontroler koji upravlja operacijama nad entitetom **Person**. Glavne funkcionalnosti:

- Dohvat svih korisnika (**GET /allPersons**)
- Dodavanje korisnika pri prijavi putem Google računa (**POST /addPerson**)
- Dohvat korisnika prema ID-u (**GET /{id}**)
- Brisanje korisnika (**DELETE /deletePerson/{id}**)

---

#### KOD I OBJAŠNJENJE

```
@RestController
@CrossOrigin(origins = "http://localhost:3000")
public class PersonController {

    @Autowired
    PersonRepo repo;

    @GetMapping("/allPersons")
    public List<Person> getAllPersons() {
        return repo.findAll();
    }

    @PostMapping("/addPerson")
    public ResponseEntity<?> addPerson(@AuthenticationPrincipal Jwt jwt) {
        String email = jwt.getClaimAsString("email");
        String name = jwt.getClaimAsString("name");

        if (email == null) {
            throw new RuntimeException("Invalid Google token: no email found.");
        }

        if (repo.findByEmail(email).isPresent()) {
            return ResponseEntity.status(409).body("User already exists");
        }

        Person person = new Person(email, true, false, false, name);
        repo.save(person);
        return ResponseEntity.ok("User added successfully");
    }

    @GetMapping("/{id}")
    public Optional<Person> getPersonById(@PathVariable Long id) {
        return repo.findById(id);
    }
}
```

```
}

@ResponseBody
public void deletePerson(@PathVariable Long id) {
    repo.deleteById(id);
}

}
```

---

## OBJAŠNJENJE

U klasi **PersonController** promjenjena je metoda **addPerson()**:

Ova metoda omogućuje dodavanje novog korisnika u bazu podataka nakon uspješne autentifikacije putem Google računa. Metoda koristi JWT token poslan iz frontenda (u Authorization zaglavlu HTTP zahtjeva) kako bi dohvatila podatke o korisniku.

Detaljno objašnjenje: \* **@PostMapping("/addPerson")** — definira REST endpoint koji prima POST zahtjeve na adresi /addPerson. \* **@AuthenticationPrincipal Jwt jwt** — omogućuje Springu da automatski prepozna i dekodira Google JWT token te ga proslijedi metodi u obliku objekta Jwt. \* **String email = jwt.getClaimAsString("email")** i **String name = jwt.getClaimAsString("name")** — dohvaćaju vrijednosti iz JWT tokena koje Google šalje nakon prijave (ime i e-mail korisnika). \* U slučaju da token ne sadrži e-mail, baca se iznimka jer je e-mail obavezan podatak za registraciju. \* Ako korisnik s istim e-mailom već postoji u bazi (**repo.findByEmail(email).isPresent()**), vraća se HTTP status 409 (Conflict) i poruka "User already exists". \* Ako korisnik ne postoji, stvara se novi objekt klase Person pomoću konstruktora new Person(email, true, false, false, name) te se spremi u bazu putem **repo.save(person)**. \* Nakon uspješnog dodavanja, backend vraća odgovor 200 OK s porukom "User added successfully".

Ova metoda čini ključnu poveznicu između frontend autentifikacije putem GoogleLogin komponente i backend baze podataka, čime omogućuje automatsko dodavanje korisnika prilikom njihove prve prijave.

## PERSONREPO (REPOSITORY/PERSONREPO.JAVA)

---

## FUNKCIJA

**PersonRepo** sučelje omogućuje komunikaciju između backend aplikacije i baze podataka za entitet **Person**. To je sloj za pristup podacima koji koristi Spring Data JPA kako bi olakšao rad s bazom bez potrebe za pisanjem SQL upita.

---

## KOD I OBJAŠNJENJE

```
package com.project.backend.repository;

import com.project.backend.model.Person;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource
```

```
e;

import java.util.Optional;

@RepositoryRestResource(path = "person")
public interface PersonRepo extends JpaRepository<Person, Long> {
    Optional<Person> findByEmail(String email);
}
```

---

## OBJAŠNJENJE

**PersonRepo** je sučelje koje nasljeđuje **JpaRepository** i omogućuje standardne CRUD operacije (Create, Read, Update, Delete) nad entitetom Person. Dodavanjem metode **findByEmail(String email)** proširuje se funkcionalnost rezpositorija kako bi omogućio pretragu korisnika prema njihovoj e-mail adresi.

## KOMUNIKACIJA S FRONTEND DIJELOM (REACT GOOGLELOGIN)

U React komponenti **Navbar.jsx** koristi se komponenta **GoogleLogin** iz biblioteke **@react-oauth/google**. Nakon uspješne prijave, frontend dohvata **credential** (JWT token) koji se šalje backendu:

```
<GoogleLogin
  onSuccess={async (credentialResponse) => {
    try {
      const idToken = credentialResponse.credential; // JWT ID token
      if (!idToken) {
        console.error("No ID token received from Google");
        return;
      }

      try {
        await axios.post(
          "http://localhost:8080/addPerson",
          {},
          {
            headers: {
              Authorization: `Bearer ${idToken}`,
            },
          }
        );
        console.log("User successfully added to DB");
      } catch (error) {
        if (error.response && error.response.status === 409) {
          console.warn("User already exists in database");
        } else {
          console.error("Error adding user to DB:", error);
        }
      }
    }
  }
}
```

```

        }
    }

    const decoded = jwtDecode(idToken);
    console.log("Decoded user:", decoded);

    setUser(decoded);
    localStorage.setItem("googleUser", JSON.stringify(decoded));
} catch (err) {
    console.error("Error processing Google login:", err);
}
}

onError={() => {
    console.log("Login Failed");
}}
/>

```

## OBJAŠNJENJE

1. Prijava — korisnik se autentificira putem Google računa.
2. Dohvat tokena — Google vraća JWT token (idToken) koji sadrži podatke o korisniku (ime, e-mail, slika).
3. Slanje backendu — token se šalje na POST /addPerson, gdje ga Spring Boot validira i sprema korisnika u bazu.
4. Dekodiranje — frontend pomoću jwt-decode čita podatke iz tokena i sprema ih u localStorage.
5. Rezultat — korisnik je prijavljen, a njegovi podaci su dostupni za prikaz u sučelju.

Ovaj proces omogućuje sigurnu i brzu autentifikaciju korisnika te automatsku sinkronizaciju s backend bazom.

## SECURITY KONFIGURACIJA (CONFIG/SECURITYCONFIG.JAVA)

## FUNKCIJA

**SecurityConfig** konfigurira pravila sigurnosti za REST API rute, onemogućava CSRF zaštitu (jer se koristi JWT) i omogućuje autentifikaciju putem Google OAuth2 JWT tokena.

## KOD I OBJAŠNJENJE

```

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf -> csrf.disable())

```

```

        .cors(cors -> cors.configurationSource(corsConfigurationSource()))
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/addPerson", "/deletePerson/**").authenticated()
            .anyRequest().permitAll()
        )
        .oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt -> {}));
    }

    return http.build();
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowedOrigins(List.of("http://localhost:3000"));
    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    config.setAllowedHeaders(List.of("Authorization", "Content-Type"));
    config.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", config);
    return source;
}
}

```

---

#### OBJAŠNJENJE SIGURNOSNIH POSTAVKI

- **CSRF isključen** jer se koristi stateless autentifikacija putem JWT tokena.
- **CORS konfiguracija** omogućava zahtjeve s frontenda (<http://localhost:3000>).
- **Autorizacija ruta:**
  - `/addPerson` i `/deletePerson/**` zahtijevaju autentifikaciju.
  - Sve ostale rute (`/allPersons`, `/id`) su javno dostupne.
- **OAuth2 Resource Server konfiguracija:** omogućuje da backend validira Googleov JWT token poslan u `Authorization` zaglavju.

---

#### JWT PROVJERA

Spring automatski validira JWT token pomoću Googleovog javnog ključa i, ako je token ispravan, omogućava pristup autentificiranim rutama.

Dio koda za Spring Security konfiguraciju izrađen je uz pomoć **ChatGPT (OpenAI)**. **Datum pristupa:** 2025-10-31 **Svrha:** Implementacija sigurnosne konfiguracije za backend sustav s OAuth2 autentifikacijom.

## DATOTEKA APPLICATION.PROPERTIES

U **src/main/resources/application.properties** potrebno je dodati sljedeću postavku kako bi Spring znao koristiti Googleov issuer URI prilikom provjere JWT tokena:

**spring.security.oauth2.resource-server.jwt.issuer-uri=https://accounts.google.com**

Ova postavka omogućuje Spring Securityju da automatski validira Google JWT tokene koristeći javne ključeve koje Google nudi na adresi <https://accounts.google.com>.

## CJELOKUPNI TIJEK AUTENTIFIKACIJE (FRONTEND ↔ BACKEND)

1. Korisnik klikne na „**Login with Google**“ u **Navbar.jsx**.
2. Google vraća **credential** (JWT token) nakon uspješne prijave.
3. Frontend šalje **POST** zahtjev na **http://localhost:8080/addPerson** s JWT tokenom u zaglavlju:

**Authorization: Bearer <JWT\_TOKEN>**

4. Backend (Spring Security) validira JWT token i dohvaća korisničke podatke (**email, name**).
5. Ako korisnik ne postoji u bazi, stvara se novi zapis u tablici **person**.
6. Ako korisnik već postoji, backend vraća HTTP status **409 Conflict**.
7. Nakon toga, korisnik ostaje prijavljen u frontendu, a podaci se pohranjuju u **localStorage**.

## ZAKLJUČAK

Ovim backend dijelom projekta ostvarena je potpuna integracija između React frontenda i Spring Boot backenda putem Google OAuth2 prijave. Backend se brine za validaciju JWT tokena, kontrolu pristupa i pohranu korisnika u bazu podataka. Cjelokupna komunikacija osigurana je standardnim sigurnosnim mehanizmima, dok je frontend zadužen za vizualnu autentifikaciju i prikaz korisničkog sučelja.

Dio dokumentacije i objašnjenja koda izrađen je uz pomoć **ChatGPT (OpenAI)** za potrebe strukturiranja i pojašnjavanja Spring Security konfiguracije. **Datum pristupa:** 2025-11-3 **Svrha:** Bolje formuliranje i jezično uređivanje teksta te objašnjenje funkcionalnosti backend sustava te integracija s Google OAuth2 autentifikacijom.

Sav sadržaj dobiven uz pomoć ChatGPT-a je samostalno provjeren, analiziran i po potrebi izmijenjen od strane autora. Alat nije korišten za generiranje cjelevitih dijelova rada bez nadzora, niti za donošenje zaključaka umjesto autora. Ova uporaba je u skladu s Etičkim kodeksom i pravilnikom Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu o korištenju alata temeljenih na umjetnoj inteligenciji.

# TIJEK IMPLEMENTIRANJA BAZE PODATAKA

## POČETAK INICIJALIZIRANJA BAZE PODATAKA (INSTALACIJE)

Koristio se spring initializr za generiranje Spring Boot projekta kako se ručno ne bi trebao pisati svaki dependency. U svrhu kreiranja Baze koristili su se sljedeći: Spring Web, Spring Boot Dev Tools, Rest Repositories, Spring Data JPA, Lombok i PostgreSQL Driver prema ovom [tutorijalu](#). Rad baze podataka pokreće se klikom tipke Run u klasi BackendApplication.

### RELACIJE

Relacije u Springu Bootu služe kako bi povezale entitete tako da se ponašaju kao prirodno povezani objekti u Javi, dok Spring Boot i baza podataka brinu da se te veze ispravno pohranjuju, dohvaćaju i održavaju u skladu s pravilima integriteta. Pokretanjem BackendApplicationa Hibernate čita entitete i kreira odgovarajuće tablice i strane ključeve prema relacijama. ## Relacija Person

```
package com.project.backend.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "person")
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(columnDefinition = "text", nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private boolean isAdmin;

    @Column(nullable = false)
    private boolean isUser;
```

```

    @Column(nullable = false)
    private boolean isOwner;

    @Column(columnDefinition = "text", nullable = false)
    private String name;

    @OneToMany(mappedBy = "person", cascade = CascadeType.ALL, orphanRemoval
    = true)
    private List<UnitReservation> unitReservations;

}

```

**@Data** generira gettere i setttere.

**@NoArgsConstructor** definira konstruktor bez argumenata.

**@AllArgsConstructor** definira konstruktor sa svim argumentima.

**@Entity** i **@Table** definiraju mapiranje klase na tablicu person. Polja **isAdmin**, **isUser** i **isOwner** označavaju uloge osobe ulogirane u aplikaciju.

**@Id** označava primarni ključ relacije, u ovom slučaju to je **id** osobe, a **strategy = GenerationType.IDENTITY** označava da će se **id** automatski inkrementirati dodavanjem osoba u tablicu.

Također, svaki stupac ima definirana svojstvo **@Column(nullable = false)** što označava da njegova vrijednost ne može biti null, a stupac **email** dodatno ima svojstvo **unique = true** što znači da svaka osoba mora imati jedinstven email.

**@OneToMany** veza povezuje korisnika s njegovim rezervacijama. Jedan korisnik može imati 0 ili više rezervacija.

**cascade = CascadeType.ALL** i **orphanRemoval = true** osiguravaju automatsko brisanje povezanih rezervacija kad se korisnik izbriše.

## RELACIJA UNIT

```

package com.project.backend.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.List;

@Data

```

```
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "unit")  
public class Unit {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idUnit;  
  
    @Column(nullable = false)  
    private Integer price;  
  
    @Column  
    private Integer numRooms;  
  
    @Column(nullable = false)  
    private Integer capAdults;  
  
    @Column(nullable = false)  
    private Integer capChildren;  
  
    @Column(nullable = false)  
    private Integer numBeds;  
  
    @Column(nullable = true)  
    private Integer rating;  
  
    @Column(nullable = false)  
    private boolean hasParking;  
  
    @Column(nullable = false)  
    private boolean hasWifi;  
  
    @Column(nullable = false)  
    private boolean hasBreakfast;  
  
    @Column(nullable = false)  
    private boolean hasTowels;  
  
    @Column(nullable = false)  
    private boolean hasShampoo;  
  
    @Column(nullable = false)
```

```

private boolean hasHairDryer;

@Column(nullable = false)
private boolean hasHeater;

@Column(nullable = false)
private boolean hasAirConditioning;

@Column(name = "is_apartment", nullable = false)
@JsonProperty("isApartment")
private boolean apartment;

@Column(columnDefinition = "text", nullable = false)
private String unitName;

@Column(columnDefinition = "text", nullable = true)
private String location;

@Column(columnDefinition = "text", nullable = false)
private String mainDescName;

@Column(columnDefinition = "text", nullable = false)
private String mainDescContent;

@Column(columnDefinition = "text", nullable = false)
private String secDescName;

@Column(columnDefinition = "text", nullable = false)
private String secDescContent;

@OneToMany(mappedBy = "unit", cascade = CascadeType.ALL, orphanRemoval = true)
private List<UnitImg> images;

@OneToMany(mappedBy = "unit", cascade = CascadeType.ALL, orphanRemoval = true)
private List<UnitReservation> unitReservations;
}

```

**@Data** generira gettere i setttere.

**@NoArgsConstructor** definira konstruktor bez argumenata.

**@AllArgsConstructor** definira konstruktor sa svim argumentima.

**@Entity i @Table** definiraju mapiranje klase na tablicu unit.

**@Id** označava primarni ključ relacije, u ovom slučaju to je id smještajne jedinice, a **strategy = GenerationType.IDENTITY** označava da će se id automatski inkrementirati dodavanjem smještajne jedinice u tablicu.

Sva polja imaju **@Column(nullable = false)**, osim **rating, numRooms, location**, što znači da nijedno od njih ne može biti null prilikom unosa podataka u tablicu. **numRooms** je atribut tipa optional zato što smještajna jedinica može biti soba pa za nju atribut za broj soba nema smisla, stoga se postavlja na null u tom slučaju.

Atributi **price, numRooms, capAdults, capChildren, numBeds, rating, hasParking, hasWifi, hasBreakfast, hasTowels, hasShampoo, hasHairDryer, hasHeater, hasAirConditioning, apartment, unitName, location, mainDescName, mainDescContent, secDescName, secDescContent** opisuju karakteristike apartmana. Polje **apartment** određuje je li smještajna jedinica apartman (true) ili soba (false). U bazi podataka se pohranjuje kao stupac `is_apartment`, dok se u JSON-u prikazuje kao `isApartment` (zahvaljujući anotaciji **@JsonProperty**).

**@OneToOne** veze povezuju smještajnu jedinicu s rezervacijama korisnika i slikama smještajne jedinice. Smještajna jedinica može imati 0 ili više rezervacija. Smještajna jedinica može imati 0 ili više slika.

**cascade = CascadeType.ALL** i **orphanRemoval = true** osiguravaju automatsko brisanje povezanih rezervacija kad se smještajna jedinica izbriše. ## Relacija UnitReservation

```
package com.project.backend.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "unitReservation")
public class UnitReservation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idUnitReservation;

    @Column(nullable = false)
    private LocalDate startDate;

    @Column(nullable = false)
    private LocalDate endDate;
}
```

```

private LocalDate endDate;

@Column(columnDefinition = "text", nullable = false)
private String status;

@ManyToOne
@JoinColumn(name = "id", nullable = false)
private Person person;

@ManyToOne
@JoinColumn(name = "idUnit", nullable = false)
private Unit unit;
}

```

**@Data** generira gettere i setttere.

**@NoArgsConstructor** definira konstruktor bez argumenata.

**@AllArgsConstructor** definira konstruktor sa svim argumentima.

**@Entity** i **@Table** definiraju mapiranje klase na tablicu rezervacije smještajne jedinice.

**@Id** označava primarni ključ relacije, u ovom slučaju to je id rezervacije smještajne jedinice, a **strategy = GenerationType.IDENTITY** označava da će se id automatski inkrementirati dodavanjem smještajne jedinice u tablicu.

Sva polja imaju **@Column(nullable = false)**, što znači da nijedno od njih ne može biti null prilikom unosa podataka u tablicu.

**startDate** i **endDate** predstavljaju datume početka i završetka rezervacije. Koristi se tip LocalDate jer omogućuje rad s datumima.

**status** Omogućuje praćenje rezervacija smještajne jedinice, je li smještajna jedinica u uporabi ili nije.

**@ManyToOne** veze povezuju rezervaciju smještajne jedinice s korisnikom i smještajnom jedinicom. Rezervacija smještajne jedinice može imati točno jednog korisnika koji je napravio rezervaciju. Rezervacija smještajne jedinice može imati točno jednu smještajnu jedinicu nad kojom je izvršena rezervacija. ## Relacija UnitImg

```
package com.project.backend.model;
```

```

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor

```

```
@AllArgsConstructor  
@Entity  
@Table(name = "unitImg")  
public class UnitImg {  
  
    @Id  
    @Column(columnDefinition = "text", nullable = false)  
    private String URL;  
  
    @ManyToOne  
    @JoinColumn(name = "idUnit", nullable = false)  
    private Unit unit;  
  
}
```

**@Data** generira gettere i setttere.

**@NoArgsConstructor** definira konstruktor bez argumenata.

**@AllArgsConstructor** definira konstruktor sa svim argumentima.

**@Entity** i **@Table** definiraju mapiranje klase na tablicu unitImg.

**@Id** označava primarni ključ relacije, u ovom slučaju to je id slike smještajne jedinice.

Sva polja imaju **@Column(nullable = false)**, što znači da nijedno od njih ne može biti null prilikom unosa podataka u tablicu.

Za pohranu slike koristi se **url** direktorija gdje se nalazi slika.

**ManyToOne** veza povezuje sliku smještajne jedinice sa smještajnom jedinicom. Slika smještajne jedinice pripada točno jednoj smještajnoj jedinici.

## REPOZITORIJI

### PERSONREPO

```
package com.project.backend.repository;  
  
import com.project.backend.model.Person;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.rest.core.annotation.RepositoryRestResource;  
  
import java.util.Optional;  
  
@RepositoryRestResource(path = "person")  
public interface PersonRepo extends JpaRepository<Person, Long> {
```

```
        Optional<Person> findByEmail(String email);  
    }
```

#### UNITREPO

```
package com.project.backend.repository;  
  
import com.project.backend.model.Unit;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.rest.core.annotation.RepositoryRestResource;  
  
import java.util.Optional;  
  
@RepositoryRestResource(path = "unit")  
public interface UnitRepo extends JpaRepository<Unit, Long> {  
    Optional<Unit> findByName(String unitName);  
}
```

#### UNITRESERVATIONREPO

```
package com.project.backend.repository;  
  
import com.project.backend.model.UnitReservation;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.rest.core.annotation.RepositoryRestResource;  
  
@RepositoryRestResource(path = "unitReservation")  
public interface UnitReservationRepo extends JpaRepository<UnitReservation, Long> {  
}
```

#### UNITIMGREPO

```
package com.project.backend.repository;  
  
import com.project.backend.model.UnitImg;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.rest.core.annotation.RepositoryRestResource;  
  
@RepositoryRestResource(path = "unitImg")  
public interface UnitImgRepo extends JpaRepository<UnitImg, String> {  
}
```

Repozitoriji u Spring Bootu su sučelja koja predstavljaju sloj za pristup podacima (Data Access Layer) i koriste se za komunikaciju s bazom podataka. Oni omogućuju čitanje, spremanje, brisanje i ažuriranje entiteta bez

potrebe za ručnim pisanjem SQL upita. Spring Data JPA automatski generira implementacije metoda iz repozitorija, što značajno pojednostavljuje rad s bazom.

Osnovne korišene anotacije su: + **@RepositoryRestResource(path = "naziv")** - označava da će se repozitorij automatski izložiti kao REST API endpoint, npr. path="person" znači da će podaci biti dostupni putem /person.

- **JpaRepository<Naziv, TipPodatka>** - osnovno sučelje koje pruža sve standardne CRUD operacije (Create, Read, Update, Delete) nad entitetima. Primjeri metoda koje automatski postoje:
  - save(entity) – sprema novi zapis ili ažurira postojeći
  - findAll() – dohvaća sve zapise
  - findById(id) – dohvaća zapis po primarnom ključu
  - deleteById(id) – briše zapis po primarnom ključu

U kodu se nalaze i neke dodatne metode koje će se objasniti u Wikiju Tijek implementiranja Backend. # Kontroleri ## PersonController

```
package com.project.backend.controller;

import com.project.backend.model.Person;
import com.project.backend.repository.PersonRepo;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@CrossOrigin(origins = "http://localhost:3000")
public class PersonController {

    @Autowired
    PersonRepo repo;

    @GetMapping("/allPersons")
    public List<Person> getAllPersons() {
        return repo.findAll();
```

```

    }

    @PostMapping("/addPerson")
    public ResponseEntity<?> addPerson(@AuthenticationPrincipal Jwt jwt) {
        String email = jwt.getClaimAsString("email");
        String name = jwt.getClaimAsString("name");

        if (email == null) {
            throw new RuntimeException("Invalid Google token: no email found.");
        };

        if (repo.findByEmail(email).isPresent()) {
            return ResponseEntity
                .status(409)
                .body("User already exists");
        }

        Person person = new Person(email, true, false, false, name);
        repo.save(person);
        return ResponseEntity.ok("User added successfully");
    }

    @GetMapping("/{id}")
    public Optional<Person> getPersonById(@PathVariable Long id) {
        return repo.findById(id);
    }

    @DeleteMapping("/deletePerson/{id}")
    public void deletePerson(@PathVariable Long id) {
        repo.deleteById(id);
    }
}

```

## UNITCONTROLLER

```

package com.project.backend.controller;

import com.project.backend.model.Unit;
import com.project.backend.repository.UnitRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

```

```
import java.util.List;
import java.util.Optional;

@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/unit")
public class UnitController {

    @Autowired
    private UnitRepo repo;

    @GetMapping("/all")
    public List<Unit> getAllUnits() {
        return repo.findAll();
    }

    @PostMapping("/add")
    public ResponseEntity<?> addUnit(@RequestBody Unit unit) {
        if (repo.findByName(unit.getUnitName()).isPresent()) {
            return ResponseEntity
                .status(409)
                .body("Unit already exists");
        }
        repo.save(unit);
        return ResponseEntity.ok("Unit added successfully");
    }

    @GetMapping("/{id}")
    public Optional<Unit> getUnitById(@PathVariable Long id) {
        return repo.findById(id);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteUnit(@PathVariable Long id) {
        repo.deleteById(id);
    }

    @PutMapping("/update/{id}")
    public ResponseEntity<?> updateUnit(@PathVariable Long id, @RequestBody Unit updatedUnit) {
        return repo.findById(id).map(existingUnit -> {
            updatedUnit.setIdUnit(id);
            repo.save(updatedUnit);
            return ResponseEntity.ok("Unit updated successfully");
        }).orElseGet(() -> ResponseEntity.notFound().build());
    }
}
```

```
    }
}
```

## UNITRESERVATIONCONTROLLER

```
package com.project.backend.controller;

import com.project.backend.model.UnitReservation;
import com.project.backend.repository.UnitReservationRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/unitReservation")
public class UnitReservationController {

    @Autowired
    private UnitReservationRepo repo;

    @GetMapping("/all")
    public List<UnitReservation> getAllReservations() {
        return repo.findAll();
    }

    @PostMapping("/add")
    public ResponseEntity<?> addReservation(@RequestBody UnitReservation res) {
        repo.save(res);
        return ResponseEntity.ok("Reservation added successfully");
    }

    @GetMapping("/{id}")
    public Optional<UnitReservation> getReservationById(@PathVariable Long id) {
        return repo.findById(id);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteReservation(@PathVariable Long id) {
        repo.deleteById(id);
    }
}
```

```
    }
}
```

## UNITIMGCONTROLLER

```
package com.project.backend.controller;

import com.project.backend.model.UnitImg;
import com.project.backend.repository.UnitImgRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/unitImg")
public class UnitImgController {

    @Autowired
    private UnitImgRepo repo;

    @GetMapping("/all")
    public List<UnitImg> getAllImages() {
        return repo.findAll();
    }

    @PostMapping("/add")
    public ResponseEntity<?> addImage(@RequestBody UnitImg img) {
        if (repo.existsById(img.getURL())) {
            return ResponseEntity
                .status(409)
                .body("Image already exists");
        }
        repo.save(img);
        return ResponseEntity.ok("Image added successfully");
    }

    @GetMapping("/{url}")
    public Optional<UnitImg> getImageByUrl(@PathVariable String url) {
        return repo.findById(url);
    }
}
```

```
@DeleteMapping("/delete/{url}")
public void deleteImage(@PathVariable String url) {
    repo.deleteById(url);
}
```

Kontroleri su klase koje predstavljaju REST API sloj aplikacije, komuniciraju s frontendom (React na localhost:3000), koriste repozitorije za pristup bazi, vraćaju podatke u JSON formatu, koriste Spring anotacije (@RestController, @Autowired, @GetMapping, itd.).

Osnovne korištene anotacije su: + **@RestController** - kombinira **@Controller** i **@ResponseBody**. Znači da klasa služi kao REST endpoint i da sve metode automatski vraćaju podatke u JSON formatu + **@CrossOrigin(origins = "http://localhost:3000")** - dozvoljava da zahtjevi s React frontenda (koji se izvršava na portu 3000) pristupaju backendu + **@Autowired** - automatski ubacuje dependency, npr. repo – Spring sam kreira instancu repozitorija + **@GetMapping, @PostMapping, @DeleteMapping** - označavaju HTTP metode koje kontroler sluša (GET, POST, DELETE) + **@RequestBody** - označava da metoda prima tijelo HTTP zahtjeva (JSON) + **@PathVariable** - označava da metoda koristi dio URL-a kao parametar + **@AuthenticationPrincipal Jwt jwt** - (samo u PersonController) – prima Google JWT token i izvlači korisnikove podatke

U kodu se nalaze i neke dodatne metode koje će se objasniti u Wikiju Tijek implementiranja Backend.

## TIJEK IMPLEMENTIRANJA FRONTEND

### POČETAK PROJEKTA(INSTALACIJE)

Prvo imamo problem kako započeti projekt. Potrebno je bilo kreirati mapu u kojoj će biti sadržan naš projekt. Pozicionirani u nju potrebno je u terminalu upisati naredbu:

**npx create-react-app client**

gdje client predstavlja ime naše aplikacije. Potrebno je pozicionirati se u navedenu mapu client naredbom u terminalu:

**cd client**

te kako bi pokrenuli našu aplikaciju glavna naredba za to jest:

**npm start**

tu naredbu je potrebno učestalo koristiti tijekom debugiranja kako bi vidjeli našu aplikaciju. Ako skidamo ovaj projekt, potrebno je prije naredbe **npm start** upisati naredbu:

**npm install**

Po defaultu naša aplikacija bi trebala biti vidljiva na adresi <http://localhost:3000> ali u terminalu će također biti ispisana adresa za korištenje. Nakon toga potrebno je unijeti adresu u bilo koji lokalni browser primjerice Google Chrome. Po defaultu dobili smo automatsku instaliranu React frontend stranicu koja ima logo i opis, ali služi isključivo za primjer. Većinu toga je bilo potrebno pobrisati iz predefiniranih template datoteka kako bi

mogli napredovati sa svojim idejama. Pobrisao sam default HTML-ove iz App.js te testirao tako da napravim par h1 tagova u HTML-u unutar App.js te spremio file nakon čega mi se pojavio h1 tag na Google Chrome-u na lokalnoj stranici.

```
<h1>Test</h1>
```

## POČETNA STRANICA(LANDING SCREEN)

Nakon instalacije svega potrebnoga za početak projekta, trebalo je napraviti početnu stranicu.

Instalirao sam react-router-dom s pomoću naredbe:

```
npm i react-router-dom
```

te sam zatim u App.js implementirao kod:

```
import LandingScreen from "./screens/Landingscreen";

import { BrowserRouter, Routes, Route } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<LandingScreen />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

koji omogućava da na ruti <http://localhost:3000/> imamo početni zaslon(LandingScreen).

Zatim sam u mapi src stvorio novu mapu screens u kojoj sam napravio datoteke LandingScreen.jsx i LandingScreen.css koje će definirati izgled početne stranice.

## REACT + CSS ANIMACIJA

Ova komponenta predstavlja početni (landing) ekran aplikacije “Room-Rently”. Korisniku prikazuje naslov, podnaslov i gumb koji vodi na glavnu stranicu (/main).

---

### REACT KOMPONENTA: LANDINGSCREEN.JSX

```
import React from "react";
import "./Landingscreen.css";
import { useNavigate } from "react-router-dom";
function LandingScreen() {
```

```

const navigate = useNavigate();
const navigatemain = () => {
  navigate("/main");
};

return (
  <div className="row landing">
    <div className="col-md-12">
      <h2>Room-Rently</h2>
      <h1>
        "Find your perfect stay from cozy rooms to modern apartments. Simpl
e
        search and effortless booking with Room-Rently."
      </h1>

      <button onClick={navigatemain}>Find a Place to Stay</button>
    </div>
  </div>
);
}

export default LandingScreen;

```

Dakle, `useNavigate()` je hook iz React Routera koji omogućava navigaciju između ruta bez reloadanja stranice. `navigate("/main")` preusmjerava korisnika na rutu /main. JSX struktura koristi Bootstrap klase (row, col-md-12) za osnovni layout, i custom klasu `landing` za stiliziranje pozadine. Klasa `row` označava red, a `col-md-12` znači da stupac zauzima cijelu širinu(12/12) na ekranima srednje veličine i većim ekranima. Sve animacije i izgled definirani su u Landingscreen.css-u.

#### CSS STILOVI: LANDINGSCREEN.CSS

##### POZADINA I OSNOVNI LAYOUT

```

.landing {
  height: 100vh;
  width: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  background: linear-gradient(135deg, #2b5876, #4e4376);
  color: white;
  text-align: center;
  position: relative;
  overflow: hidden;
  animation: fadeIn 1.5s ease-in-out;
}

```

Kreira fullscreen gradient pozadinu s plavo ljubičastim tonovima. Koristi se Flexbox za centriranje sadržaja po sredini ekrana te ulaznu animaciju odnosno fadeln za efekt postepenog pojavljivanja.

**overflow: hidden;** sprečava da rotirajući pseudo-element (::before), koji je veći od samog prozora vidi izvan prozora. Znači da se svi dijelovi pseudo-elementa koji izlaze iz granica .landing sekcije ne prikazuju.

**animation: fadeIn 1.5s ease-in-out;** animacija primjenjuje efekt postepenog pojavljivanja cijelog zaslona odmah po učitavanju stranice. ease-in-out znači da animacija započinje i završava sporije, a brža je u sredini odnosno glatki prijelaz.

---

#### EFEKT LEBDEĆEG SVJETLA U POZADINI POSTIGNUT CSS-OM

```
.landing::before {
    content: "";
    position: absolute;
    top: -50%;
    left: -50%;
    width: 200%;
    height: 200%;
    background: radial-gradient(
        circle,
        rgba(255, 255, 255, 0.1) 0%,
        transparent 70%
    );
    animation: rotateBackground 20s linear infinite;
    z-index: 0;
}
```

Koristi se pseudo element **::before** da doda prozirni sloj svjetla te radial-gradient stvara svjetlosni krug koji se polako rotira s pomoću:

```
@keyframes rotateBackground {
    0% {
        transform: rotate(0deg);
    }
    100% {
        transform: rotate(360deg);
    }
}
```

---

#### TEKSTUALNI ELEMENTI

```
.landing h2 {
    font-size: 3rem;
    letter-spacing: 2px;
    font-weight: 700;
    margin-bottom: 1rem;
```

```
    animation: slideDown 1s ease forwards;
}

.landing h1 {
  font-size: 1.8rem;
  font-weight: 400;
  color: #f8f8f8;
  max-width: 700px;
  margin: 0 auto 2rem auto;
  animation: fadeInUp 1.5s ease forwards;
}
```

h1 ima animaciju fadeInUp, lagano izlazi odozdo te h2 ima animaciju slidedown spuštanja s vrha.

```
animation: fadeInUp 1.5s ease forwards;
```

forwards znači da završno stanje animacije ostaje i nakon što se animacija završi. Bez toga bi se elementi vratili u početno stanje odnosno opet nestali nakon animacije.

---

## ANIMACIJE

```
@keyframes slideDown {
  from {
    opacity: 0;
    transform: translateY(-30px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

```
@keyframes fadeInUp {
  from {
    opacity: 0;
    transform: translateY(30px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

```
@keyframes fadeIn {
  from {
    opacity: 0;
```

```

        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

```

Objašnjenje animacija: **from** je početak animacije te nakon toga **opacity: 0** znači da je element potpuno nevidljiv. **transform: translateY(-30px)** nam zatim govori da je pomaknut 30px prema gore izvan svoje normalne pozicije. **to** označava kraj animacije: **opacity: 1** odnosno element postaje potpuno vidljiv te se s pomoću **transform: translateY(0)** vraća se svoju početnu poziciju. Dakle, postepeno element ide prema dolje(iли gore u slučaju fadeInUp) i postaje vidljiv.

Kada se fadeIn animacija pokrene, element kreće nevidljiv i malo spušten dolje. Tijekom trajanja animacije postepeno se pojavljuje (opacity od 0 do 1) te istovremeno klizi prema gore na svoje normalno mjesto (translateY(20px → 0)).

#### GUMB (“FIND A PLACE TO STAY”)

```

.landing button {
    background: linear-gradient(90deg, #ffb347 0%, #ffcc33 100%);
    color: #2b2b2b;
    font-size: 1.1rem;
    font-weight: 600;
    padding: 0.8rem 2rem;
    border: none;
    border-radius: 50px;
    cursor: pointer;
    transition: all 0.3s ease;
    box-shadow: 0 5px 15px rgba(255, 204, 51, 0.3);
    animation: fadeIn 2s ease forwards;
}

.landing button:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 20px rgba(255, 204, 51, 0.4);
}

```

Koristi gradijentnu pozadinu u žuto-narančastim tonovima te ima zaobljene rubove i sjenu za 3D efekt. **::hover** dodaje efekt izbočenja, a fadeIn animacija na gumb se pojavi postepeno s lakoćom(glatko) te ostane na mjestu nakon završetka animacije.

**box-shadow: 0 5px 15px rgba(255, 204, 51, 0.3);** dodaje sjenu ispod gumba, što daje osjećaj dubine. Pri hover-u se sjena povećava, što dodatno pojačava dojam da gumb “iskače” prema korisniku.

**transition: all 0.3s ease;** osigurava da promjene pri hover-u kao što su transform i box-shadow ne budu trenutačne, nego imaju efekt kroz vrijeme. Bez transition-a, efekt “iskakanja” bio bi nagao.

```
.col-md-12 {  
    z-index: 1;  
}
```

Ovo osigurava da se glavni sadržaj teksta i gumb prikazuju iznad animirane pozadine jer pseudo-element ima z-index: 0. Nazvan je po bootstrapu, ali nije korišten tako.

---

## RESPONZIVNOST

```
@media (max-width: 768px) {  
    .landing h2 {  
        font-size: 2.2rem;  
    }  
  
    .landing h1 {  
        font-size: 1.2rem;  
        padding: 0 1rem;  
    }  
  
    .landing button {  
        font-size: 1rem;  
        padding: 0.7rem 1.8rem;  
    }  
}
```

Automatski prilagođava veličine fontova i padding gumba za manje ekrane. Dio CSS koda u ovom repozitoriju je unaprijeđen s pomoću ChatGPT (OpenAI).

Upit korišten za dobivanje komponenti CSS-a je dobiven upitom “Potrebno mi je poboljšanje i unaprjeđenje izgleda CSS-a tako da ima plavo obojenu animaciju, prilažem ti html klase koje je potrebno koristiti” te su naredane klase navedene gore u kodu primjerice “.landing”, radi redundancije ne navodim. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepše izgleda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-11 **Svrha:** Unapređenje CSS-a refraktor, optimizacija selektora, poboljšanje responzivnosti.

## DODAVANJE OSTALIH STRANICA TE NJIHOVIH RUTA

Osim početne stranice potrebno je napraviti i druge stranice koje će nam biti potrebne u daljnjoj izradi projekta. Pratili smo YouTube tutorial koji nam je pomogao pri izradi frontenda, poveznica se nalazi ovdje :

[React Booking | Reservation App UI Design for Beginners](#). U mapi src stvorio sam novu mapu pages te u njoj mape home, list i hotel. U njima sam napravio odgovarajuće .jsx i .css datoteke, pa sam tako za home napravio Home.jsx i home.css te isto i za druge mape. S pomoću ekstenzije “ES7+ React/Redux/React-Native snippets” za VSCode, automatskom nadopunom koda, samim upisom “rafce” u Home.jsx dobio sam sljedeći predložak koda:

```
import React from "react";
const Home = () => {
  return <div>Home</div>;
};
export default Home;
```

koji na stranici prikazuje samo riječ Home. Zatim sam isto napravio i za List.jsx i Hotel.jsx te sam u App.js dodao rute:

```
import Home from "./pages/home/Home";
import List from "./pages/list/List";
import Hotel from "./pages/hotel/Hotel";
```

te u samu funkciju:

```
<Route path="/main" element={<Home />}></Route>
<Route path="/hotels" element={<List/>}></Route>
<Route path="/hotels/:id" element={<Hotel/>}></Route>
```

kako bi se dodavanjem puteva(path) na adresu <http://localhost:3000/> prikazivale sve tri nove stranice.

Pobrisao sam import React from 'react' iz svih datoteka jer nam to nije potrebno te sam umjesto toga povezao sve .jsx datoteke s njihovim odgovarajućim .css datotekama kako bi te stranice dobile nekakav izgled. Primjer koda u Home.jsx:

```
import "./home.css";
```

## GLAVNA STRANICA

### DODAVANJE NAVIGACIJSKE TRAKE

U src mapu dodao sam mapu components te u njoj mapu navbar s odgovarajućim .jsx i .css datotekama te napravio isti template kao i za rute u mapi pages. U Home.jsx sam umjesto same riječi Home stavio **<Navbar></Navbar>** koja sad pokazuje komponentu navigacijske trake te je bilo potrebno prenijeti tu navigacijsku traku iz komponenti s pomoću putanje naredbom:

```
import Navbar from "../../components/navbar/navbar";
```

U Navbar.jsx pod return dodao sam samu strukturu navigacijske trake koja nalikuje na HTML kod s time da se imena klasa označavaju s **className="imeKlase"**:

```
import "./navbar.css";
import { useNavigate } from "react-router-dom";
const Navbar = () => {
  const navigate = useNavigate();
  const navigatelandingscreen = () => {
    navigate("/");
  };
};
```

```

    return (
      <div className="navbar">
        <div className="navContainer">
          <span className="logo" onClick={navigateLandingscreen}>
            Room-Rently
          </span>
          <div className="navItems">
            <button className="navButton">Login</button>
            <button className="navButton">Logout</button>
          </div>
        </div>
      </div>
    );
}
export default Navbar;

```

Dakle dodan je blok element div za navigacijsku traku, container koji zaokružuje sve elemente navigacijske trake, linijski element span koji prikazuje ime stranice te div koji sadrži gumbu na prijavu i odjavu korisnika. Kao što je već prije objašnjeno, `useNavigate()` je hook iz React Routera te se s pomoću njega, klikom na naslov stranice, zbog `onClick={navigateLandingscreen}` prebacujemo na početnu stranicu.

Izgled navigacijske trake osmišljen je s pomoću CSS-a:

```

.navbar {
  height: 60px;
  background: linear-gradient(90deg, #0077ff, #00c6ff);
  display: flex;
  justify-content: center;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  position: sticky;
  top: 0;
  z-index: 100;
}

.navContainer {
  width: 100%;
  max-width: 1100px;
  color: white;
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0 20px;
}

```

.navbar{} određuje gdje će se nalaziti navContainer te koliko će velik biti navbar, koje boje, na kojoj poziciji u odnosu na vrh stranice te pozicijom sticky ostavljamo navigacijsku traku u vijek na vrhu bez obzira pomičemo li se prema dnu stranice. Tu nam još pomaže z-index koji nam govori da navigacijska traka mora biti ispred drugih elemenata jer ima veći z-index. Veličina navContainer-a je 100 % širine stranice, no ograničena je na maksimalnu širinu od 1100 piksela. Boja slova je bijela te su elementi containera pozicionirani tako da između imaju prazan prostor. Dakle naziv stranice je na krajnje lijevoj poziciji te su gumbi na krajnje desnoj poziciji s time da postoji i razmak lijevo i desno od 20 piksela od granica containera i njegovog sadržaja.

```
.logo {  
    font-weight: 600;  
    font-size: 1.5rem;  
    letter-spacing: 1px;  
    cursor: pointer;  
    transition: transform 0.3s ease, color 0.3s ease;  
}  
  
.logo:hover {  
    transform: scale(1.05);  
    color: #e0f7ff;  
}
```

Određena je veličina slova teksta imena stranice te kada predemo mišem preko njega, strelica se pretvori u oblik pokazivača što nam daje do znanja da se može kliknuti na njega. Također, :hover definira povećanje teksta te boju naslova prelaskom mišem, dok je u .logo{} definirano koliko dugo će trebati da se postigne to povećanje.

```
.navItems {  
    display: flex;  
    align-items: center;  
}  
  
.navButton {  
    margin-left: 15px;  
    border: none;  
    padding: 8px 18px;  
    border-radius: 25px;  
    cursor: pointer;  
    font-weight: 500;  
    color: #0077ff;  
    background-color: #ffffff;  
    transition: all 0.3s ease;  
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.15);  
}  
  
.navButton:hover {
```

```

background-color: #e0f7ff;
transform: translateY(-2px);
box-shadow: 0 4px 10px rgba(0, 119, 255, 0.3);
}

.navButton:active {
  transform: scale(0.97);
  box-shadow: 0 2px 6px rgba(0, 119, 255, 0.2);
}

```

.navItems{} određuje kako će gumbovi biti raspoređeni u njihovom containeru, dakle vodoravno te na srednjoj visini. .navButton{} osmišljen je s istim već objašnjenim funkcionalnostima kao i gumb na početnoj stranici. Određene su boje, razmak, veličina fonta, pokazivač miša te margine i obrub. :hover definira da će prelaskom mišem gumb otići prema gore, a :active da će se malo smanjiti klikom na njega.

## DODAVANJE GOOGLE PRIJAVE/ODJAVE U NAVIGACIJSKU TRAKU

Kako bismo omogućili autentifikaciju korisnika putem Google računa, u komponentu navigacijske trake Navbar.jsx dodan je kod koji koristi biblioteku **@react-oauth/google** za prijavu i odjavu, te biblioteku **axios** za dohvat podataka o korisniku. Na ovaj način korisnik se može prijaviti s pomoću Google računa, a njegovo ime i profilna slika prikazuju se u navigacijskoj traci. Za instalirati navedene pakete potrebno je unijeti ove dvije naredbe u radnom direktoriju:

```
npm install @react-oauth/google@latest
```

te

```
npm install axios
```

Na početku su uvezeni potrebni moduli:

```
import { useState, useEffect } from "react";
import { useGoogleLogin, googleLogout } from "@react-oauth/google";
import axios from "axios";
```

**useState** i **useEffect** su React hook-ovi za upravljanje lokalnim stanjem i efektima unutar komponente. **useGoogleLogin** i **googleLogout** dolaze iz biblioteke **@react-oauth/google** i koriste se za upravljanje procesima prijave i odjave. **axios** je HTTP klijent koji se koristi za dohvati korisničkih podataka s Googleovog API-ja nakon što korisnik potvrdi prijavu. Ispod **const navigate = useNavigate();** dodan je sljedeći kod:

```
// Loadanje usera iz local storagea
const [user, setUser] = useState(() => {
  const savedUser = localStorage.getItem("googleUser");
  return savedUser ? JSON.parse(savedUser) : null;
});
```

Ovaj kod koristi React hook `useState()` za stvaranje stanja user koje će čuvati informacije o trenutačno prijavljenom korisniku. Kako bi se omogućilo da korisnik ostane prijavljen i nakon što osvježi stranicu, početna vrijednost user nije prazna, već se dohvata iz localStorage-a odnosno trajna pohrana u pregledniku.

Dio CSS koda u ovom repozitoriju je također unaprijeden s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprijedenje izgleda CSS-a tako da ima plavu boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu primjerice "navButton", ostatak je gore u kodu radi redundancije ne navodim, kao i na prethodnom primjeru. Al mi je poboljšao moj CSS te nadogradio tako da vizualno ljepše izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-12

**Svrha:** Unapređenje CSS-a refraktor, poboljšanje responzivnosti.

---

#### GOOGLE PRIJAVA KORISNIKA

Za implementaciju prijave putem Google računa koristi se hook `useGoogleLogin()` iz biblioteke `@react-oauth/google`. Ovaj hook vraća funkciju login, koja se poziva prilikom klika na gumb "Login with Google". Kod izgleda ovako:

```
const login = useGoogleLogin({
  onSuccess: async (tokenResponse) => {
    try {
      const res = await axios.get(
        "https://www.googleapis.com/oauth2/v3 userinfo",
        {
          headers: { Authorization: `Bearer ${tokenResponse.access_token}` },
        }
      );
      setUser(res.data);
      localStorage.setItem("googleUser", JSON.stringify(res.data));
      console.log("User info:", res.data);
    } catch (err) {
      console.error("Error fetching user info:", err);
    }
  },
  onError: (error) => console.log("Login Failed:", error),
});
```

Funkcija `useGoogleLogin()` otvara Googleov prozor za autentifikaciju. Nakon što se korisnik uspješno prijavi, Google vraća access\_token jedinstveni token koji dokazuje da je korisnik autentificiran.

Zatim se koristi Axios za slanje GET zahtjeva prema Google API-u na `adresu`. Ova adresa vraća osnovne podatke o korisniku (ime, e-mail adresu, profilnu sliku,...), a token se šalje u zaglavljtu zahtjeva (Authorization header) u formatu: `Authorization: Bearer` Nakon što Google vrati odgovor res.data sadrži objekt s korisničkim podacima, s pomoću `setUser(res.data)` ti se podaci spremaju u trenutačno stanje

komponente, a s pomoću `localStorage.setItem("googleUser", JSON.stringify(res.data))` podaci se trajno pohranjuju u preglednik, čime se osigurava da korisnik ostane prijavljen i nakon ponovnog učitavanja stranice.

Ako dođe do greške (npr. korisnik prekine prijavu ili token istekne), poruka o grešci ispisuje se u konzolu s pomoću `console.log("Login Failed:", error)`.

---

## GOOGLE ODJAVA KORISNIKA

Za omogućavanje odjave korisnika koristi se funkcija `logout`, koja kombinira Google logout funkcionalnost, resetiranje lokalnog stanja i brisanje podataka iz preglednika:

```
const logout = () => {
  googleLogout();
  setUser(null);
  localStorage.removeItem("googleUser");
};
```

`googleLogout()` poziva funkciju iz biblioteke `@react-oauth/google`, koja briše sesiju korisnika na strani Googlea i efektivno ga odjavljuje. `setUser(null)` resetira stanje usera unutar React komponente, čime se u navigacijskoj traci ponovno prikazuje gumb Login with Google umjesto podataka o korisniku.

`localStorage.removeItem("googleUser")` uklanja spremljene podatke iz `localStorage`-a, osiguravajući da se korisnički podaci ne zadrže prilikom navigacije između ruta ili osvježavanja stranice.

---

## DINAMIČKI PRIKAZ KORISNIČKIH ELEMENATA U NAVIGACIJSKOJ TRACI

Ovaj blok koda koji je zamijenio statički div element klase `navItems` koristi uvjetno renderiranje `(!user ? ... : ...)` kako bi se sadržaj navigacijske trake prilagodio statusu prijave korisnika:

```
<div className="navItems">
  {!user ? (
    <button className="navButton" onClick={() => login()}>
      Login with Google
    </button>
  ) : (
    <>
      <img
        src={user.picture}
        alt="profile"
        style={{
          width: "35px",
          height: "35px",
          borderRadius: "50%",
          marginRight: "10px",
        }}
        referrerPolicy="no-referrer"
    />
```

```
<span>{user.name}</span>
<button className="navButton" onClick={logout}>
  Logout
</button>
</>
)
</div>
```

Ako korisnik nije prijavljen (**!user**), prikazuje se gumb „Login with Google“. Klikom na gumb poziva se funkcija **login()**, koja otvara Google prozor za autentifikaciju. Inače, ako je prijavljen, prikazuje se profilna slika korisnika i ova inline CSS svojstva: veličina slike je  $35 \times 35$  piksela, rubovi su zaobljeni (**borderRadius: "50%"**), razmak između slike i imena korisnika je (**marginRight: "10px"**), ispisuje se ime korisnika (**{user.name}**) koji je dobiven gore iz dohvata sa Google API-ja te se prikazuje gumb za odjavu (Logout), koji poziva funkciju **logout()** i briše korisničke podatke iz stanja i localStorage-a.

---

## NAVIGACIJSKA TRAKA UPDATE NAKON BACKEND UPDATEA

U novoj verziji dodan je **<div className="custom-google-login">** oko GoogleLogin komponente:  
**jsx <div className="custom-google-login"> <GoogleLogin/> </div>** Razlog dodavanja je što omogućuje lakše stiliziranje logina u CSS-u (navbar.css) te daje mogućnost dodavanja dodatnog izgleda po potrebi.

---

## DODATNI PROPS ZA GOOGLELOGIN

U drugoj verziji dodani su sljedeći props za GoogleLogin komponentu:

```
useOneTap
theme="filled_blue"
shape="rectangular"
text="signin_with"
size="medium"
width="200"
locale="en"
```

Objašnjenja svakog: \* **useOneTap** omogućuje Google One Tap login pojavljuje se mali prompt za brži login. \* **theme="filled\_blue"** promjena vizualnog izgleda gumba na plavi. \* **shape="rectangular"** Gumb je pravokutnog oblika umjesto default. \* **text="signin\_with"** Prikazuje tekst “Sign in with Google”. \* **size="medium"** Određuje veličinu gumba na srednju. \* **width="200"** Postavlja fiksnu širinu gumba na 200px. \* **locale="en"** Prikazuje gumb na engleskom jeziku.

Poboljšava korisničko sučelje i omogućuje veću kontrolu nad izgledom i ponašanjem Google login gumba.

---

## CSS UPDATE ZA POBOLJŠANJE LOGIN BUTTONA

Kao što sam gore naveo prvotno sam stvorio wrapper oko buttona ovdje sam ga uredio u CSS kodu:

```
.custom-google-login {  
    display: flex;  
    align-items: center;  
    transform: translateY(-7px);  
}
```

Nadodao sam efekte za wrapper oko buttona koji postižu ovo:  
\* Dodaje wrapper oko Google login gumba te poravnava gumb vertikalno s ostalim elementima u navbaru.  
\* transform: translateY(-7px) lagano podiže gumb kako bi bio u istoj liniji s ostalim elementima.

Stilovi za Google gumb unutar iframe-a:

```
.custom-google-login iframe {  
    margin: 0 !important;  
    border-radius: 4px !important;  
}
```

Uklanja marginu unutar iframe-a koju Google automatski dodaje te dodaje zaobljene rubove **border-radius: 4px** daju gumbu lješi izgled. Također !important prisiljava da se ignoriraju default vrijednosti odnosno prethodno definirani stilovi tako da osiguram update.

Poboljšanja vizualnog izgleda .navButton i .navItems (prethodne verzije CSS-a su ostavljene u kodu kako bi se moglo pratiti napredak i kako bi ostalo u toku s dokumentacijom)

Dodan gap: 15px u .navItems radi ravnomjernog razmaka između elemenata:

```
.navItems {  
    display: flex;  
    align-items: center;  
    gap: 15px;  
}
```

Stari stil gumba imao je plavo bijelu kombinaciju, novi je ažuriran na:

```
.navButton {  
    padding: 8px 16px;  
    background-color: #0071c2;  
    color: white;  
    border: none;  
    border-radius: 4px;  
    cursor: pointer;  
    font-weight: 500;  
}  
.navButton:hover {  
    background-color: #005a9e;  
}
```

Novi dizajn gumba bolje se slaže s plavim gradientom navbar-a te je poboljšan kontrast i hover efekt (tamnija plava prilikom hovera).

Nove promjene su poboljšale izgled nakon Backend updatea i promjene funkcionalnosti buttona stoga se kod morao prilagoditi.

## IZRADA ZAGLAVLJA

### DODAVANJE MAPI TE UVEZIVANJE U HOME.JSX

Pod mapu components dodao sam mapu header s odgovarajućim .css i .jsx datotekama. Kao svaku komponentu dodao sam Header u Home.jsx s pomoću import naredbe:

```
import Header from "../../components/header/Header";
```

te u samu strukturu istim načinom kao i za navbar <Header></Header> ispod odgovarajuće implementacije navbar-a.

### KARTICE APARTMENTS I ROOMS

Kako bi razlikovali apartmane i sobe, dodali smo kartice Apartments i Rooms. Klikom na Rooms, umjesto apartmana, na stranici će se prikazati sobe. Uveo sam CSS u Header.jsx kao i prije naredbom:

```
import "./header.css";
```

te naslove Apartments i Rooms koji zasad nemaju ikone, ali se mogu dodati umjesto komentara "ikona" ako bude potrebno. Također, apartmane sam označio kao trenutačno aktivne te sam im prema tome dodaо CSS stil.

```
const Header = () => {
  return (
    <div className="header">
      <div className="headerContainer">
        <div className="headerList">
          <div className="headerListItem active">
            {/* ikona*/}
            <span>Apartments</span>
          </div>
          <div className="headerListItem">
            {/* ikona*/}
            <span>Rooms</span>
          </div>
        </div>
      </div>
    );
};
```

```
};  
export default Header;
```

Sada imamo problem da zaglavlje ne liči na ništa. Potrebno je pozabaviti se CSS-om.

```
.header {  
    background: linear-gradient(135deg, #003580, #004a99);  
    color: white;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    padding: 40px 0;  
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);  
    position: relative;  
}
```

Ovdje je implementirana pozadinska boja te boja teksta. Zatim raspored elemenata je linijski na centru linije. Također definiran je razmak od 40 pixela lijevo i desno od elementa, sjena te relativna pozicija.

```
.headerContainer {  
    width: 100%;  
    max-width: 1024px;  
    text-align: center;  
    margin: 0px 0px 5px 0px;  
    padding: 0 20px;  
    box-sizing: border-box;  
}  
.headerContainer.listmode {  
    margin: 20px 0px 0px 0px;  
}
```

Širina je napravljena na isti način kao i kod navigacijske trake, a text je poravnat u sredinu te container ima donju marginu od 5 piksela.

```
.headerList {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    gap: 80px;  
}
```

Elementi apartmana i soba poravnati su u sredinu te su poredani slijedno po liniji s razmakom od 80 piksela.

```
.headerListItem {  
    display: flex;  
    flex-direction: column;  
    align-items: center;
```

```

    gap: 12px;
    cursor: pointer;
    transition: transform 0.3s ease, color 0.3s ease;
}
.headerListItem i {
    font-size: 2.5rem;
    margin-bottom: 50px;
}

.headerListItem span {
    font-size: 1.3rem;
    font-weight: 500;
}

.headerListItem:hover {
    transform: scale(1.1);
    color: #00c6ff;
}
.headerListItem.active {
    padding: 12px 24px;
    border-radius: 12px;
    background: rgba(255, 255, 255, 0.15);
    box-shadow: 0 0 15px rgba(0, 198, 255, 0.4);
    border: 1px solid rgba(255, 255, 255, 0.4);
    backdrop-filter: blur(8px);
    transition: all 0.3s ease;
}

```

Zasad nepostojeća ikona udaljena je od odgovarajućih opisa (Apartments i Rooms) 12 pixela, elementi su poravnati u sredinu te se prelaskom mišem prikazuje pointer. Napravljen je i CSS za ikone kojih trenutačno nema te veličina slova i debljina za span elemente Apartments i Rooms. Prelaskom preko njih, boja teksta se mijenja u plavu te se elementi povećavaju 1.1 puta. Apartments koji je active ima svoj padding sa svih strana. Granica mu je kružnog oblika sivo-bijele boje te ima sjenu i zamućena je.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprjeđenje izgleda CSS-a tako da ima svjetlo plavo boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru. AI mi je poboljšao moj css te nadogradio tako da vizualno ljepše izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-12 **Svrha:** Unapređenje CSS-a refraktor.

NASLOV, ODLOMAK TE GUMB

U Header.jsx nakon headerList-a dodan naslov i odlomak te gumb za prijavu ili registraciju:

```
<h1 className="headerTitle">Room-Rently | Find Your Perfect Apartment or Hol  
iday Stay</h1>  
<p className="headerDesc">Discover comfortable apartments and rooms for rent  
across stunning locations. Easy booking and the best prices, only with Room-R  
ently.</p>  
<button className="headerBTN">Sign in/Register</button>
```

popraćen CSS-om:

```
.headerDesc {  
    margin: 20px 0;  
    font-size: 1.2rem;  
    font-weight: 500;  
    color: #ffffff;  
    line-height: 1.5;  
    padding: 0 20px;  
}  
  
.headerBTN {  
    background-color: rgb(14, 95, 245);  
    color: white;  
    font-weight: 600;  
    border: none;  
    padding: 12px 25px;  
    border-radius: 8px;  
    cursor: pointer;  
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);  
    transition: all 0.3s ease;  
    margin: 30px;  
    max-width: 90%;  
    box-sizing: border-box;  
}  
  
.headerBTN:hover {  
    background-color: rgb(10, 70, 200);  
    transform: translateY(-2px);  
    box-shadow: 0px 6px 10px rgba(0, 0, 0, 0.15);  
}
```

Odlomak ima svoje gornje i donje margine od 20 piksela te su definirana veličina, debljina i boja fonta kao i visina retka te lijevi i desni padding. Gumb je malo svjetlijе boje od ostatka zaglavlja te su definirana obilježja fonta. Maknut je obrub, dodan razmak sa svih strana te zakriviljeni oblik granica. Također, implementirane su margine, pokazivač miša te sjena i promjena u malo tamniju boju te pomak prema gore prilikom hovera.

---

TRAŽILICA

---

## IMPLEMENTACIJA KOSTURA TE INPUT ELEMENTA ZA ODABIR IMENA APARTMANA ILI SOBE

Nakon što smo implementirali osnovne stavke, imamo problem implementacije tražilice. Dakle, nakon gumba imamo tražilicu:

```
<div className="headerSearch">
  <div className="headerSearchItem">
    {/* ikona*/}
    <input
      type="text"
      placeholder="Search by apartment name"
      className="headerSearchInput"
    ></input>
  </div>
  <div className="headerSearchItem">
    {/* ikona*/}
    <span className="headerSearchText">date to date</span>
  </div>
  <div className="headerSearchItem">
    {/* ikona*/}
    <span className="headerSearchText">2 adults 2 children 1 room</span>
  </div>
  <div className="headerSearchItem">
    {/* ikona*/}
    <button className="headerBTN">Search</button>
  </div>
</div>
```

Ona trenutačno ima fiksne tekstove no poslije ćemo to promijeniti. Sada je potrebno promijeniti izgled tražilici u CSS-u:

```
.headerSearch {
  height: 50px;
  background-color: white;
  border: 3px solid #febb02;
  display: flex;
  align-items: center;
  justify-content: space-around;
  padding: 10px 0px;
  border-radius: 5px;
  position: absolute;
  bottom: -30px;
  width: 100%;
  max-width: 1024px;
```

```
    box-sizing: border-box;  
}
```

Elementi tražilice imaju razmak oko sebe. Tražilica ima žuti rub koji je mrvicu zakriviljen te je tražilica pomaknuta 30 piksela na dolje u odnosu na zaglavlje i ima širinu istu kao i zaglavlje.

```
.headerSearchInput {  
    border: none;  
    outline: none;  
    width: 100%;  
    padding: 0 10px;  
    box-sizing: border-box;  
}  
.headerSearchText {  
    color: darkslategrey;  
    cursor: pointer;  
    padding: 0 10px;  
    white-space: nowrap;  
}
```

Polje za unos željenog imena apartmana odnosno sobe više nema obrub te je definirana boja fiksnog teksta i dodan pokazivač pri prelasku mišem preko njih.

---

#### PRIKAZ DATUMA TE KALENDARA

Pratio sam YouTube [tutorial](#) kako bih instalirao react-date-range paket koji sadrži kvalitetno sučelje za odabir raspona datuma koje će nam koristiti prilikom odabira termina u aplikaciji. 23 minute 46 sekundi je vrijeme u videozapisu kada se objasni instalacija paketa. Instalacija paketa dobije se naredbom u terminalu:

```
npm install react-date-range
```

S ove [stranice](#) iz odjeljka “DateRange” preuzeo sam kod koji se prikaze klikom na “VIEW CODE”. Kod sam mrvicu promijenio tako što sam umjesto state hook-a stavio date hook te sam uveo DateRange:

```
import { useState } from "react";  
import { DateRange } from "react-date-range";  
const [date, setDate] = useState([  
    {  
        startDate: new Date(),  
        endDate: null,  
        key: "selection",  
    },  
]);  
  
<DateRange  
    editableDateInputs={true}  
    onChange={(item) => setDate([item.selection])}
```

```
moveRangeOnFirstSelection={false}
ranges={date}
/>>;
```

Importi su normalno gdje i svi importi na vrhu datoteke, konstantu sam stavio u const Header prije returna, a **<Daterange.../>** nakon span elementa “date to date”. Ovaj kod služi za kalendar u kojem se može birati otkad do kad želimo rezervirati sobu. Nakon toga da bi sve radilo, bilo je potrebno preko terminala instalirati biblioteku date-fns za formatiranje datuma s pomoću naredbe:

```
npm i date-fns
```

No korištenjem naredbe naišao sam na problem korištenja react-date-range packagea. Naime, verzija paketa nije bila kompatibilna te je rješenje bilo unijeti ove dvije naredbe koje su prepravile problem kompatibilnosti verzija

```
npm uninstall date-fns
npm install date-fns@^3.0.0
```

Biblioteku sam uveo u kod s pomoću importa:

```
import "react-date-range/dist/styles.css"; // main css file
import "react-date-range/dist/theme/default.css"; // theme css file
import { format } from "date-fns"; // string format
```

Da bi to lijepo izgledao dodao sam klasu pod **<DateRange.../>**:

```
className = "date";
```

i u CSS-u:

```
.date {
  position: absolute;
  top: 50px;
  left: 210px;
  z-index: 2;
  transform: scale(0.9);
  transform-origin: top left;
}
```

Sada se biranje datuma nalazi ispod samog span elementa te ispred pozadine.

Dolazimo do problema dinamičkog prikazivanja datuma. Umjesto **endDate: null**, treba nam pravi datum **endDate: new Date()**. Statički tekst “date to date” izbacujemo te formatiramo dinamički prikaz datuma:

```
{
` ${format(date[0].startDate, "dd/MM/yyyy")} to ${format(
  date[0].endDate,
  "dd/MM/yyyy"
)}
```

```
)}`;  
}
```

koji su prije odabira jednaki. Nedostaje nam skrivanje kalendarja. Na početku kalendar treba biti sakriven što znači da moramo dodati hook u const Header:

```
const [opendate, setOpdate] = useState(false);
```

Zatim je potrebna izmjena koda:

```
{  
  opendate && (  
    <DateRange  
      editableDateInputs={true}  
      onChange={(item) => setDate([item.selection])}  
      moveRangeOnFirstSelection={false}  
      ranges={date}  
      className="date"  
    />  
  );  
}
```

Ovo znači da tek kada je opendate istinit, onda se otvara kalendar. Span elementu prikaza odabranih datuma dodan je event handler:

```
onClick={()=>setOpdate(!opendate)}
```

Dakle, kada kliknemo na span element, ako je kalendar sakriven onda se prikaže i obrnuto.

---

## BROJ OSOBA I SOBA

### FUNKCIONALNOST DINAMIČKOG ODABIRA BROJA OSOBA I SOBA

---

Sada treba i dinamički odabrati osobe te broj soba. Na početku, izbornik nije otvoren te se pretpostavlja da mora biti odabrana barem jedna odrasla osoba te jedna soba.

```
const [openOptions, setOpenOptions] = useState(false);  
const [options, setOptions] = useState({  
  adult: 1,  
  children: 0,  
  room: 1,  
});
```

Umjesto fiksног teksta, potrebno je formatirati dinamički:

```
{  
  `${options.adult} adult - ${options.children} children - ${options.room} ro
```

```
om`;  
}
```

te ćemo definirati isto ponašanje za prikazivanje i skrivanje izbornika klikom na span element kao i kod kalendara:

```
onClick={()=>setopenOptions(!openOptions)}
```

Sada napokon moramo dodati izbornik za biranje broja osoba te soba nakon span elementa:

```
{
  openOptions && (
    <div className="options">
      <div className="optionitem">
        <span className="optiontext">Adult</span>
        <div className="optioncounter">
          <button
            disabled={options.adult <= 1}
            className="optioncounterbutton"
            onClick={() => {
              handleoption("adult", "d");
            }}
          >
            -
          </button>
          <span className="optioncounternumber">{options.adult}</span>
        <button
          className="optioncounterbutton"
          onClick={() => {
            handleoption("adult", "i");
          }}
        >
          +
        </button>
      </div>
    </div>
    <div className="optionitem">
      <span className="optiontext">Children</span>
      <div className="optioncounter">
        <button
          disabled={options.children <= 0}
          className="optioncounterbutton"
          onClick={() => {
            handleoption("children", "d");
          }}
        >
```

```

        >
        -
      </button>
      <span className="optioncounternumber">{options.children}</span>
      <button
        className="optioncounterbutton"
        onClick={() => {
          handleoption("children", "i");
        }}
      >
        +
      </button>
    </div>
  </div>
  <div className="optionitem">
    <span className="optiontext">Room</span>
    <div className="optioncounter">
      <button
        disabled={options.room <= 1}
        className="optioncounterbutton"
        onClick={() => {
          handleoption("room", "d");
        }}
      >
        -
      </button>
      <span className="optioncounternumber">{options.room}</span>
      <button
        className="optioncounterbutton"
        onClick={() => {
          handleoption("room", "i");
        }}
      >
        +
      </button>
    </div>
  </div>
</div>
);
}

```

Imamo gumbove koji imaju disabled ponašanje za postavljanje broja manjeg od nužnog(niti jedna odrasla osoba te niti jedna soba te negativan broj djece). Plus i minus gumb imat će definirano ponašanje s pomoću

funkcije handleoption koju još nismo napravili. Logično sada implementiramo funkciju iznad returna od Headera:

```
const handleoption = (name, operation) => {
  setoptions((prev) => {
    return {
      ...prev,
      [name]: operation === "i" ? options[name] + 1 : options[name] - 1,
    };
  });
};
```

Kao što smo i u prijašnjem kodu napisali, ta funkcija prima dva parametra, ime onoga što želimo mijenjati te operaciju. Ona zatim kopira prijašnje stanje te ako je operacija “i” odnosno inkrementiranje onda dodaje jedan, a inače oduzima jedan.

#### IZGLED IZBORNIKA ZA ODABIR BROJA OSOBA I SOBA

---

Još nam preostaje implementirati izgled izbornika za biranje broja osoba te soba.

```
.options {
  z-index: 2;
  padding: 8px;
  position: absolute;
  top: 50px;
  background-color: white;
  color: gray;
  border: 2px solid rgb(255, 174, 0);
  border-radius: 5px;
  transform: scale(0.9);
  transform-origin: top left;
}
.optionitem {
  width: 180px;
  display: flex;
  justify-content: space-between;
  margin-bottom: 8px;
}
.optioncounter {
  display: flex;
  align-items: center;
  gap: 8px;
  font-size: 11px;
  color: black;
}
.optioncounterbutton {
```

```

width: 25px;
height: 25px;
border: 1px solid lightskyblue;
color: #00c6ff;
cursor: pointer;
background-color: white;
font-size: 0.8rem;
}
.optioncounterbutton:disabled {
  cursor: not-allowed;
}

```

Opcije će biti ispred ostatka stranice zbog z-komponente, isto kao i kalendar pomaknute prema dolje sa sivim tekstrom, bijelom pozadinom te žutim obrubom. Svaki item, odnosno npr. biranje broja odraslih osoba poredan je u jedan red širine 200 piksela s time da postoji razmak između span-ova(npr. Adult) te gumbova i prikaza odgovarajućih brojeva koji su zajedno u jednom div elementu. Njihov jednostavan prikaz određen je `.optioncounter{}` CSS-om. Gumbovi imaju određenu visinu, širinu, obrub te pointer za prelazak s mišem. Ako su gumbi disabled onda će miš pokazivati precrtanu crvenu kružnicu kao znak zabrane.

Komponenta je u potpunosti responzivna i optimizirana za tablete(max-width: 768px), mobitele(max-width: 480px) te vrlo male ekrane(max-width: 360px). Glavne prilagodbe su redukcija fontova i margina.**headerSearch** prelazi u vertikalni layout, date i options elementi pozicioniraju se kao donji “sheet” na ekranu, a gumb i tekst postaju manji te razmaci proporcionalno kraći.

```

@media screen and (max-width: 768px) {
  .headerList {
    gap: 40px;
  }

  .headerListItem i {
    font-size: 2rem;
    margin-bottom: 30px;
  }

  .headerListItem span {
    font-size: 1.1rem;
  }

  .headerBTN {
    padding: 10px 8px;
    margin: 0px;
    font-size: 0.9rem;
    max-width: 85%;
  }
}

```

```
.headerSearch {  
    height: auto;  
    flex-direction: column;  
    gap: 12px;  
    padding: 12px;  
    bottom: -40px;  
    position: relative;  
    margin: 0 20px 30px 20px;  
    width: calc(100% - 40px);  
}  
  
.headerSearchItem {  
    width: 100%;  
    padding: 8px;  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}  
  
.date {  
    position: fixed;  
    top: auto;  
    bottom: 0;  
    left: 50%;  
    transform: scale(0.6) translateX(-50%);  
    transform-origin: bottom center;  
}  
  
.options {  
    position: fixed;  
    top: auto;  
    bottom: 0;  
    left: 50%;  
    transform: scale(1.1) translateX(-50%);  
    transform-origin: bottom center;  
    max-height: 50vh;  
    overflow-y: auto;  
    padding: 10px;  
}  
  
.optionitem {  
    width: 160px;  
    font-size: 0.9rem;  
}
```

```
.optioncounterbutton {
  width: 22px;
  height: 22px;
}
.headerContainer h1 {
  font-size: 1.8rem;
  margin-bottom: 10px;
}

.headerDesc {
  font-size: 1rem;
  padding: 0 15px;
  margin: 15px 0;
}

.header {
  padding: 30px 0;
}
}

@media screen and (max-width: 480px) {
  .header {
    padding: 20px 0;
  }

  .headerContainer {
    padding: 0 15px;
  }

  .headerList {
    gap: 15px;
    flex-wrap: wrap;
    justify-content: space-around;
  }

  .headerListItem {
    min-width: 100px;
  }

  .headerListItem i {
    font-size: 1.5rem;
    margin-bottom: 15px;
  }
}
```

```
.headerListItem span {
  font-size: 0.8rem;
  text-align: center;
}

.headerBTN {
  padding: 8px 16px;
  margin: 0px;
  font-size: 0.8rem;
  max-width: 80%;
}

.headerSearch {
  margin: 0 20px 30px 20px;
  width: calc(100% - 30px);
  gap: 10px;
  padding: 10px;
}

.headerSearchItem {
  padding: 6px;
  font-size: 0.9rem;
}

.headerSearchInput {
  font-size: 0.9rem;
  padding: 0 8px;
}

.headerSearchText {
  font-size: 0.9rem;
  padding: 0 8px;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.date {
  transform: scale(0.7) translateX(-50%);
}

.options {
  transform: scale(1.1) translateX(-50%);
```

```
padding: 8px;
max-height: 60vh;
}

.optionitem {
  width: 140px;
  font-size: 0.8rem;
  margin-bottom: 6px;
}

.optioncounter {
  gap: 6px;
  font-size: 0.7rem;
}

.optioncounterbutton {
  width: 20px;
  height: 20px;
  font-size: 0.7rem;
}
.headerContainer h1 {
  font-size: 1.4rem;
  margin-bottom: 8px;
}

.headerDesc {
  font-size: 0.85rem;
  line-height: 1.4;
  padding: 0 10px;
  margin: 10px 0;
}
}

@media screen and (max-width: 360px) {
  .headerList {
    gap: 10px;
  }

  .headerListItem {
    min-width: 80px;
  }

  .headerListItem i {
    font-size: 1.3rem;
  }
}
```

```

        margin-bottom: 10px;
    }

.headerListItem span {
    font-size: 0.7rem;
}

.headerBTN {
    padding: 6px 12px;
    margin: 10px;
    font-size: 0.75rem;
    max-width: 75%;
}

.headerSearch {
    margin: 15px 10px 0 10px;
    width: calc(100% - 20px);
}
}

```

Promjene u responzivnosti služe za pozicioniranje elemenata poput kalendara i options prozora u donji dio ekranu kada se koristi mobilni prikaz. Cilj im je simulirati izgled tzv. "bottom sheet" panela, kakvi se često koriste u mobilnim aplikacijama npr. Booking, Google Maps, itd.

Dodajmo sada navbar i header na stranicu /hotels pomoću List.jsx-a:

```

import React from "react";
import "./list.css";
import Navbar from "../../components/navbar/navbar";
import Header from "../../components/header/Header";

const List = () => {
    return (
        <div>
            <Navbar></Navbar>
            <Header type="list"></Header>
        </div>
    );
}
export default List;

```

Ovdje smo također dodali prop type =“list” kako bi na ruti /hotels mogli vidjeti samo dio headera. Sada je u Header.jsx potrebno implementirati taj prop.

```
const Header = ({ type }) => {
  /* isti kod kao i prije*/
};
```

Dakle, prenijeli smo taj prop te je od h1 taga pa do zadnjeg itema potrebno zatvoriti strukturu s {} zagrada te unutar zagrada <> prije toga treba napisati da ako tip nije "list" onda će biti vidljiv ostatak headera, a inače su vidljivi samo span-ovi Apartments i Rooms zaokruženi s svojim div elementima. Sada kod izgleda ovako:

```
{
  type !== "list" && <>{/*od h1 pa sve do zadnjeg itema*/}</>;
}
```

Potrebno je još promijeniti da ako je tip "list" onda imamo dvije klase, a inače samo jednu zbog CSS-a:

```
<div className={type === "list" ? "headerContainer listmode": "headerContainer"}>
```

U CSS-u nadodamo kod za gornju marginu, kako bi se Apartments i Rooms prikazivali više dolje u usporedbi s main stranicom:

```
.headerContainer.listmode {
  margin: 20px 0px 0px 0px;
}
```

Na poslijetku, potrebno je za search dodati onclick event handler koji će prebaciti stranicu na adresu /hotels.

```
<button className="headerBTN" onClick={handleSearch}>
  Search
</button>
```

Sada nam još ostaje napraviti funkciju handleSearch s pomoću hook-a useNavigate uvezivanjem:

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

te dodatkom sljedećeg koda:

```
const [destination, setdestination] = useState("");
const navigate = useNavigate();
const handleSearch = () => {
  navigate("/hotels", { state: { destination, date, options } });
};
```

na zakomentiranome mjestu(pri vrhu prije returna):

```
const Header= ({type})=>{
  /* ovdje dodati*/
  return()
}
```

Klikom na gumb "Search" mijenja se adresa na /hotels te se šalje odabрано stanje. Ime apartmana ili sobe se mijenja s pomoću promjene u kodu za input:

```
<input  
  type="text"  
  placeholder="Search by apartment name"  
  className="headerSearchInput"  
  onChange={(e) => setdestination(e.target.value)}  
></input>
```

Nakon svake promjene inputa, destination(ime apartmana/sobe) se postavlja na upisanu vrijednost.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprjeđenje izgleda CSS-a tako da ima svijetlo plavo boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru primjerice klasa "headerBTN" ali zbog redundancije neću navoditi sve klase. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepeš izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-12 **Svrha:** Unapređenje CSS-a refraktor.

## KOMPONENTE KOJE SE NALAZE NA SREDINI GLAVNE STRANICE

### IZRADA FEATURED KOMPONENTE

Do sada smo napravili samo navigacijsku traku te zaglavlje. Potrebno je napraviti još mnogo komponenti koje će popuniti ostatak glavne stranice. Featured komponenta služi nam za kategoriziranje smještaja. U Home.jsx dodat ćemo nakon headera jedan div container:

```
<div className="homecontainer"></div>
```

te je u mapu components potrebno dodati novu komponentu(mapu) featured s odgovarajućim .css i .jsx datotekama. U homecontainer možemo dodati komponentu featured <Featured></Featured> te uvesti komponentu u Home.jsx naredbom **import Featured from**  
**"../../components/featured/featured";** i containeru dati CSS stil:

```
.homecontainer {  
  margin-top: 50px;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  gap: 30px;  
  padding: 0 10px;  
}
```

Sada se homecontainer nalazi 50 piksela ispod headera. Svaki njegov element je centriran, u svome redu s razmakom između svakog reda(elementa containera) od 30 piksela i paddingom s lijeve i desne strane od 10 piksela.

Preostaje nam izraditi komponentu Featured:

```
import "./featured.css";
const Featured = () => {
  return (
    <div className="featured">
      <div className="featureditem">
        
        <div className="featuredtitles">
          <h1>Sea view</h1>
          <h2>3 available apartments</h2>
        </div>
      </div>

      <div className="featureditem">
        
        <div className="featuredtitles">
          <h1>Village view</h1>
          <h2>123 available apartments</h2>
        </div>
      </div>

      <div className="featureditem">
        
        <div className="featuredtitles">
          <h1>Lake view</h1>
          <h2>167 available apartments</h2>
        </div>
      </div>
    );
};

export default Featured;
```

Radi jednostavnosti, za sada ćemo dodati 3 statična elementa sa svojim slikama te naslovima. One će označavati kategorije(npr. pogled na more) za lakše filtriranje soba i apartmana. Ovi elementi trenutačno ne pašu na stranici pa je bilo potrebno pozabaviti se s CSS-om:

```
.featured {
  width: 100%;
  max-width: 1024px;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  gap: 24px;
```

```
margin: 0 auto;
padding: 20px;
box-sizing: border-box;
z-index: 1;
}
```

Širina je ista kao i kod ostatka komponenti. Ako smo na manjim ekranima, elementi će se prebaciti u novi red po potrebi. Oni imaju razmak između od 24 piksela, padding sa svih strana od 20 piksela, automatske margeine s lijeve i desne strane te su u potpunosti s granicama unutar featured div-a te ispred ostalih komponenti stranice koji imaju z-index manji od 1 odnosno iza kalendara i izbornika za biranje broja osoba i broja soba.

```
.featureditem {
  position: relative;
  flex: 1 1 calc(33.333% - 24px);
  height: 260px;
  color: white;
  border-radius: 16px;
  overflow: hidden;
  cursor: pointer;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}

.featureditem:hover {
  transform: translateY(-6px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.25);
}
```

Svaki element ima relativnu poziciju, visinu od 260 piksela te mu je omogućeno povećanje ako ima mjesta, smanjenje ako nema mjesta te mu je početna širina trećina div-a featured umanjena za 24 piksela zbog razmaka od 24 piksela između elemenata. Tekst je bijele boje te svaki element ima zaobljene rubove i ako su elementi preveliči onda se taj overflow sakrije. Kursor je pokazivač što implicira da se elementi mogu kliknuti. Promjene pozicije i sjene traju 0.3 sekunde, a početna sjena je pomaknuta 4 piksela ispod elementa sa zamućenjem radijusa 12 piksela te navedenom bojom sjene. Prelaskom preko elementa mišem, element se pomakne prema gore za 6 piksela te sjena postaje veća i neprozirnija.

```
.featuredimg {
  width: 100%;
  height: 100%;
  object-fit: cover;
  transition: transform 0.5s ease, filter 0.5s ease;
  filter: brightness(70%);
}

.featureditem:hover .featuredimg {
```

```

    transform: scale(1.05);
    filter: brightness(60%);
}

.featureditem::after {
    content: "";
    position: absolute;
    inset: 0;
    background: linear-gradient(to top, rgba(0, 0, 0, 0.6), transparent 50%);
}

```

Slike su širine i visine 100 % featureditem containera te prekrivaju cijeli okvir s time da je višak odrezan. Prikazane su sa 70 % svjetline te su tranzicije glatke u trajanju od 0.5 sekundi. Kada se prelazi mišem preko featureditem-a, onda se slika unutar njega poveća za 5 % te se svjetlina smanji na 60%. Da bi tekst na slici bio čitljiviji, dodan je prazan element preko cijelog featureditem-a koji počinje od dna s 60 % neprozirnom crnom te na pola featureditem-a postaje potpuno nevidljiv.

```

.featuredtitles {
    position: absolute;
    bottom: 20px;
    left: 20px;
    z-index: 2;
}

.featuredtitles h1 {
    font-size: 22px;
    font-weight: 700;
    margin-bottom: 6px;
    color: #fff;
}

.featuredtitles h2 {
    font-size: 16px;
    font-weight: 400;
    color: #f1f1f1;
}

@media (max-width: 768px) {
    .featureditem {
        flex: 1 1 calc(50% - 24px);
        height: 220px;
    }
}

@media (max-width: 480px) {

```

```
.featureditem {
  flex: 1 1 100%;
  height: 200px;
}

.featuredtitles h1 {
  font-size: 18px;
}
}
```

Naslovi se nalaze 20 piksela iznad dna featureditem-a te 20 piksela desno od lijeve granice featureditem-a. Također, nalaze se ispred cijele featured komponente. Naslovi imaju određenu veličinu te debljinu i boju, a glavni naslov mjesto ima i donju marginu od 6 piksela. Na manjim ekranima, svaki featureditem će zauzimati pola odnosno cijelu širinu zaslona, pa će tako u istome redu biti dva odnosno jedan item. Osim toga, određena im je i visina te će na zaslonu širine manje od 480 piksela, glavni naslov koji označava kategoriju imati veličinu fonta 18 piksela.

---

## KOMPONENTA PROPERTYLIST

Da bi korisnik lakše filtrirao svoje preference, dodali smo komponentu propertylist koja omogućuje korisniku da odabere tip apartmana ili sobe(npr. dvokrevetna). Na glavnu stranicu(u Home.jsx), nakon featured komponente, dodao sam naslov za propertylist:

```
<h1 className="hometitle">Browse apartments by capacity</h1>
```

te u CSS stil:

```
.hometitle {
  width: 100%;
  max-width: 1024px;
  font-size: 20px;
  box-sizing: border-box;
  text-align: center;
}

@media (max-width: 600px) {
  .hometitle {
    font-size: 16px;
  }
}
```

kako bi širina odgovarala širini ostalih komponenti te da bi se naslov prikazao na sredini zaslona. Za manje ekrane, smanjena je i veličina fonta. Sada dodajmo mapu za komponentu propertylist te odgovarajuće .css i .jsx datoteke. Napravimo HTML strukturu istu kao i za featured komponentu:

```
import "./propertylist.css";
```

```
const Propertylist = () => {
  return (
    <div className="pList">
      <div className="plistItem">
        
        <div className="plisttitle">
          <h1>1 Bed</h1>
          <h2>69 Apartments</h2>
        </div>
      </div>

      <div className="plistItem">
        
        <div className="plisttitle">
          <h1>2 Beds</h1>
          <h2>21 Apartments</h2>
        </div>
      </div>

      <div className="plistItem">
        
        <div className="plisttitle">
          <h1>3 Beds</h1>
          <h2>67 Apartments</h2>
        </div>
      </div>

      <div className="plistItem">
        
        <div className="plisttitle">
          <h1>4 Beds</h1>
          <h2>420 Apartments</h2>
        </div>
      </div>

      <div className="plistItem">
        
        <div className="plisttitle">
          <h1>5 Beds</h1>
          <h2>41 Apartments</h2>
        </div>
      </div>
    </div>
  );
};
```

```
};

export default Propertylist;
```

te dodajmo naš propertylist na glavnu stranicu nakon njegovog naslova naredbom **<Propertylist></Propertylist>**. Sada je potrebno komponenti dati svoj stil:

```
.plist {
    width: 100%;
    max-width: 1024px;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    gap: 24px;
    margin: 0 auto;
    padding: 20px;
    box-sizing: border-box;
}

.plistItem {
    flex: 1 1 calc(33.333% - 24px);
    border-radius: 14px;
    overflow: hidden;
    cursor: pointer;
    background-color: #fff;
    box-shadow: 0 3px 10px rgba(0, 0, 0, 0.08);
    transition: transform 0.25s ease, box-shadow 0.25s ease;
}

.plistItem:hover {
    transform: translateY(-6px);
    box-shadow: 0 6px 18px rgba(0, 0, 0, 0.15);
}

.plistimg {
    width: 100%;
    height: 180px;
    object-fit: cover;
    transition: transform 0.4s ease;
}

.plistItem:hover .plistimg {
    transform: scale(1.05);
}
```

Glavni container kao i njegovi elementi(itemi) i slike imaju standardni stil koji smo koristili i kod featured komponente gdje je isti i objašnjen. Razlike u odnosu na njega su to da ovdje nemamo definiranu visinu itema, z-index containera i izmjenu svjetline slike, a imamo definiranu visinu slike od 180 piksela. Ostatak CSS stila:

```
.plisttitle {  
    padding: 14px 16px;  
    text-align: left;  
    background-color: #fff;  
}  
  
.plisttitle > h1 {  
    font-size: 20px;  
    font-weight: 700;  
    color: #222;  
    margin-bottom: 6px;  
}  
  
.plisttitle > h2 {  
    font-size: 16px;  
    font-weight: 400;  
    color: #666;  
}  
  
@media (max-width: 768px) {  
    .plistItem {  
        flex: 1 1 calc(50% - 24px);  
    }  
}  
  
@media (max-width: 480px) {  
    .plistItem {  
        flex: 1 1 100%;  
    }  
  
.plistimg {  
    height: 160px;  
}  
}
```

Budući da je visina fiksna, ovdje naslovi neće biti preko nego ispod slike. Naslovi imaju padding sa svih strana, poravnat tekst u lijevo te bijelu pozadinu. Veći naslov ima određen veći i deblji font tamnije boje te je donjom marginom odvojen od manjeg naslova. Za manje ekrane određena su dva elementa liste u svakome redu, a za najmanje ekrane imamo samo jedan element u redu te je visina slike umanjena za 20 piksela.

Dio CSS koda u ovom repozitoriju je također unaprijeden s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprjeđenje izgleda CSS-a tako da ima bolji izgled, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru primjerice klasa "plistItem" ali zbog redundancije neću navoditi sve klase. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepeš izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-12 **Svrha:** Unapređenje CSS-a refraktor, poboljšanje responzivnosti.

---

## KOMPONENTA FEATUREDPROPERTIES

Još nam za lakši odabir nedostaje komponenta za najpopularnije apartmane, odnosno sobe. Na glavnu stranicu (Home.jsx) potrebno je dodati naslov za featuredproperties te komponentu featuredproperties koju još nismo napravili:

```
import Featuredproperties from "../../components/featuredproperties/featuredproperties";  
  
/* Nakon propetylist-a */  
<h1 className="hometitle">Apartments guests love</h1>  
<Featuredproperties></Featuredproperties>
```

U mapu komponenti, potrebno je dodati featuredproperties mapu te odgovarajuće .css i .jsx datoteke. Sada je potrebno napisati strukturu featuredproperties-a te čemo za sad napraviti navigaciju za prvi element koja će klikom na njega voditi na stranicu /hotels/1 na početak stranice:

```
import "./featuredproperties.css";  
import { useNavigate } from "react-router-dom";  
const Featuredproperties = () => {  
  const navigate = useNavigate();  
  const handleapartman = () => {  
    navigate("/hotels/1");  
    window.scrollTo(0, 0);  
  };  
  return (  
    <div className="fp">  
      <div className="fpitem" onClick={handleapartman}>  
          
        <span className="fpname">Apartments Ani</span>  
        <span className="fploc">Paviljon 3</span>  
        <span className="fpprice">Starting from 120$</span>  
        <div className="fprating">  
          <button>9.9</button>  
          <span>Excellent</span>  
        </div>  
      </div>  
    </div>
```

```

<div className="fpitem">
  
  <span className="fpname">Apartments Miku</span>
  <span className="fploc">Paviljon 2</span>
  <span className="fpprice">Starting from 41$</span>
  <div className="fprating">
    <button>9.1</button>
    <span>Excellent</span>
  </div>
</div>

<div className="fpitem">
  
  <span className="fpname">Apartments Krapić</span>
  <span className="fploc">Paviljon 1</span>
  <span className="fpprice">Starting from 12000$</span>
  <div className="fprating">
    <button>10.0</button>
    <span>Excellent</span>
  </div>
</div>

<div className="fpitem">
  
  <span className="fpname">Apartmani Jakov</span>
  <span className="fploc">Paviljon 4</span>
  <span className="fpprice">Starting from 150$</span>
  <div className="fprating">
    <button>9.5</button>
    <span>Excellent</span>
  </div>
</div>
</div>
);
};

export default Featuredproperties;

```

Svaki element(popularni apartman/soba) imat će svoju sliku, ime, lokaciju(broj paviljona i sl.), cijenu te ocjenu korisnika s gumbom i opisom ocjene. Sada je potrebno ovoj komponenti dodati CSS stil:

```

.fp {
  width: 100%;
  max-width: 1024px;
  display: flex;

```

```

flex-wrap: wrap;
justify-content: space-between;
gap: 24px;
margin: 0 auto;
padding: 20px;
box-sizing: border-box;
}

.fpitem {
  flex: 1 1 calc(25% - 24px);
  display: flex;
  flex-direction: column;
  background-color: #fff;
  border-radius: 12px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.08);
  transition: transform 0.2s ease, box-shadow 0.2s ease;
  cursor: pointer;
  overflow: hidden;
}

.fpitem:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 18px rgba(0, 0, 0, 0.15);
}

.fpimg {
  width: 100%;
  height: 200px;
  object-fit: cover;
}

```

Isto kao i za propertylist, no sada ćemo umjesto tri elementa po redu imati četiri te će visina slike biti 20 piksela veća.

```

.fpname {
  padding: 10px;
  font-weight: 700;
  font-size: 18px;
  color: #222;
}

.fploc {
  padding: 10px;
  font-weight: 400;
  font-size: 15px;

```

```

    color: #666;
}

.fpprice {
    padding: 10px;
    font-weight: 500;
    font-size: 16px;
    color: #111;
}

.fprating {
    padding: 20px;
    display: flex;
    align-items: center;
    margin-top: 8px;
}

```

Za ime apartmana/sobe, lokaciju te cijenu definirani su padding, veličina fonta te debljina i boja. Za ocjenu definiran je padding i gornja margina te su gumbu i span-u centrirane visine.

```

.fprating > button {
    background-color: #001f54;
    color: white;
    border: none;
    border-radius: 6px;
    padding: 4px 8px;
    margin-right: 8px;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.2s ease;
}

.fprating > button:hover {
    background-color: #003b8e;
}

.fprating > span {
    font-size: 16px;
    color: #333;
}

@media (max-width: 768px) {
    .fpitem {
        flex: 1 1 calc(50% - 24px);
    }
}

```

```
}
```

```
@media (max-width: 480px) {
```

```
    .fpitem {
```

```
        flex: 1 1 100%;
```

```
    }
```

```
}
```

Gumbovima je definirana pozadinska boja, boja teksta, zakriviljena nevidljiva granica, padding sa svih strana te desna margina. Tekst je podebljan te se pokazivačem miša implicira da se gumb može stisnuti. Kada stavimo miš na gumb, boja gumba također lagano prelazi u svjetlu plavu boju. Također, definirani su veličina fonta te boja za span element(ocjenu). Za manje ekrane, prikazuje se dva odnosno jedan element u svakome redu.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprijeđenje izgleda CSS-a tako da ima tamnije plavo boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru primjerice klasa "fpitem" ali zbog redundancije neću navoditi sve klase. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepešte izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-12 **Svrha:** Unapređenje CSS-a refraktor, poboljšanje u responzivnosti.

## KOMPONENTE PODNOŽJA STRANICE

### KOMPONENTA MAILLIST

Za one koji se žele pretplatiti kako bi dobivali obavijesti o najboljim ponudama dodati ćemo komponentu maillist gdje se može upisati korisnikov e-mail te se pretplatiti klikom na gumb subscribe. Potrebno je na glavnu stranicu u datoteku Home.jsx nakon featuredproperties komponente dodati komponentu maillist:

```
import Maillist from "../../components/maillist/maillist";
```

```
/* Nakon <Featuredproperties></Featuredproperties>*/
```

```
<Maillist></Maillist>;
```

Sada je potrebno u mapu components dodati mapu maillist s odgovarajućim .css i .jsx datotekama. Struktura komponente izgleda vrlo jednostavno:

```
import "./maillist.css";
const Maillist = () => {
    return (
        <div className="mail">
            <h1 className="mailtitle">Save time, save money!</h1>
            <span className="maildesc">
                Sign up and we'll send the best deals to you
            </span>
            <div className="mailinputcontainer">
```

```

        <input type="text" placeholder="Your email" />
        <button>Subscribe</button>
    </div>
</div>
);
};

export default Maillist;

```

Komponenta ima svoj container, naslov, opis(span element) te container gdje je moguće upisati e-mail s pomoću input elementa i gumb za pretplatu. Dodajmo maillist komponenti CSS stil:

```

.mail {
    width: 100%;
    margin-top: 60px;
    background: linear-gradient(135deg, #001a66, #0033cc);
    color: #fff;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 24px;
    padding: 60px 20px;
    text-align: center;
    border-radius: 16px;
    box-shadow: 0 6px 20px rgba(0, 0, 50, 0.3);
}

.mail h1 {
    font-size: 28px;
    font-weight: 700;
    margin: 0;
}

.mail span {
    font-size: 16px;
    font-weight: 300;
    max-width: 500px;
    color: #e0e0e0;
}

```

Container će biti širine cijele stranice, malo odmaknut od featuredproperties te će mu pozadina biti prijelaz iz tamne plave u svijetlu plavu dijagonalno od gornjeg lijevog prema donjem desnom kutu. Boja teksta je bijela, elementi su poravnati u sredinu po visini zaslona te je svaki element u svojem redu s razmakom od 24 piksela između njih. Dodatno, sadržaj containera ima svoj padding u odnosu na granicu containera te je tekst centriran. Rubovi containera su zaobljeni, a tamno plava sjena sa 30 % neprozirnosti mu je 6 piksela ispod te joj je radijus zamudjenosti 20 piksela. Naslovu su određeni veličina fonta i njegova debljina te su mu margine 0

piksela. Span element ima definiranu veličinu, debljinu te boju teksta, kao i maksimalnu širinu elementa od 500 piksela.

```
.mailinputcontainer {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    flex-wrap: wrap;  
    gap: 12px;  
    width: 100%;  
    max-width: 500px;  
}  
  
.mailinputcontainer > input {  
    flex: 1;  
    min-width: 220px;  
    height: 48px;  
    padding: 0 16px;  
    border: none;  
    border-radius: 8px;  
    outline: none;  
    font-size: 16px;  
    color: #333;  
    transition: box-shadow 0.2s ease;  
}  
  
.mailinputcontainer > input:focus {  
    box-shadow: 0 0 0 3px rgba(255, 255, 255, 0.4);  
}
```

Elementi containera za pretplatu su centrirani u jednome redu te je razmak između njih 12 piksela. Container je širok kao i cijela stranica s ograničenjem za ekrane veće od 500 piksela na statičku širinu od 500 piksela. Input element pokriva koliko može mjesta(najmanje 220 piksela širina) te mu je visina 48 piksela. Placeholder tekst mu je sive boje veličine 16 piksela te ima lijevi i desni padding od 16 piksela. Obruba nema te su kutevi zakriviljeni. Na fokus(klik miša) input element dobije bijelu sjenu neprozirnosti 40 % s radijusom širenja izvan ruba elementa od 3 piksela.

```
.mailinputcontainer > button {  
    height: 48px;  
    padding: 0 24px;  
    background-color: #0052ff;  
    color: #fff;  
    font-weight: 600;  
    border: none;  
    border-radius: 8px;
```

```

    cursor: pointer;
    transition: background-color 0.25s ease, transform 0.2s ease;
}

.mailinputcontainer > button:hover {
    background-color: #003fcc;
    transform: translateY(-2px);
}

.mailinputcontainer > button:active {
    transform: scale(0.98);
}

```

Gumb je iste visine kao i input element te mu je sadržaj pomaknut od granica elementa s lijeve i desne strane za 24 piksela. Boja pozadine je plava, a boja teksta je bijela debljine 600. Obruba nema, a rubovi su zaobljeni. Pokazivač miša koji se pojavi prelaskom preko gumba implicira da se gumb može kliknuti. Također, kada stavimo pokazivač miša ispred gumba, gumb se pomakne prema gore za 2 piksela te mu boja postane tamnija nijansa plave. Prilikom klika na gumb, on se smanji na 98 % svoje vrednosti.

```

@media (max-width: 480px) {
    .mail {
        padding: 40px 16px;
    }

    .mail h1 {
        font-size: 22px;
    }

    .mail span {
        font-size: 14px;
    }

    .mailinputcontainer > button {
        width: 100%;
    }
}

```

Za manje ekrana(480 piksela i manje) umanjeni su padding elemenata glavnog containera te veličina fonta naslova i span elementa. Također, gumb je prebačen u novi red jer mu je širina ista širini ekrana, kako bi bilo više mesta za unos korisnikovog e-maila.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprijeđenje izgleda CSS-a tako da ima plavu boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru primjerice klasa "mailinputcontainer" ali zbog redundancije neću navoditi sve klase. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepše izgleda naspram mog početnog

koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-13

**Svrha:** Unapređenje CSS-a refraktor, poboljšanje u responzivnosti.

---

## KOMPONENTA FOOTER

Na posljetku, potrebno je dodati komponentu podnožja stranice. U Home.jsx (na glavnu stranicu) uvesti ćemo komponentu footer koju još nismo napravili:

```
import Footer from "../../components/footer/footer";
```

te ćemo ju smjestiti ispod komponente maillist:

```
/* <Maillist></Maillist>*/  
<Footer></Footer>
```

Sada je potrebno napraviti komponentu podnožja kao zadnji element glavne stranice. U mapu components, potrebno je dodati novu mapu footer te u nju smjestiti footer.jsx i footer.css datoteke. Komponenta podnožja sastojat će se od neporedanih lista u svojem posebnom containeru te će se taj container i div element koji prikazuje copyright nalaziti u containeru podnožja:

```
import "./footer.css";  
const Footer = () => {  
  return (  
    <div className="footer">  
      <div className="flists">  
        <ul className="flist">  
          <li className="flistitem">Countries</li>  
          <li className="flistitem">Regions</li>  
          <li className="flistitem">Cities</li>  
          <li className="flistitem">Districts</li>  
          <li className="flistitem">Hotels</li>  
        </ul>  
        <ul className="flist">  
          <li className="flistitem">Countries</li>  
          <li className="flistitem">Regions</li>  
          <li className="flistitem">Cities</li>  
          <li className="flistitem">Districts</li>  
          <li className="flistitem">Hotels</li>  
        </ul>  
        <ul className="flist">  
          <li className="flistitem">Countries</li>  
          <li className="flistitem">Regions</li>  
          <li className="flistitem">Cities</li>  
          <li className="flistitem">Districts</li>  
          <li className="flistitem">Hotels</li>  
        </ul>  
      </div>  
    </div>  
  );  
};
```

```

        </ul>
        <ul className="flist">
            <li className="flistitem">Countries</li>
            <li className="flistitem">Regions</li>
            <li className="flistitem">Cities</li>
            <li className="flistitem">Districts</li>
            <li className="flistitem">Hotels</li>
        </ul>
        <ul className="flist">
            <li className="flistitem">Countries</li>
            <li className="flistitem">Regions</li>
            <li className="flistitem">Cities</li>
            <li className="flistitem">Districts</li>
            <li className="flistitem">Hotels</li>
        </ul>
    </div>
    <div className="ftext">Copyright © 2025 Room-Rently</div>
</div>
);
};

export default Footer;

```

Podnožju je potrebno dodati CSS stil:

```

.footer {
    width: 100%;
    max-width: 1024px;
    font-size: 15px;
}

.flists {
    width: 100%;
    display: flex;
    justify-content: space-between;
    margin-bottom: 50px;
}

.flist {
    list-style: none;
    padding: 0;
}

.flistitem {
    margin-bottom: 10px;
    color: rgb(23, 23, 104);
    cursor: pointer;
}

```

Podnožje će biti širine cijelog zaslona s ograničenjem na 1024 piksela, što znači da će za ekrane veće od 1024 piksela, širina podnožja ostati (biti ograničena) na 1024 piksela. Veličina fonta postavljena je na 15 piksela. Container koji sadrži neporedane liste će zauzimati cijelu širinu containera podnožja te će neporedane liste unutar njega imati razmak među sobom. Za sadržaj toga containera, još je definirana i donja margina od 50 piksela. Neporedane liste nemaju nikakve bullete ispred teksta, kao ni padding između svakog elementa liste. Svaki element neporedanih lista ima postavljenu donju marginu od 10 piksela te je boja teksta plava i kursor miša postaje pointer kada ga držimo ispred elementa liste.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponenti CSS-a je dobiven upitom "Potrebno mi je poboljšanje i unaprjeđenje izgleda CSS-a tako da ima plavo boju, prilažem ti html klase koje je potrebno koristiti" te su naredane klase navedene gore u kodu kao i na prethodnom primjeru primjerce klase "listitem" ali zbog redundancije neću navoditi sve klase. AI mi je poboljšao moj CSS te nadogradio tako da vizualno ljepše izgleda naspram mog početnog koda. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-10-13

**Svrha:** Unapređenje CSS-a refraktor.

## STRANICA LIST

Potrebno je napraviti search izbornik na stranici list gdje će se prenositi informacije s glavne stranice, ali će biti moguće i mijenjati prenesene podatke. Do sada smo stavili samo navigacijsku traku te zaglavje. Sada moramo na isti način kao i u headeru prikazati sve podatke te omogućiti njihovu promjenu. List.jsx će sada izgledati ovako:

```
import React from "react";
import "./list.css";
import Navbar from "../../components/navbar/navbar";
import Header from "../../components/header/Header";
import { useLocation } from "react-router-dom";
import { useState } from "react";
import { format } from "date-fns";
import { DateRange } from "react-date-range";
import Searchitem from "../../components/searchitem/searchitem";
const List = () => {
  const location = useLocation();
  const [destination, setdestination] = useState(location.state.destination);
  const [date, setdate] = useState(location.state.date);
  const [opendate, setopendate] = useState(false);
  const [options, setoptions] = useState(location.state.options);
  return (
    <div>
      <Navbar></Navbar>
      <Header type="list"></Header>
      <div className="listcontainer">
        <div className="listwrapper">
          <div className="listsearch">
```

```
<h1 className="lstable">Search</h1>
<div className="lsitem">
  <label>Apartment name</label>
  <input placeholder={destination} type="text" />
</div>
<div className="lsitem">
  <label>Check-in date</label>
  <span onClick={() => setopendate(!opendate)}>` ${format(
    date[0].startDate,
    "dd/MM/yyyy"
  )} to ${format(date[0].endDate, "dd/MM/yyyy")}`</span>
  {opendate && (
    <DateRange
      onChange={(item) => setdate([item.selection])}
      minDate={new Date()}
      ranges={date}
    />
  )}
</div>
<div className="lsitem">
  <label>Options</label>
  <div className="lsoptions">
    <div className="lsoptionitem">
      <span className="lsoptiontext">
        Min price <small>per night</small>
      </span>
      <input type="number" min={0} className="lsoptioninput" />
    </div>
    <div className="lsoptionitem">
      <span className="lsoptiontext">
        Max price <small>per night</small>
      </span>
      <input type="number" min={0} className="lsoptioninput" />
    </div>
    <div className="lsoptionitem">
      <span className="lsoptiontext">Adult </span>
      <input
        type="number"
        min={1}
        className="lsoptioninput"
        placeholder={options.adult}
      />
    </div>
    <div className="lsoptionitem">
```

```

        <span className="lsoptiontext">Children </span>
        <input
            type="number"
            className="lsoptioninput"
            min={0}
            placeholder={options.children}
        />
    </div>
    <div className="lsoptionitem">
        <span className="lsoptiontext">Room </span>
        <input
            type="number"
            className="lsoptioninput"
            min={1}
            placeholder={options.room}
        />
    </div>
</div>
<button>Search</button>
</div>
<div className="listresult">
    <Searchitem />
    <Searchitem />
</div>
</div>
</div>
);
};

export default List;

```

Dakle, nakon headera dodali smo container i wrapper te sam listsearch. Naslov je normalno search te imamo lsitem-e koji će prikazivati naziv sobe/apartmana, datum i odabrane opcije. Da bismo dobili podatke s glavne stranice, koristimo **useLocation()** hook koji u sebi ima state koji smo poslali s glavne stranice. Naziv sobe odnosno apartmana ima placeholder destination koji dobiva s glavne stranice s pomoću

**location.state.destination** te nam je također potreban import **useState()** hook-a. Datum dobivamo na isti način te ga dinamički prikazujemo isto kao i u headeru. Naravno, klikom na input, pojavi se

kalendar gdje se može promijeniti datum kao i kod headera te se ponovnim klikom zatvori. Opcijama ćemo dodati odabir minimalne i maksimalne cijene po noćenju te cijene neće moći ići ispod nule. Broj odraslih, djece i soba dobivamo s glavne stranice, a za promjenu imamo iste ograde kao i na glavnoj stranici. Dakle, barem jedna odrasla osoba, jedna soba te broj djece ne smije biti manji od nule. Na dnu imat ćemo gumb search za pretraživanje. Osim toga, na stranici je potrebno prikazati i rezultate pretraživanja pa ćemo za sada staviti 8 searchitem komponenta. Searchitem komponenta još nije napravljena, no prvo se trebamo pozabaviti CSS-om pa ju možemo trenutno zakomentirati:

```
.listcontainer {  
    display: flex;  
    justify-content: center;  
    margin-top: 30px;  
    padding: 0 20px;  
    box-sizing: border-box;  
    width: 100%;  
}  
  
.listwrapper {  
    width: 100%;  
    max-width: 1100px;  
    display: flex;  
    gap: 24px;  
    align-items: flex-start;  
}
```

Containeru će sadržaj biti flex te centriran s gornjom marginom od 30 piksela odnosno bit će pomaknut od headera. Padding sadržaja od granica će biti 20 piksela od lijeve i desne strane te će container biti iste širine kao i stranica. Također, wrapper će uzimati cijelu širinu containera s ograničenjem od 1100 piksela te će razmak elemenata wrappera iznosići 24 piksela. Elementi će započinjati na vrhu wrappera te će zauzimati samo onoliko visine koliko im je potrebno.

```
.listsearch {  
    flex: 1;  
    background: linear-gradient(135deg, #f3c247, #f9d76a);  
    padding: 20px;  
    border-radius: 16px;  
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);  
    color: #333;  
    max-width: 100%;  
    box-sizing: border-box;  
}  
  
.lstitle {  
    font-size: clamp(18px, 2vw, 20px);  
    font-weight: 700;
```

```
    color: #444;
    margin-bottom: 16px;
    word-wrap: break-word;
}

Search će imati žutu pozadinu s dijagonalnim prijelazom iz tamnije u svjetlu od gornjeg desnog kuta. Sadržaj će biti pomaknut za 20 piksela u odnosu na granice te će kutevi biti zakriviljeni. Ima crnu sjenu i crnu boju teksta sivo-crnu te maksimalnu širinu od 100 % containera. Naslov će imati minimalnu veličinu od 18 piksela i maksimalnu od 20 piksela s idealnom veličinom od 2 % širine stranice. Za njega su također definirani debljina fonta, boja, gornja margina te ponašanje prelamanja naslova kada nema mjesta na stranici.

.lsitem {
    display: flex;
    flex-direction: column;
    gap: 6px;
    margin-bottom: 16px;
    width: 100%;
}

.lsitem > label {
    font-size: clamp(13px, 1.5vw, 14px);
    font-weight: 500;
    color: #222;
}

.lsitem > input,
.lsitem > span {
    height: 38px;
    border: none;
    border-radius: 6px;
    padding: 8px 10px;
    font-size: clamp(13px, 1.5vw, 14px);
    box-sizing: border-box;
    width: 100%;
    max-width: 100%;
}

.lsitem > input {
    background-color: #fff;
    outline: none;
    transition: box-shadow 0.2s ease;
}

.lsitem > input:focus {
    box-shadow: 0 0 0 3px rgba(0, 0, 0, 0.2);
```

```

}

.lsitem > span {
  background-color: #fff;
  display: flex;
  align-items: center;
  cursor: pointer;
  color: #555;
  transition: background-color 0.2s ease;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}

.lsitem > span:hover {
  background-color: #f0f0f0;
}

```

Elementi liste bit će svaki u svome redu s razmakom od 6 piksela te donjom marginom od 16 piksela i zauzimat će 100 % širine containera. Za labelu, definirana je responzivnost veličine fonta, debljina fonta te boja. Za input i span elemente imena sobe i datuma, definirana je visina, zakriviljeni rubovi, padding sa svih strana, responzivnost veličine fonta te širina u odnosu na container. Za input je dodatno definirana boja teksta te obrub i sjena kada kliknemo na input. Osim toga, span elementu dodana je boja pozadine te je visina teksta centrirana. Kursor je pointer dok ga držimo iznad spana te je boja teksta siva. Ako je element prevelik, onda se višak sakrije te će višak teksta biti prikazan s tri točke, a sav tekst će biti prikazan u istome redu. Na hover, boja span elementa postaje malo tamnija.

```

.lsoptions {
  padding: 10px 0;
  border-top: 1px solid rgba(0, 0, 0, 0.1);
  margin-top: 10px;
  width: 100%;
}

.lsoptionitem {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
  font-size: clamp(13px, 1.5vw, 14px);
  color: #444;
  width: 100%;
  gap: 10px;
}

```

Opcije imaju definirani gornji i donji padding od 10 piksela te gornji obrub crne boje s neprozirnosti 10 %. Također imaju gornju marginu te zauzimaju cijelu širinu containera. Elementi lsoptionitema imaju razmak među sobom te su centrirani u odnosu na visinu. Osim toga imaju donju marginu, responzivnu veličinu fonta, boju teksta te zauzimaju cijelu širinu containera i razmaknuti su od drugih itema za 10 piksela.

```
.lsoptioninput {  
    width: 60px;  
    min-width: 50px;  
    padding: 5px;  
    border-radius: 6px;  
    border: 1px solid #ccc;  
    outline: none;  
    text-align: center;  
    transition: border-color 0.2s ease;  
    font-size: clamp(13px, 1.5vw, 14px);  
    box-sizing: border-box;  
}  
  
.lsoptioninput:focus {  
    border-color: #0099cc;  
}
```

Input elementi koji služe za promjenu brojeva u opcijama imaju definiranu širinu, padding, zakrivljene rubove te bijelu granicu. Nemaju obrub te im je tekst centriran. Veličina fonta je responzivna, a klikom na input element, granica dobiva plavu boju.

```
.listsearch > button {  
    padding: 12px;  
    background-color: #008b8b;  
    color: white;  
    font-weight: 600;  
    border: none;  
    width: 100%;  
    cursor: pointer;  
    border-radius: 8px;  
    transition: background-color 0.25s ease, transform 0.2s ease;  
    margin-top: 10px;  
    font-size: clamp(14px, 1.8vw, 16px);  
    box-sizing: border-box;  
}  
  
.listsearch > button:hover {  
    background-color: #007575;  
    transform: translateY(-2px);  
}
```

Gumb ima svoj padding, boju pozadine i teksta, debljinu fonta te zauzima cijelu širinu containera. Rubovi gumba su zakriviljeni, a kursor kada ga držimo ispred gumba postaje pointer. Osim toga, definirani su gornja margina te responzivna veličina fonta. U veličinu elementa uračunata je i granica, a kada hover-amo gumb mišem, boja pozadine se promjeni u tamniju te se gumb pomakne za 2 piksela prema gore.

```
.listresult {  
    flex: 3;  
    display: flex;  
    flex-direction: column;  
    gap: 20px;  
    min-width: 0;  
}
```

Rezultati pretraživanja zauzimaju 3/4 širine containera, dok listsearch zauzima ostatak. Elementi su mu poredani u istome stupcu, svaki u svojem redu s razmakom od 20 piksela te mogućnosti smanjenja širine.

```
@media (max-width: 900px) {  
    .listwrapper {  
        flex-direction: column;  
        gap: 20px;  
    }  
  
    .listsearch {  
        width: 100%;  
        max-width: none;  
    }  
  
    .listresult {  
        width: 100%;  
    }  
}
```

Za ekrane širine manje ili jednake 900 piksela, search će zauzimati 100 % širine ekrana kao i rezultati te će se rezultati nalaziti ispod njega razmaknuti za 20 piksela.

```
@media (max-width: 768px) {  
    .listcontainer {  
        padding: 0 15px;  
        margin-top: 20px;  
    }  
  
    .listwrapper {  
        gap: 16px;  
    }  
  
    .listsearch {
```

```

    padding: 16px;
    border-radius: 12px;
}

.lsoptionitem {
    flex-wrap: wrap;
}

.lsoptioninput {
    width: 100%;
    max-width: 80px;
}
}

```

Za ekrane širine manje ili jednake 768 piksela, container će imati manji lijevi i desni padding te gornju marginu. Također, razmak između searcha i rezultata će biti manji, kao i padding i zakrivljenje rubova kod searcha. Input element će sada zauzimati širine koliko može, s maksimalnom širinom od 80 piksela.

```

@media (max-width: 480px) {
.listcontainer {
    padding: 0 10px;
    margin-top: 15px;
}

.listsearch {
    padding: 12px;
    border-radius: 10px;
}

.lsitem {
    margin-bottom: 12px;
}

.lsitem > input,
.lsitem > span {
    height: 36px;
    padding: 6px 8px;
}

.listsearch > button {
    padding: 10px;
}

.lsoptions {
    padding: 8px 0;
}

```

```

        }
    }

Kao i kod ekrana širine manje ili jednake 768 piksela, kod ekrana širine manje ili jednake 480 piksela smanjit
će se još više iste stvari kod listcontainer-a te listsearch-a. Smanjit će se donja margina lsitem-a te visina
input i span elementa unutar njih. Padding će se također smanjiti za njih kao i za gumb te opcije.

@media (max-width: 360px) {
    .listcontainer {
        padding: 0 8px;
    }

    .listsearch {
        padding: 10px;
    }

    .lsoptionitem {
        flex-direction: column;
        align-items: flex-start;
    }

    .lsoptioninput {
        max-width: 100%;
    }
}

* {
    max-width: 100%;
}

html,
body {
    overflow-x: hidden;
}

```

Za ekrane širine manje ili jednake 360 piksela smanjen je padding containera i searcha. Elementi lsoptionitem-a poravnati su u lijevo, a input elementi sada zauzimaju 100 % širine containera(nalaze se u svojem redu). Širina sadržaja stranice ne smije biti veća od 100 %, a eventualni višak će biti sakriven.

#### KOMPONENTA SEARCHITEM KOJA SE KORISTI NA STRANICI LIST

Sada kada smo završili s izradom stranice list, potrebno je napraviti komponentu searchitem koja će se koristiti na stranici list, a trenutačno je zakomentirana. Napravimo mapu seachitem u mapi components s odgovarajućim .jsx i .css datotekama. Sada možemo otkomentirati komponentu na stranici list. Struktura komponente izgledat će ovako:

```

import "./searchitem.css";
const Searchitem = () => {
  return (
    <div className="searchitem">
      
      <div className="sidesc">
        <h1 className="siTitle">Tower Street Apartments</h1>
        <span className="siDistance">500m from center</span>
        <span className="siTaxiOp">Free airport taxi</span>
        <span className="siSubtitle">
          Studio Apartment with Air conditioning
        </span>
        <span className="siFeatures">
          Entire studio . 1 bathroom . 21m2 1 full bed
        </span>
        <span className="siCancelOp">Free cancellation </span>
        <span className="siCancelOpSubtitle">
          You can cancel later, so lock in this great price today!
        </span>
      </div>
      <div className="sidetails">
        <div className="sirating">
          <span>Excellent</span>
          <button>8.9</button>
        </div>
        <div className="sidailetexts">
          <span className="siprice">67$</span>
          <span className="sitaxop">Includes taxes and fees</span>
          <button className="sicheckbutton">See availability</button>
        </div>
      </div>
    </div>
  );
};

export default Searchitem;

```

Dakle, div container imat će sliku apartmana te sve potrebne informacije vezane za apartman i gumb koji će prikazati raspoloživost. Sada je potrebno komponenti dodati izgled s pomoću CSS stila:

```

.searchitem {
  display: flex;
  justify-content: space-between;
  gap: 24px;
  border: 1px solid #e0e0e0;
}

```

```

border-radius: 12px;
padding: 16px;
margin-bottom: 24px;
background-color: #fff;
box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);
transition: transform 0.25s ease, box-shadow 0.25s ease;
}

.searchitem:hover {
  transform: translateY(-4px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.12);
}

```

Elementi containera(slika, opis i detalji) imaju razmak među sobom od 24 piksela. Granica containera je sive boje sa zakrvljenim rubovima. Sadržaj containera pomaknut je od granica za 16 piksela te container ima donju marginu od 24 piksela i bijelu pozadinsku boju. Osim toga, definirana je crna sjena ispod containera s neprozirnosti 5 % koja hover-om postaje manje neprozirna te je veća i više ispod elementa. Kada hover-amo container, također, on se pomakne za 4 piksela prema gore.

```

.siimg {
  width: 200px;
  height: 200px;
  object-fit: cover;
  border-radius: 10px;
  flex-shrink: 0;
}

.sidesc {
  flex: 2;
  display: flex;
  flex-direction: column;
  gap: 8px;
  justify-content: space-between;
}

.sidetails {
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: flex-end;
  text-align: right;
}

```

Slika zauzima 200 piksela širine i visine te je višak odrezan. Rubovi slike su zakrivljeni te nije dopušteno smanjenje slike. Opis zauzima duplo više mesta od detalja te je svaki njegov element u svome redu s razmakom od 8 piksela. Elementi detalja su također svaki u svome redu s razmakom između njih te se oni nalaze na desnoj strani containera kao i njihov tekst koji je poravnat udesno.

```
.siTitle {  
    font-size: 20px;  
    font-weight: 700;  
    color: #0d2ca8;  
    margin-bottom: 4px;  
}  
  
.siDistance,  
.siTaxiOp,  
.siSubtitle,  
.siFeatures,  
.siCancelOpSubtitle {  
    font-size: 14px;  
    color: #555;  
}  
  
.siTaxiOp {  
    background-color: #00b341;  
    color: #fff;  
    padding: 3px 8px;  
    border-radius: 6px;  
    font-size: 13px;  
    font-weight: 500;  
    width: fit-content;  
}
```

Naslovu opisa definirane su veličina, debljina te boja fonta kao i donja margina. Ostalim elementima opisa, zajedno su definirani veličina te boja fonta. Opciji koja prikazuje informacije o tome je li taxi s aerodroma besplatan određeni su pozadinska te boja teksta, padding sa svih strana, zaobljeni rubovi, veličina i debljina fonta te će širina elementa odgovarati njenome sadržaju.

```
.siSubtitle {  
    font-weight: 600;  
    color: #333;  
}  
  
.siFeatures {  
    color: #666;  
}
```

```

.siCancelOp {
    font-size: 13px;
    color: #00b341;
    font-weight: 700;
}

.siCancelOpSubtitle {
    color: #00b341;
    font-size: 13px;
}

```

Podnaslovu su definirane debljina i boja fonta, a značajkama samo boja. Opcijama otkazivanja definirane su veličina, debljina te boja fonta, dok njenome podnaslovu nije definirana debljina fonta.

```

.sirating {
    display: flex;
    justify-content: flex-end;
    align-items: center;
    gap: 8px;
}

.sirating > span {
    font-weight: 500;
    font-size: 15px;
    color: #333;
}

.sirating > button {
    background-color: #000040;
    color: white;
    border: none;
    padding: 6px 10px;
    border-radius: 6px;
    font-weight: 700;
    cursor: pointer;
    transition: background-color 0.2s ease;
}

.sirating > button:hover {
    background-color: #001a80;
}

```

Elementi ocjene(opis i gumb), nalaze se na desnoj strani njihovog containera, centrirane visine te s razmakom među njima od 8 piksela. Opisu su definirane debljina, veličina i boja teksta, dok gumbu nije definirana

veličina fonta. Gumb osim toga ima tamno plavu pozadinu bez granice te padding sa svih strana. Kutevi su mu zaobljeni, a hover-om kurzor postaje pointer te pozadinska boja svjetlija.

```
.siddetailtexts {  
    display: flex;  
    flex-direction: column;  
    gap: 6px;  
    align-items: flex-end;  
    text-align: right;  
}  
  
.siprice {  
    font-size: 22px;  
    font-weight: 700;  
    color: #111;  
}  
  
.sitaxop {  
    font-size: 13px;  
    color: #888;  
}
```

Span elementi cijene i poreza te gumb su svaki u svome redu s razmakom od 6 piksela te su poravnati udesno kao i njihov tekst. Cijeni su definirani veličina, debljina te boja fonta, dok porezu nije definirana debljina.

```
.sicheckbutton {  
    background-color: #00a2ff;  
    color: white;  
    font-weight: 600;  
    padding: 10px 18px;  
    border: none;  
    border-radius: 6px;  
    cursor: pointer;  
    transition: background-color 0.25s ease, transform 0.2s ease;  
}  
  
.sicheckbutton:hover {  
    background-color: #0088dd;  
    transform: translateY(-2px);  
}
```

Gumb ima definirane standardne stvari kao što su pozadinska te boja fonta, debljina fonta, padding sa svih strana, zakriviljeni rubovi te nevidljiva granica. Na hover, kurzor postaje pointer, pozadinska boja tamnija te se gumb pomakne za 2 piksela prema gore.

```

@media (max-width: 1024px) {
  .searchitem {
    gap: 20px;
    padding: 14px;
  }

  .siimg {
    width: 180px;
    height: 180px;
  }

  .siTitle {
    font-size: 18px;
  }

  .siprice {
    font-size: 20px;
  }
}

```

Za ekrane širine manje ili jednake 1024 piksela, elementi containera imaju manji razmak među sobom te padding. Slika je također manja, kao i veličina fonta glavnog naslova te cijene.

```

@media (max-width: 768px) {
  .searchitem {
    flex-direction: column;
    align-items: stretch;
    text-align: left;
    gap: 16px;
    padding: 16px;
  }

  .siimg {
    width: 100%;
    height: 220px;
    margin-bottom: 8px;
  }

  .sidesc {
    gap: 12px;
  }

  .sidetails {
    align-items: stretch;
    text-align: left;
  }
}

```

```

        gap: 16px;
    }

    .sideshowtexts {
        align-items: flex-start;
        text-align: left;
    }

    .sirating {
        justify-content: space-between;
        margin-bottom: 8px;
    }

    .siTaxiOp {
        align-self: flex-start;
    }
}

```

Za ekrane širine manje ili jednake 768 piksela, svaki element containera je u svome redu, tekst je poravnat ulijevo, razmak između elemenata je 16 piksela te razmak sadržaja containera od njegovih granica sa svim stranama iznosi 16 piksela. Slika zauzima cijelu širinu containera te je visoka 220 piksela i ima donju marginu od 8 piksela. Elementi opisa imaju razmak među sobom od 12 piksela što je više nego prije, budući da sada svaki zauzima cijeli red. Detaljima je tekst poravnat ulijevo te imaju razmak među sobom od 16 piksela. Cijena, porez te gumb su također poravnati ulijevo kao i opcije za taxi. Span i gumb ocjene imaju razmak među sobom(nalaze se na krajnje lijevoj odnosno desnoj poziciji) te donju marginu od 8 piksela.

```

@media (max-width: 480px) {
    .searchitem {
        padding: 12px;
        margin-bottom: 16px;
        gap: 12px;
    }

    .siimg {
        height: 180px;
    }

    .siTitle {
        font-size: 18px;
    }

    .siDistance,
    .siTaxiOp,
    .siSubtitle,
    .siFeatures,

```

```

.siCancelOpSubtitle {
    font-size: 13px;
}

.siPrice {
    font-size: 20px;
}

.sicheckbutton {
    padding: 12px 16px;
    font-size: 14px;
    width: 100%;
}

.sirating > button {
    padding: 8px 12px;
    font-size: 14px;
}
}

```

Za ekrane širine manje ili jednake 480 piksela, razmak među elementima containera je manji kao i padding te donja margina containera. Također, manja je visina slike te font naslova. Ostalim elementima opisa smanjena je veličina fonta, kao i cijeni. Gumb za raspoloživost, definirana je veličina fonta, povećan je gornji i donji padding, a smanjen lijevi i desni te sada gumb zauzima cijelu širinu containera. Gumbu ocjene povećan je padding te je definirana veličina fonta.

```

@media (max-width: 360px) {
    .searchitem {
        padding: 10px;
        border-radius: 8px;
    }

    .siimg {
        height: 160px;
    }

    .siTitle {
        font-size: 16px;
    }

    .siPrice {
        font-size: 18px;
    }
}

```

Za ekrane širine manje ili jednake 360 piksela, containeru je smanjen padding te zakriviljenost rubova. Slici je smanjena visina, a glavnom naslovu te cijeni veličina fonta.

## STRANICA HOTEL

Potrebno je napraviti stanicu koja će se prikazivati kada kliknemo na neki hotel.

### UVEĆANI PRIKAZ SLIKA TE POMICANJE PO SLIKAMA

Napravimo kostur stranice koristeći napravljene komponente, slike te sam tekst koji će se prikazivati na stranici sa svojim div container-ima i wrapper-ima:

```
import Header from "../../components/header/Header";
import Maillist from "../../components/maillist/maillist";
import Footer from "../../components/footer/footer";
import Navbar from "../../components/navbar/navbar";
import { useState } from "react";
import "./hotel.css";

const Hotel = () => {
  const [slidenum, setslidenum] = useState(0);
  const [open, setopen] = useState(false);
  const photos = [
    {
      src: "/20210710_084619.jpg",
    },
    {
      src: "/20210710_085443.jpg",
    },
    {
      src: "/20210710_085121.jpg",
    },
    {
      src: "/20210710_085154.jpg",
    },
    {
      src: "/20210710_085438.jpg",
    },
    {
      src: "/20210710_085443.jpg",
    },
  ];
  const handleopen = (i) => {
    setslidenum(i);
    setopen(true);
  }
}
```

```
};

const handlemove = (direction) => {
    let newslidenumber;
    if (direction === "l") {
        newslidenumber = slidenumber === 0 ? 5 : slidenumber - 1;
    } else {
        newslidenumber = slidenumber === 5 ? 0 : slidenumber + 1;
    }
    setslidenumber(newslidenumber);
};

return (
    <div>
        <Navbar></Navbar>
        <Header type="list"></Header>
        <div className="hotelcontainer">
            {open && (
                <div className="slider">
                    <span className="close" onClick={() => setopen(false)}>
                        ✕
                    </span>
                    <span className="arrow" onClick={() => handlemove("l")}>
                        ←
                    </span>
                    <div className="sliderwrapper">
                        <img src={photos[slidenumber].src} alt="" className="sliderimg" />
                    </div>
                    <span className="arrow" onClick={() => handlemove("r")}>
                        →
                    </span>
                </div>
            )}
            <div className="hotelwrapper">
                <button className="booknow">Reserve or book now</button>
                <h1 className="hotelttitle">Apartments Ani</h1>
                <div className="hoteladress">
                    {/* Logo */}
                    <span>Odranska ulica 8 Zagreb</span>
                </div>
                <span className="hoteldistance">
                    Excelent location -500m from center
                </span>
                <span className="hotelpricehighlight">
```

Book a stay over 67\$ at this property and get a free airport taxi

```
</span>
<div className="hotelimages">
  {photos.map((photo, index) => (
    <div className="hotelimgwrapper" key={index}>
      <img
        onClick={() => handleopen(index)}
        src={photo.src}
        alt=""
        className="hotelimg"
      />
    </div>
  )))
</div>
<div className="hoteldetails">
  <div className="hoteldetailstexts">
    <h1 className="hotelttitle">Stay in the heart of Krakow</h1>
    <p className="hoteldesc">
      Located a 5-minute walk from St. Florian's Gate in Krakow, To
    
```

wer

d

Street Apartments has accommodations with air conditioning an

is

e

st

free WiFi. The units come with hardwood floors and feature a fully equipped kitchenette with a microwave, a flat-screen TV, and a private bathroom with shower and a hairdryer. A fridge also offered, as well as an electric tea pot and a coffee machine. Popular points of interest near the apartment includ

Cloth Hall, Main Market Square and Town Hall Tower. The nearest airport is John Paul II International Kraków Balice, 16.1 km from Tower Street Apartments, and the property offers a paid airport shuttle service.

```
</p>
</div>
<div className="hoteldetailsprice">
  <div className="hotelDetailsPrice">
    <h1>Perfect for a 9-night stay !</h1>
    <span>
      Located in the real heart of Krakow, this property has an
      excellent location score of 9.8!
    </span>
    <h2>
```

```

        <b>$945</b>(9 nights)
      </h2>
      <button>Reserve or Book Now !</button>
    </div>
  </div>
</div>
<Maillist />
<Footer></Footer>
</div>
</div>
);
};

export default Hotel;

```

Unutar div elementa klase hotelimages definirano je da će se naše slike prikazati po redu, svaka sa svojim wrapperom, a klikom na neku od njih poziva se funkcija koja postavlja broj slike koja je kliknuta te open poprima istinitu vrijednost. Sada se na početku hotelcontainer-a prikazuje "slider" odnosno kliknuta slika te span elementi strelica za otvaranje ostalih slika i križić za zatvaranje slike. Kada kliknemo na križić, open se postavlja na false što znači da će se slider zatvoriti. Pomicanje po slikama klikom na strelice definirano je **handlemove()** funkcijom koja prima smjer kao argument te postavlja novu sliku. Ako je trenutačna slika s indeksom 0 te je kliknuta lijeva strelica, onda se prikazuje peta slika, a inače slika s indeksom manjim za 1. Isto tako, ako je trenutačno otvorena slika s indeksom 5 te je kliknuta desna strelica, onda će biti prikazana slika s indeksom 0, a inače slika s indeksom većim za 1. Ovako je definirano ponašanje za 6 slika, da se može beskonačno klikati strelice.

## GOOGLE MAPS

Inspiracija izrade i integracije Google maps servisa napravljena je korištenjem [Tutorial videozapisa](#) Za samu izradu elementa koristila se web-stranica [Generator maps html koda](#) koja nam unosom potrebnih parametara za našu stranicu stvori div/IFrame komponentu koja se onda ubaci u naš Hotel.jsx kod. Parametri koje sam unio u stranicu pod Enter your settings su -> Title: Apartman Ani, Address: Odranska Ulica 8, Coordinates -> program automatski popunio, Height: 600, Width: 100 %, te ostali parametri poput views, Zoom su imali automatski postavljene default vrijednosti na Map, 400 m (district), Auto-fit Width je uključen te on omogućuje da se expanda na veličinu Containera. Nakon unosa željenih parametara možemo s desne strane prekopirati iFrame kod koji koristimo na Hotel.jsx fileu. Nije bilo potrebno instalirati vanjske pakete niti povezivati se s Google maps Api-em što omogućuje lakše integriranje u stranicu. Na kraju hotelwrapper-a, nakon hoteldetails div elementa, potrebno je dodati sljedeći kod:

```

<div className="gmap-frame">
  <iframe
    width="100%"
    height="600"
    frameborder="0"
    scrolling="no"
    marginheight="0"
  >

```

```
marginwidth="0"
src="https://maps.google.com/maps?width=100%25&height=600&hl=en&
mp;q=Odranska%20Ulica%208+(Apartman%20Ani)&t=&z=14&ie=UTF8&iwloc=B&output=embed"
></iframe>
</div>
```

---

#### INLINE CSS OBJAŠNJENJA:

- **width="100%"** - Širina **<iframe>** elementa - 100 % znači da će iframe zauzeti cijelu širinu roditeljskog elementa.
- **height="600"** - Visina iframe-a u pikselima. U ovom slučaju 600px.
- **frameborder="0"** - Stari HTML atribut koji uklanja okvir (border) oko iframe-a. Vrijednost 0 znači nema okvira.
- **scrolling="no"** - Kontrolira da li iframe prikazuje scroll barove. no znači da scroll bar neće biti prikidan, čak ako sadržaj iframe-a prelazi veličinu.
- **marginheight="0" i marginwidth="0"** - Definiraju vanjske margine (padding) unutar iframe-a. Vrijednost 0 znači da nema margina.

---

#### OBJAŠNJENJE PARAMETARA U URL-U:

- **width=100%25** - širina mape (100 %, %25 je URL encoding za %)
- **height=600** - visina mape (600px)
- **hl=hr** - jezik sučelja mape (hr - hrvatski)
- **q=Odranska Ulica 8 (Apartman Ani)** - lokacija koju mapa prikazuje
- **t=** - tip mape (m = standard, k = satelit, h = teren itd.); ovdje prazan = standard
- **z=14** - zoom razina (1 = cijeli svijet, 20 = ulica)
- **ie=UTF8v** - encoding za URL, UTF-8
- **iwloc=B** - inicijalna pozicija info window-a (oznaka B na mapi)
- **output=embed** - vraća mapu u embed formatu za iframe

#### CSS STIL

Naposljeku, potrebno je našim elementima dodati CSS stil kako bi stranica izgledala lijepo i interaktivno:

```
.hotelcontainer {
  display: flex;
  align-items: center;
  flex-direction: column;
  margin-top: 20px;
  padding: 0 20px;
  box-sizing: border-box;
}
.slider span {
  font-size: 24px;
```

```

    cursor: pointer;
    margin: 0 10px;
}
.hotelwrapper {
    width: 100%;
    max-width: 1024px;
    display: flex;
    flex-direction: column;
    gap: 20px;
    position: relative;
}

```

Elementi containera, nalaze se svaki u svome redu, poravnati u sredinu po širini. Container ima gornju marginu od 20 piksela kako bi bio odmaknut od zaglavlja te lijevi i desni padding od 20 piksela. Span elementima slidera(strelicama i križiću), definirane su lijeva i desna margina, veličina fonta te prilikom hovera preko njih, kurzor postaje pointer. Wrapper hotela, koji u biti sadrži sve osim komponenti i slidera, zauzima 100 % širine hotel containera s maksimalnom širinom od 1024 piksela. Njegovi elementi smješteni su svaki u svome redu s razmakom među njima od 20 piksela. Pozicija mu je relativna kako bi booknow gumb mogli pozicionirati apsolutno u odnosu na njega.

```

.hoteltitle {
    font-size: 28px;
    font-weight: 700;
    color: #222;
    margin-bottom: 5px;
}

.hoteladdress {
    font-size: 14px;
    display: flex;
    align-items: center;
    color: #555;
}

.hoteldistance {
    color: rgb(0, 191, 255);
    font-weight: 500;
    font-size: 14px;
}

.hotelpricehighlight {
    color: green;
    font-weight: 600;
    font-size: 14px;
}

```

Glavnom naslovu definirani su veličina, debljina i boja fonta te donja margina kako bi bio razmaknut od adrese. Adresa ima veličinu fonta od 14 piksela sive boje te joj je visina centrirana. Također, udaljenost i istaknute informacije u vezi cijene imaju određenu boju, debljinu te veličinu fonta.

```
.booknow {  
    position: absolute;  
    top: 10px;  
    right: 0;  
    border: none;  
    padding: 12px 24px;  
    background: linear-gradient(135deg, rgb(0, 115, 255), rgb(0, 90, 200));  
    color: #ffffff;  
    border-radius: 8px;  
    font-weight: bold;  
    cursor: pointer;  
    transition: all 0.3s ease;  
}  
  
.booknow:hover {  
    background: linear-gradient(135deg, rgb(0, 100, 220), rgb(0, 70, 180));  
    transform: translateY(-2px);  
}
```

Gumb booknow postavljen je na krajnje desnom mjestu hotelwrapper-a udaljen 10 piksela od njegova vrha. Granica mu je skrivena te mu je definiran padding sa svih strana. Pozadinska boja započinje sa svjetlo plavom iz gornjeg lijevog kuta te prelazi dijagonalno u tamno plavu. Rubovi su mu zaobljeni te mu je tekst bijel i podebljan. Prilikom hover-a, cursor postaje pointer te gumb postaje tamniji i pomakne se za 2 piksela prema gore.

```
.hotelimages {  
    display: flex;  
    flex-wrap: wrap;  
    justify-content: space-between;  
    gap: 10px;  
}  
  
.hotelimgwrapper {  
    flex-basis: 32%;  
    border-radius: 10px;  
    overflow: hidden;  
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.08);  
    transition: transform 0.3s ease, box-shadow 0.3s ease;  
}  
  
.hotelimgwrapper:hover {
```

```

    transform: scale(1.03);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.15);
}

.hotelimg {
    width: 100%;
    height: 200px;
    object-fit: cover;
    display: block;
}

```

Slike su po 3 u svakome redu s razmakom između njih te razmakom između redova od 10 piksela. Rubovi slika su zaobljeni te je višak sakriven. Također je definirana sjena koja se povećava na hover kao i sama slika. Slike su širine 100 % wrappera, visine 200 piksela, prikazane su kao block element te zauzimaju cijeli wrapper.

```

.hoteldetails {
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
    gap: 30px;
    margin-top: 30px;
}

.hoteldetailstexts {
    flex: 3;
    background-color: #ffffff;
    border-radius: 10px;
    padding: 20px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
}

.hoteldetailstexts h1 {
    font-size: 22px;
    font-weight: 600;
    color: #222;
}

.hoteldesc {
    font-size: 15px;
    line-height: 1.6;
    color: #444;
    margin-top: 20px;
    text-align: justify;
}

```

```
.hoteldetailsprice {  
    flex: 1.3;  
    display: flex;  
    justify-content: center;  
}
```

Elementi containera detalja imaju razmak među sobom od 30 piksela te započinju na krajnje lijevom mjestu containera, a container ima gornju marginu od 30 piksela. Tekstovi zauzimaju 3/4.3, dok detalji cijene zauzimaju 1.3/4.3 širine containera. Također, tekstovima su određeni pozadinska boja, zaobljeni kutevi, padding sa svih strana te sjena. Glavnome naslovu definirani su veličina, debljina te boja fonta, dok opisu nije definirana debljina fonta. Visina linija paragrafa opisa iznosi 1.6 te opis ima gornju marginu i obostrano poravnat tekst. Detalji cijene osim definirane veličine, potpuno su u containeru te im je sadržaj centriran.

```
.hotelDetailsPrice {  
    background-color: #8ef6fc;  
    border-radius: 12px;  
    padding: 30px 25px;  
    display: flex;  
    flex-direction: column;  
    gap: 20px;  
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);  
    width: 100%;  
    max-width: 360px;  
}  
  
.hotelDetailsPrice > h1 {  
    font-size: 20px;  
    color: #333;  
    font-weight: 600;  
}  
  
.hotelDetailsPrice > span {  
    font-size: 14px;  
    color: #444;  
    line-height: 1.5;  
}  
  
.hotelDetailsPrice > h2 {  
    font-size: 22px;  
    font-weight: 500;  
    color: #111;  
}
```

Detalji cijene koji se nalaze u containeru sličnog naziva, imaju definiranu boju pozadine, zakrivljene rubove i padding sa svih strana. Osim toga, njegovi elementi, nalaze se svaki u svome redu razmaknuti za 20 piksela te

on ima sjenu i zauzima 100 % širine svog containera s maksimalnom širinom od 360 piksela. Za naslove unutar detalja cijena definirani su veličina, boja i debljina fonta, dok je za span element koji opisuje lokaciju umjesto debljine fonta definirana visina reda.

```
.hotelDetailsPrice > button {  
    border: none;  
    padding: 14px 20px;  
    background: linear-gradient(135deg, rgb(0, 115, 255), rgb(0, 90, 200));  
    color: #fff;  
    border-radius: 8px;  
    font-weight: bold;  
    font-size: 16px;  
    cursor: pointer;  
    transition: all 0.3s ease;  
}  
  
.hotelDetailsPrice > button:hover {  
    background: linear-gradient(135deg, rgb(0, 100, 220), rgb(0, 70, 180));  
    transform: translateY(-2px);  
}
```

Gumb unutar detalja cijene ima gotovo isti stil s razlikom u veličini paddinga i dodatkom veličine fonta te je maknuto apsolutno pozicioniranje jer se ovaj gumb nalazi na dnu detalja cijene slijedno u svome redu ispod cijene.

```
@media (max-width: 900px) {  
    .hoteldetails {  
        flex-direction: column;  
    }  
  
    .hoteldetailsprice {  
        width: 100%;  
    }  
  
    .hotelimg {  
        height: 180px;  
    }  
  
    .booknow {  
        position: static;  
        width: 100%;  
        margin-bottom: 10px;  
    }  
}
```

```

@media (max-width: 600px) {
    .hotelimgwrapper {
        flex-basis: 48%;
    }
}

@media (max-width: 400px) {
    .hotelimgwrapper {
        flex-basis: 100%;
    }
}

```

Za ekrane širine manje ili jednake 900 piksela, detalji cijene prelaze u svoj red te zauzimaju 100 % širine hoteldetails containera. Slikama je smanjena visina za 20 piksela, a gumb booknow koji se prije nalazio na krajnje desnoj poziciji hoterwrapper-a u istome redu kao glavni naslov, sada je pozicioniran statički i zauzima 100 % širine wrappera(u svome jer redu iznad naslova) te ima donju marginu od 10 piksela kako bi bio odmaknut od naslova. Za ekrane širine manje ili jednake 600 odnosno 400 piksela slike su dvije odnosno jedna po svakome redu.

```

.slider {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw;
    height: 100vh;
    background-color: rgba(0, 0, 0, 0.367);
    z-index: 999;
    display: flex;
    align-items: center;
}

.sliderwrapper {
    width: 100%;
    height: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
}

.close {
    position: absolute;
    top: 20px;
    right: 20px;
}

```

```
.sliderimg {  
    width: 80%;  
    height: 80vh;  
}
```

Slider ne prati tok dokumenta nego ima fiksnu poziciju preko cijelog ekrana. Nalazi se ispred ostatka stranice te je visina njegovih elemenata poravnata u sredinu. Ostatak stranice je zatamnjen zbog crne pozadinske boje neprozirnosti 36.7 %. Wrapper slike zauzima širinu i visinu slidera koliko može te mu je sadržaj centriran. Križić za zatvaranje slidera ima apsolutnu poziciju u odnosu na slider te se tako nalazi 20 piksela ispod vrha te 20 piksela lijevo od granica slidera. Naposljetku, slika će zauzimati 80 % širine wrappera te 80 % visine zaslona.

## STRANICA FORM ZA UNOS I UREĐIVANJE SMJEŠTAJNIH JEDINICA

ApartmentForm.jsx je komponenta omogućuje dodavanje i uređivanje smještajnih jedinica apartmana ili soba putem jednostavnog i intuitivnog sučelja. Podaci se spremaju u backend , a moguće je i dodavanje fotografija te označavanje dostupnih pogodnosti “amenities”. Komponenta koristi React hook-ove i react-router-dom za dinamičko dohvaćanje, navigaciju i upravljanje stanjem forme.

---

### UVOD POTREBNIH MODULA

```
import React, { useState, useEffect } from "react";  
import { useParams, useNavigate } from "react-router-dom";  
import "./ApartmentForm.css";
```

Na početku su uvezeni osnovni React moduli i hook-ovi:

- useState koristi se za upravljanje lokalnim stanjem forme (formData, images).
- useEffect omogućuje dohvaćanje podataka s backend-a ako je u URL-u prisutan id.
- useParams dohvaca dinamički parametar id iz URL-a.
- useNavigate omogućuje navigaciju korisnika na drugu stranicu (npr. /admin nakon uspješnog spremanja).
- ApartmentForm.css sadrži stilove za vizualno oblikovanje forme.

---

### INICIJALNO STANJE KOMPONENTE

```
const [formData, setFormData] = useState({  
    unitName: "",  
    mainDescriptionTitle: "",  
    mainDescription: "",  
    secondaryDescriptionTitle: "",  
    secondaryDescription: "",  
    price: "",  
    isApartment: true,  
    amenities: {  
        parking: false,
```

```

    wifi: false,
    breakfast: false,
    towels: false,
    shampoo: false,
    hairDryer: false,
    heater: false,
    airConditioning: false,
  },
);

```

Ovim kodom definiramo početno stanje forme formData. Svi tekstualni unosi naziv, opis, cijena... inicijalno su prazni stringovi, dok su opcije pogodnosti "amenities" boolean vrijednosti koje smo predefinirali i koji odgovaraju bazi podataka. isApartment služi za razlikovanje tipa jedinice apartman ili soba i inicijalno je postavljen na true što predstavlja da je jedinica apartman.

```
const [images, setImages] = useState([]);
```

images čuva listu slika koje korisnik učita svaka slika sadrži file i lokalni url za prikaz.

#### HANDLECHANGE(E) UNIVERZALNA OBRADA SVIH POLJA U FORMI

```

const handleChange = (e) => {
  const { name, value, type, checked } = e.target;
  if (name in formData.amenities) {
    setFormData({
      ...formData,
      amenities: { ...formData.amenities, [name]: checked },
    });
  } else {
    setFormData({ ...formData, [name]: value });
  }
};

```

#### DESTRUKTURIRANJE EVENT OBJEKTA

- Kada korisnik unese ili promijeni neki podatak u formi, React event (e) sadrži sve potrebne informacije o tom input elementu.

Destrukturiranjem dohvaćamo:

- name naziv inputa (npr. "unitName", "wifi")
- value tekstualna ili numerička vrijednost inputa
- type tip elementa (text, checkbox, radio, number...)
- checked boolean vrijednost checkbox-a (true ili false)

#### PROVJERA JE LI PROMIJENJENO "AMENITY" POLJE

Linija:

```
if (name in formData.amenities)
```

provjerava nalazi li se naziv inputa (npr. "wifi", "towels") unutar objekta formData.amenities. Time razlikujemo "obične" inpute tekst, broj, opis od checkbox opcija za pogodnosti.

## AŽURIRANJE STATE-A

---

Ako je riječ o pogodnostima (checkbox), ažurira se samo taj checkbox unutar formData.amenities, dok se ostali podaci ne diraju:

```
amenities: { ...formData.amenities, [name]: checked }
```

Time se koristi spread operator ... kako bi se sačuvale postojeće vrijednosti drugih checkbox-ova. U suprotnom slučaju tekstualni input, radio gumb..., ažurira se odgovarajuće polje u glavnom formData objektu:

```
setFormData({ ...formData, [name]: value });
```

Time korisnik može slobodno unositi tekst, broj ili označavati pogodnosti sve promjene se odmah spremaju u React state bez potrebe za dodatnim funkcijama za svaki pojedini input.

---

## HANDLEIMAGEUPLOAD(E) UČITAVANJE SLIKA

```
const handleImageUpload = (e) => {
  const files = Array.from(e.target.files);
  const newImages = files.map((file) => ({
    file,
    url: URL.createObjectURL(file),
  }));
  setImages((prev) => [...prev, ...newImages]);
};
```

Služi za dohvati svih odabralih datoteka

```
const files = Array.from(e.target.files);
```

e.target.files vraća FileList objekt. Funkcija Array.from() ga pretvara u pravu JS listu kako bi se mogla koristiti map() i druge metode.

Nakon toga ide kreiranje objekta za svaku sliku

```
const newImages = files.map((file) => ({
  file,
  url: URL.createObjectURL(file),
}));
```

Za svaku sliku se stvara privremeni lokalni URL pomoću URL.createObjectURL(file). Taj URL omogućuje trenutni prikaz slike u pregledniku, bez slanja na server. Svaka slika se sprema kao objekt s dva ključa: \* file originalni objekt datoteke \* url lokalni link za prikaz

## AŽURIRANJE STANJA S NOVIM SLIKAMA

---

```
setImages((prev) => [...prev, ...newImages]);
```

Spread operator ... dodaje nove slike uz već postojeće (ako korisnik više puta učitava). Time korisnik vidi odmah pregled učitanih slika u formi, prije nego ih pošalje backendu.

---

#### REMOVEIMAGE(INDEX) BRISANJE SLIKE IZ PREGLEDA

```
const removeImage = (index) => {
  setImages((prev) => prev.filter((_, i) => i !== index));
};
```

Funkcija prima indeks slike koju treba izbrisati. setImages ažurira trenutno stanje images tako da koristi .filter() metodu te zadrži sve slike osim one čiji je indeks jednak index koji je poslan.

```
prev.filter((_, i) => i !== index)
```

underscore \_ se koristi jer nam sama vrijednost nije potrebna, samo indeks i. Rezultat je nova lista slika bez obrisanе. Time postižemo kada korisnik klikne gumb “Remove” ispod slike, ona se odmah briše iz prikaza i ne šalje se na server.

---

#### JSX STRUKTURA KOMPONENTE APARTMENTFORM

Ovaj dio predstavlja vizualni prikaz forme i povezuje korisnički unos s logikom komponente. Svaki input element koristi React state formData i images te funkcije za obradu dogadaja handleChange, handleImageUpload, removeImage, handleSubmit. Cilj ove forme je omogućiti unos, uređivanje i prikaz podataka o smještajnim jedinicama (apartmanima ili sobama).

```
return (
  <div className="form-container">
    <h2>{id ? `Edit Unit #${id}` : "Create New Unit"}</h2>
```

Komponenta vraća JSX sadržaj unutar glavnog div elementa s klasom “form-container” što je glavni okvir forme. Naslov (**<h2>**) se dinamički mijenja ovisno o postojanju id parametra

- Ako id postoji prikazuje se “Edit Unit #id”.
- Ako id ne postoji prikazuje se “Create New Unit”.

Na taj način se koristi ista forma za uređivanje i dodavanje smještaja.

Glavna HTML

oznaka

```
<form onSubmit={handleSubmit} className="apartment-form">
```

**onSubmit={handleSubmit}** povezuje formu s funkcijom koja šalje podatke na backend kada korisnik klikne gumb Submit ili Update. Klasa “apartment-form” koristi se za stilizaciju.

---

#### UNOS NAZIVA JEDINICE

```
<label>Unit Name</label>
<input
  type="text"
  name="unitName"
  value={formData.unitName}
  onChange={handleChange}
  required
/>
```

Standardni tekstualni input koji koristi kontroliranu vrijednost `value={formData.unitName}`. `onChange={handleChange}` osigurava da se promjena odmah pohrani u `formData`. `required` osigurava da polje ne može ostati prazno pri slanju forme.

#### ODABIR TIPOA JEDINICE APARTMENT ILI ROOM

---

```
<label>Unit Type</label>
<div className="radio-group">
  <label>
    <input
      type="radio"
      name="isApartment"
      checked={formData.isApartment === true}
      onChange={() => setFormData({ ...formData, isApartment: true })}
    />
    Apartment
  </label>
  <label>
    <input
      type="radio"
      name="isApartment"
      checked={formData.isApartment === false}
      onChange={() => setFormData({ ...formData, isApartment: false })}
    />
    Room
  </label>
</div>
```

Koristi se radio grupa s dvije opcije

- Apartment `isApartment = true`
- Room `isApartment = false` checked svojstvo određuje koja je opcija trenutno aktivna. Klikom na neku opciju ažurira se `formData.isApartment` u stanju komponente pomoću `setFormData`. Klasa "radio-group" koristi se u CSS-u.

#### OPISNI PODACI GLAVNI I SEKUNDARNI OPISI

---

Ova sekcija omogućuje unos naslova i sadržaja za glavni i sekundarni opis.

```
<label>Main Description Title</label>
<input
  type="text"
  name="mainDescriptionTitle"
  value={formData.mainDescriptionTitle}
  onChange={handleChange}
/>

<label>Main Description</label>
<textarea
  name="mainDescription"
  rows="3"
  value={formData.mainDescription}
  onChange={handleChange}
/>

<label>Secondary Description Title</label>
<input
  type="text"
  name="secondaryDescriptionTitle"
  value={formData.secondaryDescriptionTitle}
  onChange={handleChange}
/>

<label>Secondary Description</label>
<textarea
  name="secondaryDescription"
  rows="3"
  value={formData.secondaryDescription}
  onChange={handleChange}
/>
```

Kombinacija `<input>` i `<textarea>` elemenata omogućuje unos tekstualnih podataka. Svako polje koristi `handleChange`, što omogućuje automatsko ažuriranje React state-a. `rows="3"` određuje visinu tekstualnog polja.

Tekstualna polja služe za unos naslova i sadržaja glavnog opisa te naslova i sadržaja sekundarnog opisa (dodatne informacije, npr. pogodnosti lokacije)

## CIJENA

---

```
<label>Price (€)</label>
<input
  type="number"
```

```
    name="price"
    value={formData.price}
    onChange={handleChange}
/>
```

Koristi se **type="number"** kako bi se omogućio unos samo numeričkih vrijednosti. Vrijednost se spremi u formData.price, a backend kasnije očekuje broj pretvoren pomoću **parseInt()** u handleSubmit. Ova vrijednost se koristi za prikaz cijene jedinice na stranici i u bazi podataka.

## UČITAVANJE I PREGLED SLIKA

---

```
<div className="image-upload">
  <label>Images</label>
  <input type="file" multiple accept="image/*" onChange={handleImageUpload} />
  <div className="image-preview">
    {images.map((img, index) => (
      <div key={index} className="preview-item">
        <img src={img.url} alt={`Image ${index + 1}`} />
        <button type="button" onClick={() => removeImage(index)}>
          Remove
        </button>
      </div>
    )))
  </div>
</div>
```

input **type="file"** omogućuje odabir više slika . **accept="image/\*"** ograničava odabir samo na slikovne datoteke. Funkcija handleImageUpload stvara lokalne URL-ove URL.createObjectURL za trenutačni prikaz.

**images.map()** prolazi kroz sve slike i prikazuje: \* **<img>** element za pregled slike \* Gumb "Remove" koji poziva **removeImage(index)** za brisanje pojedine slike iz liste \* Klase image-upload, image-preview, i preview-item koriste se za stilizaciju i raspored slika.

## ODABIR POGODNOSTI (AMENITIES)

---

```
<div className="checkbox-section">
  <h4>Amenities</h4>
  {Object.keys(formData.amenities).map((option) => (
    <label key={option} className="checkbox-label">
      <input
        type="checkbox"
        name={option}
        checked={formData.amenities[option]}
        onChange={handleChange}
      />
      {option.replace(/([A-Z])/g, " $1").replace(/\./, (str) => str.toUpperCase())}
    </label>
  ))}
```

```
        se()})
      </label>
    )))
</div>
```

**Object.keys(formData.amenities)** dohvaca sve nazive pogodnosti (wifi, parking, towels, itd.). Svaki checkbox koristi **checked={formData.amenities[option]}** za kontrolu stanja i ima **onChange={handleChange}** koji automatski ažurira state ispod svakog checkboxa prikazuje se naziv pogodnosti u ljepšem formatu: **option.replace(/([A-Z])/g, " \$1").replace(/\./, (str) => str.toUpperCase())** Regex dodaje razmak ispred velikih slova i kapitalizira prvo slovo npr. “airConditioning” u “Air Conditioning”. Klase checkbox-section i checkbox-label služe za CSS.

## GUMB ZA SLANJE FORME

---

```
<button type="submit" className="submit-btn">
  {id ? "Update" : "Submit"}
</button>
```

Gumb pokreće onSubmit događaj forme i time poziva funkciju handleSubmit(). Tekst na gumbu se dinamički mijenja ovisno o kontekstu \* Ako postoji id “Update” \* Inače “Submit” Klasa “submit-btn” definira stil gumba .

## APARTMENTFORM CSS

Ovaj CSS definira izgled, raspored i interakciju forme za unos smještajnih jedinica ApartmentForm. Naglasak je na svijetlom i čistom dizajnu s nagovještajem plave boje u skladu s temom turističkih i smještajnih web-aplikacija.

---

### .FORM-CONTAINER GLAVNI OKVIR FORME

```
.form-container {
  max-width: 650px;
  margin: 40px auto;
  padding: 30px;
  background-color: #e8f4fd;
  border-radius: 16px;
  box-shadow: 0 4px 15px rgba(0, 100, 200, 0.2);
  font-family: 'Segoe UI', sans-serif;
  color: #03426a;
}
```

- max-width: 650px forma ima maksimalnu širinu, centriра se u viewportu.
- margin: 40px auto dodaje razmak od vrha ili dna i automatski centriра po širini.
- padding: 30px unutarnji razmak.
- background-color: #e8f4fd svijetloplava pozadina.
- border-radius: 16px zaobljeni rubovi, vizualno ljepši dojam.
- box-shadow suptilna sjena ispod forme za dubinu.

- font-family koristi se čitljiv, moderan font.
  - color osnovna boja teksta tamno plava za dobar kontrast.
- 

## NASLOV FORME

```
.form-container h2 {  
    text-align: center;  
    color: #025c9a;  
    margin-bottom: 25px;  
}
```

Korišten je text align koji centrira naslov forme a boja #025c9a je jača nijansa plave, naglašava naslov.  
margin-bottom dodaje prostor prije sadržaja forme.

---

## OZNAČAVANJE POLJA (LABEL)

```
.apartment-form label {  
    display: block;  
    margin-top: 12px;  
    font-weight: 500;  
}
```

- display: block svaki label zauzima svoj red.
  - margin-top vertikalni razmak između polja.
  - font-weight: 500 srednje zadebljani font čini label jasno vidljivim.
- 

## POLJA ZA UNOS TEKSTA I BROJEVA

```
.apartment-form input[type="text"],  
.apartment-form input[type="number"],  
.apartment-form textarea {  
    width: 100%;  
    padding: 10px;  
    margin-top: 6px;  
    border: 1px solid #b5d5f5;  
    border-radius: 8px;  
    background-color: #f9fcff;  
    font-size: 15px;  
    transition: 0.2s ease;  
}
```

Polja su puna širina unutar forme te lagano plava pozadina i mekani rubovi **border-radius: 8px**.  
Transition omogućuje glatki prijelaz prilikom fokusa. font-size: 15px za ugodnu čitljivost.

---

## EFEKT FOKUSA

```
.apartment-form input:focus,  
.apartment-form textarea:focus {  
    border-color: #4aa3ff;  
    outline: none;  
    background-color: #ffffff;  
}
```

Kada korisnik klikne u input onda \* okvir border postaje plav vizualna potvrda fokusa. \* outline: none uklanja zadani “glow” efekt. \* pozadina postaje čisto bijela daje osjećaj aktivnog polja.

---

## UČITAVANJE SLIKA

Kontejner:

```
.image-upload {  
    margin-top: 20px;  
}
```

Ima marginu 20 pixela

Pregled slika:

```
.image-preview {  
    display: flex;  
    flex-wrap: wrap;  
    margin-top: 10px;  
    gap: 12px;  
}
```

- display: flex + flex-wrap: wrap slike se automatski raspoređuju u više redova.
- gap: 12px prostor između svake slike.

---

## POJEDINA SLIKA

```
.preview-item {  
    position: relative;  
    display: inline-block;  
}
```

Omogućuje pozicioniranje gumba “Remove” unutar slike pomoću **position: absolute**.

Slika:

```
.preview-item img {  
    width: 120px;  
    height: 100px;  
    object-fit: cover;  
    border-radius: 8px;
```

```
    border: 2px solid #b5d5f5;
}
```

Sve slike imaju jednaku veličinu. object-fit: cover osigurava da se slika proporcionalno izreže bez izobličenja. border-radius i border prate vizualni stil forme.

#### GUMB ZA BRISANJE SLIKE:

---

```
.preview-item button {
  position: absolute;
  top: 5px;
  right: 5px;
  background-color: #ff5c5c;
  color: white;
  border: none;
  border-radius: 6px;
  padding: 4px 6px;
  cursor: pointer;
  font-size: 12px;
  transition: background 0.2s;
}

.preview-item button:hover {
  background-color: #e74c3c;
}
```

Gumb se pozicionira u gornji desni kut slike. Crvena boja (#ff5c5c) jasno označava funkciju brisanja. Opet ima transition kao i gore što je objašnjeno. Hover nijansa (#e74c3c) daje vizualnu povratnu informaciju o akciji.

---

#### CHECKBOX

```
.checkbox-section {
  margin-top: 20px;
}

.checkbox-section h4 {
  margin-bottom: 10px;
  color: #025c9a;
}

.checkbox-label {
  display: block;
  margin-bottom: 6px;
}
```

margin-top odvaja sekciju od gornjih elemenata. h4 ima istu plavu boju kao naslovi radi vizualne konzistencije. Svaki checkbox ima display: block lijep vertikalni raspored.

---

#### SEKCIJA ZA CIJENU

```
.price-input {  
    display: flex;  
    align-items: center;  
    gap: 6px;  
}  
  
.price-input input {  
    flex: 1;  
}  
  
.price-input span {  
    font-weight: bold;  
    color: #03426a;  
}
```

display: flex omogućuje da broj i oznaka valute budu u istom redu. gap: 6px daje prostor između broja i “€” simbola. flex: 1 rasteže input tako da koristi preostali prostor.

---

#### RADIO GRUPA APARTMENT / ROOM

```
.radio-group {  
    display: flex;  
    gap: 1rem;  
    margin-bottom: 1rem;  
}  
  
.radio-group label {  
    display: flex;  
    align-items: center;  
    gap: 0.3rem;  
    cursor: pointer;  
}
```

Koristi flex raspored kako bi radio opcije bile horizontalno poravnate. gap: 1rem osigurava razmak između “Apartment” i “Room”. cursor: pointer čini cijelu oznaku klikabilnom, ne samo kružić.

---

#### GUMB ZA SLANJE

```
.submit-btn {  
    margin-top: 25px;  
    width: 100%;
```

```

padding: 12px;
background-color: #4aa3ff;
border: none;
border-radius: 10px;
color: white;
font-size: 16px;
font-weight: bold;
cursor: pointer;
transition: background 0.3s ease;
}

.submit-btn:hover {
  background-color: #3a91e0;
}

```

- Gumb je širok koliko i forma width: 100%.
- Boja #4aa3ff prati plavu temu.
- hover tamnija nijansa #3a91e0 daje efekt “aktivnog” gumba.
- transition osigurava glatki prijelaz pri prelasku mišem kao i prije.
- Vizualno zatvara formu i jasno označava akciju slanja.

Dio CSS koda u ovom repozitoriju je također unaprijeden s pomoću ChatGPT (OpenAI). Forma je generirana putem umjetne inteligencije te kasnije nadograđena. Razlog korištenja umjetne inteligencije je lakši razvoj sučelja za unos podataka po našim željenim parametrima te povezani CSS stil. Cijeli razgovor možete pronaći na poveznici <https://chatgpt.com/share/690b88c9-06d0-8012-97df-ba1e3d51edc1>. AI mi je stvorio osnovnu stranicu sa povezanim CSS-om koja je kasnije ručno nadograđena. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-11-5 **Srđa:** Izrada osnovne stranice te CSS-a.

## KOMPONENTA RENTALUNITS.JSX

Komponenta RentalUnits prikazuje popis svih smještajnih jedinica Apartmani / Sobe koje su spremljene u backend bazi podataka. Korisniku omogućuje pregled svih jedinica dohvaćenih iz API-ja, uređivanje postojeće jedinice, brisanje jedinice, dodavanje nove jedinice.

---

### UVOZ MODULA I POČETNA POSTAVKA

```

import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "./RentalUnits.css";

```

- useState koristi se za pohranu i ažuriranje popisa jedinica (units).
- useEffect izvršava se kod učitavanja komponente, i koristi se za dohvatanje podataka iz backend API-ja.
- useNavigate hook iz react-router-dom biblioteke koji omogućuje navigaciju između ruta.
- RentalUnits.css povezan CSS dokument.

---

## DEFINICIJA KOMPONENTE

```
const RentalUnits = () => {
  const [units, setUnits] = useState([]);
  const navigate = useNavigate();
```

- units niz koji sadrži sve smještajne jedinice.
- setUnits funkcija za ažuriranje stanja.
- navigate koristi se za preusmjeravanje korisnika na druge rute (npr. /form ili /form/:id).

---

## UREĐIVANJE POSTOJEĆE JEDINICE

```
const handleEdit = (id) => {
  navigate(`/form/${id}`);
};
```

- Kada korisnik klikne gumb “Edit”, funkcija handleEdit koristi navigate za preusmjeravanje na URL /form/:id.
- Taj URL otvara ApartmentForm komponentu, koja prema ID-u dohvaća podatke o konkretnoj jedinici i omogućuje uređivanje.

---

## BRISANJE JEDINICE

```
const handleDelete = async (id) => {
  if (window.confirm("Are you sure you want to delete this unit?")) {
    try {
      const response = await fetch(`http://localhost:8080/unit/delete/${id}`,
{
      method: "DELETE",
    });
      if (response.ok) {
        setUnits((prev) => prev.filter((unit) => unit.id !== id));
      } else {
        alert("Failed to delete unit.");
      }
    } catch (err) {
      console.error("Error deleting unit:", err);
    }
  }
};
```

- Pita korisnika za potvrdu (window.confirm()).
- Ako korisnik potvrdi, šalje DELETE zahtjev na backend (/unit/delete/{id}).
- Ako je odgovor pozitivan response.ok, jedinica se uklanja iz lokalnog stanja pomoću: **setUnits((prev) => prev.filter((unit) => unit.id !== id));**; čime se odmah ažurira prikaz bez ponovnog učitavanja stranice.

- Ako nije uspješno, prikazuje se alert poruka o pogrešci.

## DODAVANJE NOVE JEDINICE

```
const handleCreate = () => {
  navigate("/form");
};
```

Kada korisnik klikne gumb “+ Create New Unit”, otvara se forma bez ID-a. Budući da ApartmentForm provjerava postoji li id u URL-u, zna da je riječ o novom unosu.

## RENDER STRANICE ADMIN

```
return (
  <div className="rental-units-container">
    <h1 className="title">Rental Units</h1>
    <ul className="units-list">
      {units.map((unit) => (
        <li key={unit.id} className="unit-item">
          <div className="unit-info">
            <span className="unit-name">{unit.name}</span>
            <span className="unit-type">({unit.type})</span>
          </div>
          <div className="unit-actions">
            <button className="edit-button" onClick={() => handleEdit(unit.id)}>
              Edit
            </button>
            <button className="delete-button" onClick={() => handleDelete(unit.id)}>
              Delete
            </button>
          </div>
        </li>
      )))
    </ul>

    <button className="create-button" onClick={handleCreate}>
      + Create New Unit
    </button>
  </div>
);
```

- Glavni kontejner: .rental-units-container sadrži sve elemente prikaza.
- Naslov: <h1> označava naslov sekcije.

- Lista jedinica: `<ul className="units-list">`
- Svaka jedinica (`<li>`) prikazuje naziv i tip Apartment/Room.
- Edit preusmjerava korisnika na /form/:id
- Delete briše jedinicu nakon potvrde
- Na dnu se nalazi gumb za dodavanje nove jedinice.

## RENTALUNITS.CSS

Ovaj CSS određuje izgled stranice koja prikazuje listu smještajnih jedinica . Cilj dizajna je jednostavan s naglaskom na čitljivost i intuitivno korištenje.

---

### GLAVNI KONTEJNER

```
.rental-units-container {
  max-width: 600px;
  margin: 40px auto;
  padding: 20px;
  background-color: #fafafa;
  border-radius: 12px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}
```

- max-width: 600px; Ograničava širinu sadržaja kako ne bi bio preširok.
- margin: 40px auto; Centriranje kontejnera na sredinu stranice s razmakom od vrha.
- padding: 20px; Unutarnji razmak .
- background-color: #fafafa; Svetlosiva pozadina .
- border-radius: 12px; Blago zaobljeni rubovi .
- box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1); Lagana sjena stvara osjećaj dubine i odvojenosti od pozadine. ### Naslov stranice

```
.title {
  text-align: center;
  margin-bottom: 20px;
  color: #333;
}
```

- text-align: center; Poravnava naslov “Rental Units” na sredinu.
- margin-bottom: 20px; Odvaja naslov od ostatka sadržaja.
- color: #333; Tamnosiva boja za dobar kontrast bez oštine crne (#000).

---

### LISTA JEDINICA

```
.units-list {
  list-style: none;
  padding: 0;
```

```
    margin: 0;  
}
```

- list-style: none; Uklanja točkice (•) koje su standardne u HTML listama.
- padding i margin: 0; Uklanja zadane razmake koje preglednici dodaju elementima. ### Jedna jedinica

```
.unit-item {  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
    padding: 12px;  
    margin-bottom: 10px;  
    background-color: #fff;  
    border-radius: 8px;  
    border: 1px solid #ddd;  
}
```

- display: flex; Omogućuje da se podaci i gumbi prikažu u jednom redu.
- justify-content: space-between; Lijevo su informacije, desno akcijski gumbi.
- align-items: center; Vertikalno centririra sadržaj.
- padding: 12px; Dodaje unutarnji razmak unutar svake kartice.
- margin-bottom: 10px; Stvara razmak između kartica.
- background-color: #fff; Bijela pozadina.
- border-radius: 8px; Zaobljeni rubovi.
- border: 1px solid #ddd; Diskretni obrub pomaže u vizualnom razdvajaju elemenata.

---

## INFORMACIJE O JEDINICI

```
.unit-info {  
    display: flex;  
    flex-direction: column;  
}
```

- display: flex; flex-direction: column; Postavlja naziv i tip jedinice jedan ispod drugog.

---

## NAZIV JEDINICE

```
.unit-name {  
    font-weight: bold;  
}
```

- font-weight: bold; Podebljava naziv kako bi se istaknuo kao glavni podatak.

---

## TIP JEDINICE

```
.unit-type {  
    color: #666;  
    font-size: 0.9em;  
}
```

- color: #666; Svjetlija siva boja .
- font-size: 0.9em; Blago smanjen font .

---

#### AKCIJSKI GUMBI UREDI / OBRIŠI

```
.unit-actions button {  
    margin-left: 8px;  
    padding: 6px 12px;  
    border: none;  
    border-radius: 6px;  
    cursor: pointer;  
}
```

- margin-left: 8px; Razmak između gumba “Edit” i “Delete”.
- padding: 6px 12px; Udoban razmak unutar gumba za lakše klikanje.
- border: none; Uklanja standardni rub gumba.
- border-radius: 6px; Blago zaobljeni rubovi gumba.
- cursor: pointer; Pokazivač miša se promijeni da se može kliknut.

#### GUMB “EDIT”

---

```
.edit-button {  
    background-color: #007bff;  
    color: white;  
}
```

- background-color: #007bff; plava boja za akcije tipa “uredi”.
- color: white; Bijeli tekst .

#### GUMB “DELETE”

---

```
.delete-button {  
    background-color: #dc3545;  
    color: white;  
}
```

- background-color: #dc3545; Crvena boja koja upozorava da je riječ o brisanju.
- color: white; bijela boja.

#### GUMB “CREATE NEW UNIT”

---

```
.create-button {  
    display: block;  
    margin: 20px auto 0;  
    padding: 10px 20px;  
    background-color: #28a745;  
    color: white;  
    border: none;  
    border-radius: 8px;  
    cursor: pointer;  
}  
}
```

- display: block; margin: 20px auto 0; Gumb je centriran ispod liste jedinica.
- padding: 10px 20px; Dovoljno velik .
- background-color: #28a745; Zelena boja označava akciju (dodaj / kreiraj).
- border-radius: 8px; U skladu s ostatkom dizajna.
- cursor: pointer; Pokazuje da je gumb aktivan.

#### HOVER EFEKT

---

```
.create-button:hover {  
    background-color: #218838;  
}
```

Kad korisnik prijede mišem preko gumba, zelena postaje malo tamnija (#218838). Daje vizualni feedback korisniku.

Dio CSS koda u ovom repozitoriju je također unaprijeđen s pomoću ChatGPT (OpenAI). Uput korišten za dobivanje komponente jsx-a te CSS-a je dobiven upitom koj se nalazi na poveznici <https://chatgpt.com/share/690b9a50-fe54-8012-b0fb-96ef7c2b96c6>. AI mi je stvorio osnovnu stranicu sa povezanim CSS-om koja je kasnije ručno nadograđena i unaprijeđena. **Alat:** ChatGPT (OpenAI)

**Datum pristupa:** 2025-11-5 **Svrha:** Izrada osnovne stranice te CSS-a.

## TIJEK IMPLEMENTIRANJA FULLSTACK

### DODAVANJE CRUD OPERACIJA ZA SMJEŠTAJNE JEDINICE

#### POVEZIVANJE SUBMITA I BACKENDA/BAZE

Većina operacija za dodavanje u bazu je već implementirana i opisana. Trebalo je samo u formi (form.jsx u frontendu) za submit ispravno slati backendu podatke. Payload napravljen na temelju baze podataka i tablice unit. (“id\_unit” “cap\_adults” “cap\_children” “has\_air\_conditioning” “has\_breakfast” “has\_hair\_dryer” “has\_heater” “has\_parking” “has\_shampoo” “has\_towels” “has\_wifi” “is\_apartment” “location” “main\_desc\_content” “main\_desc\_name” “num\_beds” “num\_rooms” “price” “rating” “sec\_desc\_content” “sec\_desc\_name” “unit\_name”) Nakon šaljemo post upit na već postavljen /unit/add s podacima iz forme. Alertom testiramo je li uspješno ili nije. Na kraju refreshamo formu da se može upisati nova smještajna jedinica.

```
const handleSubmit = async (e) => {
  e.preventDefault();

  const unitPayload = {
    unitName: formData.unitName,
    mainDescName: formData.mainDescriptionTitle,
    mainDescContent: formData.mainDescription,
    secDescName: formData.secondaryDescriptionTitle,
    secDescContent: formData.secondaryDescription,
    price: parseInt(formData.price),
    numRooms: 1,
    capAdults: 2,
    capChildren: 0,
    numBeds: 1,
    hasParking: formData.amenities.parking,
    hasWifi: formData.amenities.wifi,
    hasBreakfast: formData.amenities.breakfast,
    hasTowels: formData.amenities.towels,
    hasShampoo: formData.amenities.shampoo,
    hasHairDryer: formData.amenities.hairDryer,
    hasHeater: formData.amenities.heater,
    hasAirConditioning: formData.amenities.airConditioning,
    isApartment: true,
    location: "Zagreb, Croatia", // fixat da povlači lokaciju od lokacije hotela da nije hard kodirano
    rating: 0 // fixat kasnije da bude rating sobe na temelju recenzija
  };

  try {
    const response = await fetch("http://localhost:8080/unit/add", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(unitPayload),
    });

    if (response.ok) {
      alert("Unit added successfully!");
      setFormData({
        unitName: "",
        mainDescriptionTitle: "",
        mainDescription: "",
        secondaryDescriptionTitle: "",
        secondaryDescription: "",
        price: "",
      });
    }
  } catch (error) {
    console.error(error);
  }
};
```

```

        amenities: {
            parking: false,
            wifi: false,
            breakfast: false,
            towels: false,
            shampoo: false,
            hairDryer: false,
            heater: false,
            airConditioning: false,
        },
    });
} else {
    const errorText = await response.text();
    alert("Error: " + errorText);
}
} catch (error) {
    console.error("Error submitting form:", error);
    alert("Something went wrong!");
}
};


```

## POVEZIVANJE FRONTEND PRIKAZA SMJEŠTAJNIH JEDINICA S BAZOM

---

### FORM.JSX

Form nam nije imao opciju za označavanje jel jedinica apartman ili nije pa sam dodao checkbox da se označi Apartment ili Room. Bilo je problema i sa slanjem true false za isApartment to je isto popravljeno. Sad nas submitanjem forme vraća na listu svih jedinica gdje se prikaže naša nova jedinica.

useEffect metodu u form.jsx koristimo kasnije za popunjavanje forme s podacima iz baze tokom editanja postojećih jedinica. Poziva funkciju s backenda za izvlačenje podataka iz baze i zapisuje ih u formu.

```

import React, { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router-dom";
import "./ApartmentForm.css";

const ApartmentForm = () => {
    const { id } = useParams();
    const navigate = useNavigate();

    const [formData, setFormData] = useState({
        unitName: "",
        mainDescriptionTitle: "",
        mainDescription: "",
        secondaryDescriptionTitle: "",

```

```
secondaryDescription: "",  
price: "",  
isApartment: true,  
amenities: {  
  parking: false,  
  wifi: false,  
  breakfast: false,  
  towels: false,  
  shampoo: false,  
  hairDryer: false,  
  heater: false,  
  airConditioning: false,  
},  
});  
  
const [images, setImages] = useState([]);  
  
useEffect(() => {  
  if (id) {  
    fetch(`http://localhost:8080/unit/${id}`)  
      .then((res) => res.json())  
      .then((data) => {  
        setFormData({  
          unitName: data.unitName || "",  
          mainDescriptionTitle: data.mainDescName || "",  
          mainDescription: data.mainDescContent || "",  
          secondaryDescriptionTitle: data.secDescName || "",  
          secondaryDescription: data.secDescContent || "",  
          price: data.price || "",  
          isApartment: data.isApartment ?? true,  
          amenities: {  
            parking: data.hasParking || false,  
            wifi: data.hasWifi || false,  
            breakfast: data.hasBreakfast || false,  
            towels: data.hasTowels || false,  
            shampoo: data.hasShampoo || false,  
            hairDryer: data.hasHairDryer || false,  
            heater: data.hasHeater || false,  
            airConditioning: data.hasAirConditioning || false,  
          },  
        });  
      })  
      .catch((err) => console.error("Error fetching unit:", err));  
  }  
})
```

```
        }
    }, [id]);

const handleChange = (e) => {
  const { name, value, type, checked } = e.target;
  if (name in formData.amenities) {
    setFormData({
      ...formData,
      amenities: { ...formData.amenities, [name]: checked },
    });
  } else {
    setFormData({ ...formData, [name]: value });
  }
};

const handleImageUpload = (e) => {
  const files = Array.from(e.target.files);
  const newImages = files.map((file) => ({
    file,
    url: URL.createObjectURL(file),
  }));
  setImages((prev) => [...prev, ...newImages]);
};

const removeImage = (index) => {
  setImages((prev) => prev.filter((_, i) => i !== index));
};

const handleSubmit = async (e) => {
  e.preventDefault();

  const unitPayload = {
    unitName: formData.unitName,
    mainDescName: formData.mainDescriptionTitle,
    mainDescContent: formData.mainDescription,
    secDescName: formData.secondaryDescriptionTitle,
    secDescContent: formData.secondaryDescription,
    price: parseInt(formData.price),
    numRooms: 1,
    capAdults: 2,
    capChildren: 0,
    numBeds: 1,
    hasParking: formData.amenities.parking,
    hasWifi: formData.amenities.wifi,
  };
}
```

```
        hasBreakfast: formData.amenities.breakfast,
        hasTowels: formData.amenities.towels,
        hasShampoo: formData.amenities.shampoo,
        hasHairDryer: formData.amenities.hairDryer,
        hasHeater: formData.amenities.heater,
        hasAirConditioning: formData.amenities.airConditioning,
        isApartment: formData.isApartment,
        location: "Zagreb, Croatia",
        rating: 0,
    };

try {
    const response = await fetch("http://localhost:8080/unit/add", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(unitPayload),
    });

    if (response.ok) {
        alert("Unit added successfully!");
        navigate("/admin");
    } else {
        const errorText = await response.text();
        alert("Error: " + errorText);
    }
} catch (error) {
    console.error("Error submitting form:", error);
    alert("Something went wrong!");
}
};

return (
    <div className="form-container">
        <h2>{id ? `Edit Unit ${id}` : "Create New Unit"}</h2>
        <form onSubmit={handleSubmit} className="apartment-form">
            <label>Unit Name</label>
            <input
                type="text"
                name="unitName"
                value={formData.unitName}
                onChange={handleChange}
                required
            />
        </form>
    </div>
);
```

```
{}
<label>Unit Type</label>
<div className="radio-group">
  <label>
    <input
      type="radio"
      name="isApartment"
      checked={formData.isApartment === true}
      onChange={() => setFormData({ ...formData, isApartment: true })}
    />
    Apartment
  </label>
  <label>
    <input
      type="radio"
      name="isApartment"
      checked={formData.isApartment === false}
      onChange={() => setFormData({ ...formData, isApartment: false
})}>
    />
    Room
  </label>
</div>

<label>Main Description Title</label>
<input
  type="text"
  name="mainDescriptionTitle"
  value={formData.mainDescriptionTitle}
  onChange={handleChange}
/>

<label>Main Description</label>
<textarea
  name="mainDescription"
  rows="3"
  value={formData.mainDescription}
  onChange={handleChange}
/>

<label>Secondary Description Title</label>
<input
  type="text"
```

```
        name="secondaryDescriptionTitle"
        value={formData.secondaryDescriptionTitle}
        onChange={handleChange}
    />

    <label>Secondary Description</label>
    <textarea
        name="secondaryDescription"
        rows="3"
        value={formData.secondaryDescription}
        onChange={handleChange}
    />

    <label>Price (€)</label>
    <input
        type="number"
        name="price"
        value={formData.price}
        onChange={handleChange}
    />

    <div className="image-upload">
        <label>Images</label>
        <input type="file" multiple accept="image/*" onChange={handleImageUpload} />
        <div className="image-preview">
            {images.map((img, index) => (
                <div key={index} className="preview-item">
                    <img src={img.url} alt={`Image ${index + 1}`} />
                    <button type="button" onClick={() => removeImage(index)}>
                        Remove
                    </button>
                </div>
            )))
        </div>
    </div>

    <div className="checkbox-section">
        <h4>Amenities</h4>
        {Object.keys(formData.amenities).map(option => (
            <label key={option} className="checkbox-label">
                <input
                    type="checkbox"
                    name={option}

```

```

        checked={formData.amenities[option]}
        onChange={handleChange}
    />
    {option.replace(/([A-Z])/g, " $1").replace(/\./, (str) => str.t
oUpperCase())}
    </label>
  ))}
</div>

<button type="submit" className="submit-btn">
  {id ? "Update" : "Submit"}
</button>
</form>
</div>
);
};

export default ApartmentForm;

```

---

#### RENTALUNITS.JSX

Trebalo je promjeniti RentalUnits.jsx kako bi mogao koristiti podatke iz već postojeće funkcije iz baze/backenda. (/unit/all koji vraća json svih podataka iz baze) Dodana je parcijalna funkcionalnost za delete button ali nije još u funkcionalnosti. Treba izmjeniti malo backend funkciju

useEffect metoda povlači kao i u form.jsx s backenda sve podatke iz baze te prikazuje one potrebne na stranici kako bi se vidjela situacija spremljenih jedinica i kako bi ih se moglo brisati i editati.

```

import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "./RentalUnits.css";

const RentalUnits = () => {
  const [units, setUnits] = useState([]);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchUnits = async () => {
      try {
        const response = await fetch("http://localhost:8080/unit/all");
        if (!response.ok) throw new Error("Failed to fetch units");
        const data = await response.json();
      }
    }
  });
}

export default RentalUnits;

```

```
        const formatted = data.map((u) => ({
          id: u.id,
          name: u.unitName,
          type: u.isApartment ? "Apartment" : "Room",
        }));
      
```

```
      setUnits(formatted);
    } catch (err) {
      console.error("Error fetching units:", err);
    }
  };

```

```
  fetchUnits();
}, []);
```

```
const handleEdit = (id) => {
  navigate(`/form/${id}`);
};
```

```
const handleDelete = async (id) => {
  if (window.confirm("Are you sure you want to delete this unit?")) {
    try {
      const response = await fetch(`http://localhost:8080/unit/delete/${id}`)
    , {
        method: "DELETE",
      });
      if (response.ok) {
        setUnits((prev) => prev.filter((unit) => unit.id !== id));
      } else {
        alert("Failed to delete unit.");
      }
    } catch (err) {
      console.error("Error deleting unit:", err);
    }
  }
};
```

```
const handleCreate = () => {
  navigate("/form");
};
```

```
return (
  <div className="rental-units-container">
    <h1 className="title">Rental Units</h1>
```

```

        <ul className="units-list">
          {units.map((unit) =>
            <li key={unit.id} className="unit-item">
              <div className="unit-info">
                <span className="unit-name">{unit.name}</span>
                <span className="unit-type">({unit.type})</span> {}
              </div>
              <div className="unit-actions">
                <button className="edit-button" onClick={() => handleEdit(unit.id)}>
                  Edit
                </button>
                <button className="delete-button" onClick={() => handleDelete(unit.id)}>
                  Delete
                </button>
              </div>
            </li>
          ))}
        </ul>

        <button className="create-button" onClick={handleCreate}>
          + Create New Unit
        </button>
      </div>
    );
  };

export default RentalUnits;

```

Na kraju je trebalo još dodati css za novi način prikazivanja.

```

.radio-group {
  display: flex;
  gap: 1rem;
  margin-bottom: 1rem;
}

.radio-group label {
  display: flex;
  align-items: center;
  gap: 0.3rem;
  cursor: pointer;
}

```

---

## UNIT.JAVA

Ovdje je trebalo dodati import...

```
import com.fasterxml.jackson.annotation.JsonProperty;
```

...i mijenjati ove linije...

```
@Column(nullable = false)
private boolean isApartment; //ako je true onda je jedinica apartman, u suprotnom je soba
```

...u ovo. (Možda ovo nije bilo niti potrebno, ali sad funkcionira pa neka ostane.)

```
@Column(name = "is_apartment", nullable = false)
@JsonProperty("isApartment")
private boolean apartment;
```

## DELETE FUNKCIONALNOST NA LISTI JEDINICA

Trebalo je samo u RentalUnits.jsx promjeniti...

```
id: u.id,
```

Ovo u...

```
id: u.idUnit,
```

Sad radi delete button i briše se iz baze.

## UPDATE FUNKCIONALNOST ZA JEDINICE

---

## UNITCONTROLLER.JAVA

Trebalo je napraviti novu funkciju za update već postojećih jedinica u tablici. Nadodajemo na kraj ovo.

```
@PutMapping("/update/{id}")
public ResponseEntity<?> updateUnit(@PathVariable Long id, @RequestBody Unit updatedUnit) {
    return repo.findById(id).map(existingUnit -> {
        updatedUnit.setIdUnit(id);
        repo.save(updatedUnit);
        return ResponseEntity.ok("Unit updated successfully");
    }).orElseGet(() -> ResponseEntity.notFound().build());
}
```

---

## FORM.JSX

Mijenjamo način na koji frontend submita formu

Dodajemo ovo prije try operacije u handleSubmit funkciji. Ovime ovisno imamo li id ili ne šaljemo submit na funkciju dodavanja ili updateanja koju smo malo prije nadodali.

```
const url = id
  ? `http://localhost:8080/unit/update/${id}`
  : "http://localhost:8080/unit/add";
const method = id ? "PUT" : "POST";
```

U try bloku imamo...

```
const response = await fetch("http://localhost:8080/unit/add", {
  method: "POST",
```

...što mijenjamo s ovime. Sada će fetchat metodu i url koji smo prije odredili ovisno o tome imamo li ili ne ID jedinice.

```
const response = await fetch(url, {
  method,
```

Također mijenjamo alert da znamo da smo updateali, a ne dodali jedinicu. Ovo..

```
  alert("Unit added successfully!");
```

...u ovo.

```
  alert(id ? "Unit updated successfully!" : "Unit added successfully!");
```

**Autor: Josip Mrakovčić Datum završetka: 5.11.2025. Utrošeno vrijeme: ~8 sati # Fixovi ## Admin profil**

Imali smo problem da svi ulogirani i registrirani profili automatski dobivaju admin privilegije. Nije bilo buttona za redirect na /admin page.

---

#### PERSONCONTROLLER.JAVA

Promjenjen je kod kako bi se točnije spremale varijable kod dodavanja novog usera. Zamijenili smo ovu liniju...

```
Person person = new Person(email, true, false, false, name);
```

...ovim linijama.

```
Person person = new Person();
person.setEmail(email);
person.setName(name);
```

```
boolean isFirstUser = repo.count() == 0;
```

```
person.setAdmin(isFirstUser);
```

```
person.setOwner(false);
person.setUser(true);
```

Na kraju funkcije sam promjenio response da vidim ako mi se admin uspješno stvorio.

```
return ResponseEntity.ok("User added successfully" + (isFirstUser ? " (as admin)" : ""));
```

Nadodana je funkcija koja vraća trenutno ulogiranog korisnika /me. Potrebno za validaciju tokena i pristup funkcijama vezanih uz admina.

```
@GetMapping("/me")
public ResponseEntity<?> getCurrentUser(@AuthenticationPrincipal Jwt jwt)
{
    String email = jwt.getClaimAsString("email");
    if (email == null) {
        return ResponseEntity.badRequest().body("Invalid token");
    }

    Optional<Person> person = repo.findByEmail(email);
    if (person.isEmpty()) {
        return ResponseEntity.status(404).body("User not found");
    }

    return ResponseEntity.ok(person.get());
}
```

---

#### PERSON.JAVA

Nadodane linije kako bi se osiguralo da kod povratka JSON-a iz baze sigurno dobijemo uloge profila.

```
import com.fasterxml.jackson.annotation.JsonProperty; // kod importova na dodano

@JsonProperty("is_admin") // NADODANO
@Column(nullable = false)
private boolean isAdmin;

@JsonProperty("is_user") // NADODANO
@Column(nullable = false)
private boolean isUser;

@JsonProperty("is_owner") // NADODANO
@Column(nullable = false)
private boolean isOwner;
```

---

## APP.JS

Client ID za OAuth2 je promjenjen na novi.

---

## NAVBAR.JSX

U logoutu sa stranice je dodan refresh za stranicu kako bi se maknuo button za admine koji se odlogiraju.

```
localStorage.removeItem("googleUser");
window.location.reload(); // NADODANO
```

Nadodana kopija dohvata podatka jer je postojao bug gdje postojeći user u bazi bi davao error i ne bi se dohvaćali njegovi podaci nit se ulogirao.

```
const { data: userFromDB } = await axios.get(
  `${process.env.REACT_APP_API_URL}/me`,
  { headers: { Authorization: `Bearer ${idToken}` } }
);
```

Spremamo u browser kombinaciju tokena i podataka iz baze za kasniju obradu. Također refreshamo page ako je admin user da mu se pokaže admin button. Zamijenjene ove linije koda...

```
console.log("Decoded user:", decoded);
setUser(decoded);
localStorage.setItem("googleUser", JSON.stringify(decoded));
```

...s ovima.

```
const finalUser = { ...decoded, ...userFromDB };

setUser(finalUser);
localStorage.setItem("googleUser", JSON.stringify(finalUser));

// Refresh da se prikaže admin gumb
window.location.reload();
```

---

## HEADER.JSX

Nadodane linije nakon importova kako bi se povukli trenutni podaci usera koji je ulogiran.

```
const [user, setUser] = useState(() => {
  const savedUser = localStorage.getItem("googleUser");
  return savedUser ? JSON.parse(savedUser) : null;
});
```

Promjenjen je button Sign in/Register...

```
<button className="headerBTN">Sign in/Register</button>
```

...u button za redirect na admin stranicu.

```
{user?.is_admin && (
  <button
    className="headerBTN"
    onClick={() => navigate("/admin")}
  >
    Admin Page
  </button>
)}
```

## UREĐIVANJE /ADMIN PAGEA (SITNI FIXOVI)

Admin page nam nije imao button za povratak na main page. Form page nam nije imao cancel button ako se predomisli kod unosa novih ili uređivanja postojećih jedinica. Form nam nije imao neke atribute koji su u bazi podataka poput numRooms, capChild, capAdult. Zabraniti gostima da uđu na admin i form page. ### RentalUnits.jsx Provjeravamo ako je user admin tako što dodajemo ove linije u useEffect metodu.

```
const savedUser = localStorage.getItem("googleUser");
const user = savedUser ? JSON.parse(savedUser) : null;

// Ako nije admin redirectaj ga na main page
if (!user || !user.is_admin) {
  navigate("/main");
  return;
}
```

Dodan handler za povratak na main page.

```
const handleBackToMain = () => {
  navigate("/main");
};
```

Zamijenjena ova linija...

```
<h1 className="title">Rental Units</h1>
```

...s ovima kako bismo dodali button.

```
<div className="header-row">
  <h1 className="title">Rental Units</h1>
  <button className="back-button" onClick={handleBackToMain}>
    ← Back to Main Page
  </button>
</div>
```

---

RENTALUNITS.CSS

Dodajemo dizajn dodanom buttonu.

```
.back-button {  
    background-color: #007bff;  
    color: white;  
    border: none;  
    padding: 8px 14px;  
    border-radius: 6px;  
    cursor: pointer;  
    font-weight: 500;  
    transition: 0.2s ease-in-out;  
    margin-left: 10px;  
}  
  
.back-button:hover {  
    background-color: #0056b3;  
}  
  
.header-row {  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```

---

## FORM.JSX

Treba nadodati Cancel button i opcije za unos kreveta, prostorija itd. U formi za input podataka dodajemo.

```
capAdults: 2,  
capChildren: 0,  
numRooms: 1,  
numBeds: 1,
```

U useEffect dodajemo zabranu pristupa neadminima. Isto ko na /admin pageu.

```
const savedUser = localStorage.getItem("googleUser");  
const user = savedUser ? JSON.parse(savedUser) : null;  
  
if (!user || !user.is_admin) {  
    navigate("/main");  
    return;  
}
```

U fetchu podataka iz baze kojom se popunjavaju podaci kod edita jedinice dodajemo.

```
capAdults: data.capAdults || 2,  
capChildren: data.capChildren || 0,  
numRooms: data.numRooms || 1,  
numBeds: data.numBeds || 1,
```

U unitPayloadu mjenjamo...

```
numRooms: 1,  
capAdults: 2,  
capChildren: 0,  
numBeds: 1,
```

...s ovime. Također stavljamo zabranu slanja više od 1 sobe ako je type jedinice room.

```
capAdults: parseInt(formData.capAdults),  
capChildren: parseInt(formData.capChildren),  
numRooms: formData.isApartment ? parseInt(formData.numRooms) : 1,  
numBeds: parseInt(formData.numBeds),
```

Dodajemo handler za cancel u formi.

```
const handleCancel = () => {  
  if (window.confirm("Are you sure you want to cancel? Changes will not be  
  saved.")) {  
    navigate("/admin");  
  }  
};
```

Mjenjana je struktura stranice. Nadodajemo da se numRooms input prikaže ako je apartment type izabran, stavljamo ga desno od odabira tipa jedinice kako se ne bi "shiftali" ostali inputi.

```
<div className="radio-group with-rooms">  
  <div className="radio-options">  
    <label>  
      <input  
        type="radio"  
        name="isApartment"  
        checked={formData.isApartment === true}  
        onChange={() => setFormData({ ...formData, isApartment: true  
      })}>  
      />  
      Apartment  
    </label>  
    <label>  
      <input  
        type="radio"  
        name="isApartment"
```

```
        checked={formData.isApartment === false}
        onChange={() => setFormData({ ...formData, isApartment: false
})}
    />
    Room
    </label>
</div>

{formData.isApartment && (
    <div className="num-rooms-inline">
        <label>Rooms:</label>
        <input
            type="number"
            name="numRooms"
            value={formData.numRooms}
            onChange={handleChange}
            min="1"
        />
    </div>
)
}
</div>

<label>Capacity (Adults)</label>
<input
    type="number"
    name="capAdults"
    value={formData.capAdults}
    onChange={handleChange}
    min="1"
/>

<label>Capacity (Children)</label>
<input
    type="number"
    name="capChildren"
    value={formData.capChildren}
    onChange={handleChange}
    min="0"
/>

<label>Number of Beds</label>
<input
    type="number"
    name="numBeds"
```

```
        value={formData.numBeds}
        onChange={handleChange}
        min="1"
    />
```

Na kraj prije kraja, prije

nadodajemo linije za cancel button.

```
<div className="button-row">
    <button type="submit" className="submit-btn">
        {id ? "Update" : "Submit"}
    </button>
    <button
        type="button"
        className="cancel-btn"
        onClick={handleCancel}
    >
        Cancel
    </button>
</div>
```

---

#### APARTMENTFORM.CSS

Mjenamo dizajn form stranice kako bi imali usklađen dizajn novih buttona, inputa sa starima. Mjenamo neke stare dizajne i dodajemo nove.

```
.button-row {
    display: flex;
    justify-content: space-between;
    gap: 12px;
    margin-top: 25px;
}

.submit-btn,
.cancel-btn {
    flex: 1;
    padding: 12px 0;
    height: 46px;
    border: none;
    border-radius: 10px;
    font-size: 16px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.2s ease;
    display: flex;
```

```

    align-items: center;
    justify-content: center;
    line-height: 1;
}

.submit-btn {
    background-color: #4aa3ff;
    color: white;
}

.submit-btn:hover {
    background-color: #3a91e0;
    transform: translateY(-1px);
}

.cancel-btn {
    background-color: #ff7f7f;
    color: white;
}

.cancel-btn:hover {
    background-color: #e86a6a;
    transform: translateY(-1px);
}

```

**Autor: Josip Mrakovčić Datum završetka: 8.11.2025. Utrošeno vrijeme: ~6 sati**

## USER LISTA NA ADMIN PAGEU

PODIJELITI ADMIN PAGE NA UNIT I USERS TABOVE

## RENTALUNITS.JSX

Mjenjanje početka glavne metode RentalUnits u...

```

const AdminDashboard = () => {
    const [activeTab, setActiveTab] = useState("units");
    const [units, setUnits] = useState([]);
    const [users, setUsers] = useState([]);
    const [currentUser, setCurrentUser] = useState(null);
    const navigate = useNavigate();

```

Na kraj useEffect metode dodajemo. Ovime postavljamo trenutnog usera kako bismo kasnije mogli brisati druge korisnike i zabraniti useru da briše sam sebe. Također povlačimo sve jedinice i usere.

```
setCurrentUser(user);

fetchUnits();
fetchUsers();
}, [navigate]);
```

Dodana metoda fetchanja usera iz baze podataka kako bi se podaci prikazali u listi.

```
const fetchUsers = async () => {
  try {
    const response = await fetch(` ${process.env.REACT_APP_API_URL}/allPersons`);
    if (!response.ok) throw new Error("Failed to fetch users");
    const data = await response.json();
    setUsers(
      data.map((u) => {
        let role = "User";
        if (u.is_admin) role = "Admin";
        else if (u.is_owner) role = "Owner";
        return {
          id: u.id,
          name: u.name,
          email: u.email,
          role,
        };
      })
    );
  } catch (err) {
    console.error("Error fetching users:", err);
  }
}
```

Dodana provjera i slanje tokena kod brisanja unita.

```
const token = localStorage.getItem("access_token");
const response = await fetch(` ${process.env.REACT_APP_API_URL}/deletePerson/${id}`, {
  method: "DELETE",
  headers: {
    Authorization: `Bearer ${token}`,
  },
});
```

Dodajemo metodu brisanja korisnika. Pazimo da korisnik ne može obrisati sam sebe te šaljemo token usera da provjerimo je li admin kad pokuša obrisati neki profil.

```
const handleDeleteUser = async (id, email) => {
  if (email === currentUser?.email) {
```

```

        alert("You cannot delete yourself!");
        return;
    }
    if (window.confirm(`Are you sure you want to delete user ${email}?`)) {
        try {
            const token = localStorage.getItem("access_token");

            const response = await fetch(`${process.env.REACT_APP_API_URL}/deletePerson/${id}`, {
                method: "DELETE",
                headers: {
                    Authorization: `Bearer ${token}`,
                },
            });

            if (response.ok) {
                setUsers((prev) => prev.filter((user) => user.id !== id));
            } else if (response.status === 403) {
                alert("You are not allowed to delete this user!");
            } else {
                alert("Failed to delete user.");
            }
        } catch (err) {
            console.error("Error deleting user:", err);
        }
    }
};


```

Na kraju mjenjamo return za admin dashboard jer smo morali mjenjati struktura zbog user taba. U kratko odvajamo user tab i unit tab, slažemo prijašnje buttone i na kraju mjenjamo export name jer više nisu samo rentalunits na tom pageu.

```

return (
    <div className="admin-dashboard">
        <aside className="sidebar">
            <h2>Admin Panel</h2>
            <button
                className={`tab-btn ${activeTab === "units" ? "active" : ""}`}
                onClick={() => setActiveTab("units")}
            >
                 Units
            </button>
            <button
                className={`tab-btn ${activeTab === "users" ? "active" : ""}`}
            >

```

```
    onClick={() => setActiveTab("users")}
  >
   Users
</button>
<hr />
<button className="back-main-btn" onClick={handleBackToMain}>
  ← Back to Main
</button>
</aside>

<main className="dashboard-content">
  {activeTab === "units" && (
    <section className="section-block">
      <h1>Rental Units</h1>
      <ul className="units-list">
        {units.map((unit) => (
          <li key={unit.id} className="unit-item">
            <div className="unit-info">
              <span className="unit-name">{unit.name}</span>
              <span className="unit-type">({unit.type})</span>
            </div>
            <div className="unit-actions">
              <button className="edit-button" onClick={() => handleEdit
(unit.id)}>
                Edit
              </button>
              <button className="delete-button" onClick={() => handleDe
leteUnit(unit.id)}>
                Delete
              </button>
            </div>
          </li>
        )))
      </ul>
      <button className="create-button" onClick={handleCreate}>
        + Create New Unit
      </button>
    </section>
  )}
  {activeTab === "users" && (
    <section className="section-block">
      <h1>Registered Users</h1>
```

```

        <table className="users-table">
          <thead>
            <tr>
              <th>Name</th>
              <th>Email</th>
              <th>Role</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {users.map((user) => (
              <tr key={user.id}>
                <td>{user.name}</td>
                <td>{user.email}</td>
                <td className={`role-${user.role.toLowerCase()}`}>{user.r
ole}</td>
                <td>
                  <button
                    className={`delete-button ${
                      user.email === currentUser?.email ? "disabled-btn" :
                      ""
                    }`}
                    onClick={() => handleDeleteUser(user.id, user.email)}
                    disabled={user.email === currentUser?.email}
                  >
                    Delete
                  </button>
                </td>
              </tr>
            )));
          </tbody>
        </table>
      </section>
    )}
  </main>
</div>
);
};

export default AdminDashboard;

```

---

#### RENTALUNITS.CSS

Opet smo morali mjenjati stil stranice kako bi se posložilo sve ispravno.

```
.admin-dashboard {
  display: flex;
  gap: 20px;
  margin: 40px auto;
  max-width: 1200px;
  padding: 20px;
}

.sidebar {
  width: 220px;
  background: #e8f4fd;
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0 2px 8px rgba(0, 100, 200, 0.1);
  display: flex;
  flex-direction: column;
  align-items: stretch;
}

.sidebar h2 {
  color: #025c9a;
  text-align: center;
  margin-bottom: 20px;
}

.tab-btn {
  padding: 10px 12px;
  margin-bottom: 10px;
  border: none;
  border-radius: 8px;
  background: #cfe7fc;
  color: #03426a;
  cursor: pointer;
  transition: background 0.3s ease;
  font-size: 15px;
  font-weight: 500;
}

.tab-btn.active {
  background-color: #4aa3ff;
  color: white;
}

.tab-btn:hover {
```

```
background-color: #b5d5f5;
}

.back-main-btn {
  margin-top: auto;
  padding: 10px 15px;
  background-color: #007bff;
  border: none;
  border-radius: 8px;
  color: white;
  font-weight: 600;
  cursor: pointer;
  transition: background 0.2s ease-in-out;
}

.back-main-btn:hover {
  background-color: #0056b3;
}

.dashboard-content {
  flex: 1;
  background: #fff;
  border-radius: 12px;
  box-shadow: 0 2px 8px rgba(0, 100, 200, 0.1);
  padding: 25px;
}

.section-block {
  background: #f9fcff;
  border-radius: 12px;
  padding: 20px;
  box-shadow: 0 2px 8px rgba(0, 100, 200, 0.05);
  margin-bottom: 20px;
}

.section-block h1 {
  color: #025c9a;
  margin-bottom: 15px;
  text-align: left;
}

.users-table {
  width: 100%;
  border-collapse: collapse;
```

```
    font-size: 15px;
}

.users-table th,
.users-table td {
    padding: 10px;
    border-bottom: 1px solid #cde4fa;
    text-align: left;
}

.users-table th {
    background-color: #e8f4fd;
    font-weight: 600;
    color: #025c9a;
}

.users-table tr:hover {
    background-color: #f0faff;
}

.role-admin {
    color: #e74c3c;
    font-weight: bold;
}

.role-owner {
    color: #f39c12;
    font-weight: 600;
}

.role-user {
    color: #2e8b57;
}

.user-actions {
    display: flex;
    justify-content: flex-end;
    gap: 8px;
}

.delete-user-btn {
    background-color: #dc3545;
    color: white;
    border: none;
```

```
padding: 6px 12px;
border-radius: 6px;
cursor: pointer;
font-size: 14px;
transition: background 0.2s ease-in-out;
}

.delete-user-btn:hover {
background-color: #b02a37;
}
.rental-units-container {
background: #f9fcff;
border-radius: 12px;
padding: 25px;
box-shadow: 0 2px 8px rgba(0, 100, 200, 0.05);
}

.rental-units-container .title {
color: #03426a;
text-align: left;
margin-bottom: 20px;
font-size: 22px;
}

.rental-units-container .units-list {
list-style: none;
padding: 0;
margin: 0;
display: flex;
flex-wrap: wrap;
gap: 12px;
}

.unit-item {
flex: 1 1 calc(50% - 12px);
display: flex;
justify-content: space-between;
align-items: center;
padding: 12px 14px;
background-color: #fff;
border: 1px solid #d5e6f8;
border-radius: 10px;
box-shadow: 0 1px 4px rgba(0, 100, 200, 0.05);
transition: transform 0.1s ease-in-out, box-shadow 0.2s ease-in-out;
```

```
}

.unit-item:hover {
  transform: translateY(-2px);
  box-shadow: 0 3px 8px rgba(0, 100, 200, 0.1);
}

.unit-info {
  display: flex;
  flex-direction: column;
}

.unit-name {
  font-weight: 600;
  color: #03426a;
}

.unit-type {
  color: #666;
  font-size: 0.9em;
}

.unit-actions button {
  margin-left: 8px;
  padding: 6px 12px;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  transition: background 0.2s ease;
}

.edit-button {
  background-color: #4aa3ff;
  color: white;
}

.edit-button:hover {
  background-color: #3a91e0;
}

.delete-button {
  background-color: #dc3545;
  color: white;
}
```

```
.delete-button:hover {
  background-color: #b02a37;
}

.create-button {
  margin-top: 20px;
  width: 100%;
  padding: 10px;
  background-color: #28a745;
  color: white;
  border: none;
  border-radius: 10px;
  font-weight: 600;
  cursor: pointer;
  transition: background 0.3s ease;
}

.create-button:hover {
  background-color: #218838;
}

.header-row {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

.back-button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 8px 14px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 500;
  transition: background 0.2s ease-in-out;
}

.back-button:hover {
  background-color: #0056b3;
}
```

```
.rental-units-container .units-list {
  justify-content: center;
}

.unit-item {
  flex: 0 1 280px;
  margin: 6px;
}

.create-button {
  display: block;
  margin: 25px auto 0 auto;
  width: 280px;
}

.disabled-btn {
  background-color: #ccc !important;
  color: #666 !important;
  cursor: not-allowed !important;
  border: none !important;
  opacity: 0.5;
}

.users-table .delete-button {
  padding: 6px 12px;
  border-radius: 6px;
  border: none;
  background-color: #dc3545;
  color: white;
  cursor: pointer;
  transition: 0.2s;
}

.users-table .delete-button:hover:not(:disabled) {
  background-color: #b92b38;
}
```

## FUNKCIONALNOST BRISANJA KORISNIKA

---

### NAVBAR.JSX

Moramo spremiti token kako bismo mogli identificirati admina i potvrditi da je to on. U logout treba dodati...

```
localStorage.removeItem("access_token");
```

...dok u signin treba dodati.

```
localStorage.setItem("access_token", idToken);
```

---

#### PERSONCONTROLLER.JAVA

Moramo promjeniti deletePerson funkciju kako bi pravilno provjeravala je li osoba admin ili nije u bazi.

```
public ResponseEntity<?> deletePerson(@PathVariable Long id, @AuthenticationPrincipal Jwt jwt) {
    String email = jwt.getClaimAsString("email");
    Optional<Person> current = repo.findByEmail(email);

    if (current.isEmpty()) return ResponseEntity.status(403).body("Unauthorized");

    Person currentUser = current.get();
    if (currentUser.getId().equals(id)) {
        return ResponseEntity.status(403).body("You cannot delete yourself!");
    }

    repo.deleteById(id);
    return ResponseEntity.ok("User deleted");
}
```

Autor: Josip Mrakovčić Datum završetka: 8.11.2025. Utrošeno vrijeme: ~1 sat

#### NETLIFY BUGOVI I RESPONZIVNOST ADMIN I FORM PAGEA

##### NETLIFY BUGOVI

Imamo problema da hostanjem frontenda na Netlify refresh stranice baca bug da ne postoji stranica. Treba mjenjati sve localhost vrijednosti. ### Hardkodirane localhost varijable U svim .java kontrolerima treba mjenjati ovo...

```
@CrossOrigin(origins = "http://localhost:3000")
```

...u ovo kako bismo kasnije kod hostanja na Render mogli upisati env varijable da se spaja ispravno.

```
@CrossOrigin(origins = "${FRONTEND_URL}")
```

Slično je napravljeno i za frontend za backend i za backend za bazu podataka, ali to već imamo. ### Refresh baca page not found na Netlify Promjenili smo sve...

```
window.location.reload();
```

...u ovo.

```
    navigate(0);
```

S time da to nije bio fix koji je popravio stvari. Rješenje je dodati \_redirect file u public folder na frontendu.

```
/* /index.html 200
```

Sad packageanjem frontenda s npm run build i uploadanjem tog builda na Netlify imamo ispravno refreshanje.  
## Responzivnost ### RentalUnits.css Samo dodajemo na kraju basic responzivnost

```
@media (max-width: 1024px) {  
    .admin-dashboard {  
        flex-direction: column;  
        align-items: center;  
    }  
  
    .sidebar {  
        width: 100%;  
        flex-direction: row;  
        justify-content: space-around;  
        align-items: center;  
        margin-bottom: 20px;  
    }  
  
    .sidebar h2 {  
        display: none;  
    }  
  
    .tab-btn {  
        flex: 1;  
        margin: 5px;  
    }  
  
    .back-main-btn {  
        margin-top: 0;  
        background-color: #4aa3ff;  
    }  
  
    .dashboard-content {  
        width: 100%;  
    }  
  
    .units-list {  
        justify-content: center;  
    }
```

```
}

@media (max-width: 768px) {
  .unit-item {
    flex: 0 1 100%;
  }

  .users-table th,
  .users-table td {
    padding: 8px 6px;
    font-size: 14px;
  }

  .tab-btn,
  .back-main-btn {
    font-size: 14px;
    padding: 8px;
  }

  .section-block {
    padding: 15px;
  }
}

@media (max-width: 500px) {
  .admin-dashboard {
    padding: 10px;
  }

  .unit-item {
    flex-direction: column;
    align-items: flex-start;
  }

  .unit-actions {
    width: 100%;
    display: flex;
    justify-content: flex-end;
    gap: 10px;
    margin-top: 8px;
  }

  .create-button {
    width: 90%;
```

```
}

.users-table {
  font-size: 13px;
}

.users-table th,
.users-table td {
  padding: 6px;
}
}
```

---

#### APARTMENTFORM.CSS

Sličnu stvar radimo i za form page.

```
@media (max-width: 1024px) {
  .form-container {
    width: 90%;
    padding: 25px;
  }

  .apartment-form input,
  .apartment-form textarea {
    font-size: 15px;
  }

  .image-preview img {
    width: 100px;
    height: 80px;
  }
}

@media (max-width: 768px) {
  .form-container {
    width: 95%;
    margin: 20px auto;
    padding: 20px;
  }

  .radio-group {
    flex-direction: column;
    align-items: flex-start;
    gap: 0.5rem;
  }
}
```

```
.num-rooms-inline {
  margin-top: 10px;
}

.image-preview {
  justify-content: center;
}

.preview-item img {
  width: 90px;
  height: 75px;
}

.button-row {
  flex-direction: column;
}

.submit-btn,
.cancel-btn {
  width: 100%;
}
}

@media (max-width: 480px) {
  .form-container {
    padding: 15px;
    border-radius: 10px;
  }

  .form-container h2 {
    font-size: 20px;
  }

  label {
    font-size: 14px;
  }

  .apartment-form input,
  .apartment-form textarea {
    font-size: 14px;
    padding: 8px;
  }
}
```

```
.preview-item img {  
    width: 80px;  
    height: 70px;  
}  
  
.checkbox-label {  
    font-size: 14px;  
}  
}
```

**Autor: Josip Mrakovčić Datum završetka: 8.11.2025. Utrošeno vrijeme: ~1 sati**

### IZJAVA O KORIŠTENJU ALATA UMJETNE INTELIGENCIJE

Tijekom izrade ovog rada korišten je alat ChatGPT (model GPT-5) tvrtke OpenAI kao pomoćni alat za:

- formuliranje i jezično uređivanje teksta,
- pojašnjenje teorijskih pojmoveva,
- generiranje primjera koda i provjeru sintakse.

Sav sadržaj dobiven uz pomoć ChatGPT-a je samostalno provjeren, analiziran i po potrebi izmijenjen od strane autora. Alat nije korišten za generiranje cjelovitih dijelova rada bez nadzora, niti za donošenje zaključaka umjesto autora. Ova uporaba je u skladu s Etičkim kodeksom i pravilnikom Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu o korištenju alata temeljenih na umjetnoj inteligenciji.