

Dokumentacija sustava

Quantum Hotel

Razvojni tim: Sinovi pobjede (grupa TG03.4)

Članovi tima:

- Fran Bistrović
- Nina Jurić
- Dina Jandel
- Marija Špoljarić
- Lukas Kraljić
- Marko Majstorović
- Matija Tušek

Nastavnik: Vlado Sruk

Asistent: Miljenko Krhen

Demonstrator: Ivo Gabud

Kolegij: Programsко inženjerstvo

Ak. godina: 2025./2026.

Verzija: 2.0

Sadržaj:

1. Opis i zahtjevi projektnog zadataka.....	3
2. Analiza zahtjeva sustava.....	6
2.1. Funkcionalni zahtjevi	
2.2. Nefunkcionalni zahtjevi	
2.3. Analiza prema ulogama	
2.4. Dionici projekta	
3. Specifikacija zahtjeva sustava.....	15
3.1. Opis obrazaca uporabe	
3.2. Dijagrami obrazaca uporabe	
3.3. Sekvencijski dijagrami	
3.4. Provjera uključenosti funkcionalnosti	
4. Arhitektura i dizajn sustava.....	37
4.1. Opis arhitekture	
4.2. Obrazloženje odabira arhitekture	
4.3. Organizacija sustava na visokoj razini	
4.4. Organizacija aplikacije	
4.5. Baza podataka	
4.6. Dijagram razreda	
4.7. Dinamičko ponašanje aplikacije	
5. Arhitektura komponenata i razmještaja.....	59
5.1. Dijagram komponenata	
5.2. Dijagram razmještaja	
6. Ispitivanje programskog rješenja.....	61
6.1. Ispitivanje komponenti	
6.2. Ispitivanje sustava	
7. Tehnologije za implementaciju aplikacije.....	91
8. Upute za puštanje u pogon.....	93
9. Zaključak i budući rad.....	101
A. Popis literature.....	102
B. Dnevnik promjene dokumentacije.....	103
C. Prikaz aktivnosti grupe.....	104

1. Opis i zahtjevi projektnog zadatka

U današnjem dinamičnom tržištu usluga, hotelijerstvo se suočava s mnogim izazovima, uključujući složenost upravljanja rezervacijama, optimizaciju kapaciteta, kao i zadovoljstvo gostiju. Tradicionalni načini vođenja hotelskog poslovanja, uključujući ručno upravljanje rezervacijama i podacima o gostima, često dovode do grešaka, preklapanja termina i neefikasnosti. Također, nepostojanje integriranih sustava koji olakšavaju komunikaciju s gostima i praćenje poslovnih performansi može rezultirati lošim korisničkim iskustvom i smanjenjem konkurentnosti na tržištu.

Kao odgovor na ove izazove, razvijena je platforma Quantum Hotel koja ima za cilj unaprijediti i digitalizirati procese upravljanja hotelom. Projekt je rezultirao razvojem cjelovitog informacijskog sustava za hotelsko poslovanje, koji omogućuje automatizaciju ključnih poslovnih funkcija, poput rezervacija, upravljanja smještajem, optimizacije resursa i komunikacije s gostima.

Projekt se temelji na modernim tehnologijama koje omogućuju efikasno upravljanje hotelskim poslovanjem, smanjenje pogrešaka, brži i jednostavniju komunikaciju te bolji uvid u poslovne rezultate. Platforma pruža hotelskom osoblju alat za bolje praćenje kapaciteta i zauzeća, dok gostima omogućuje jednostavnu, sigurnu i transparentnu online rezervaciju.

Ovaj projekt je uspješno završen, a finalni rezultat je sveobuhvatan sustav koji olakšava svakodnevno poslovanje hotela, poboljšava iskustvo gostiju i optimizira radne procese unutar hotela.

Platforma nudi sljedeće ključne funkcionalnosti:

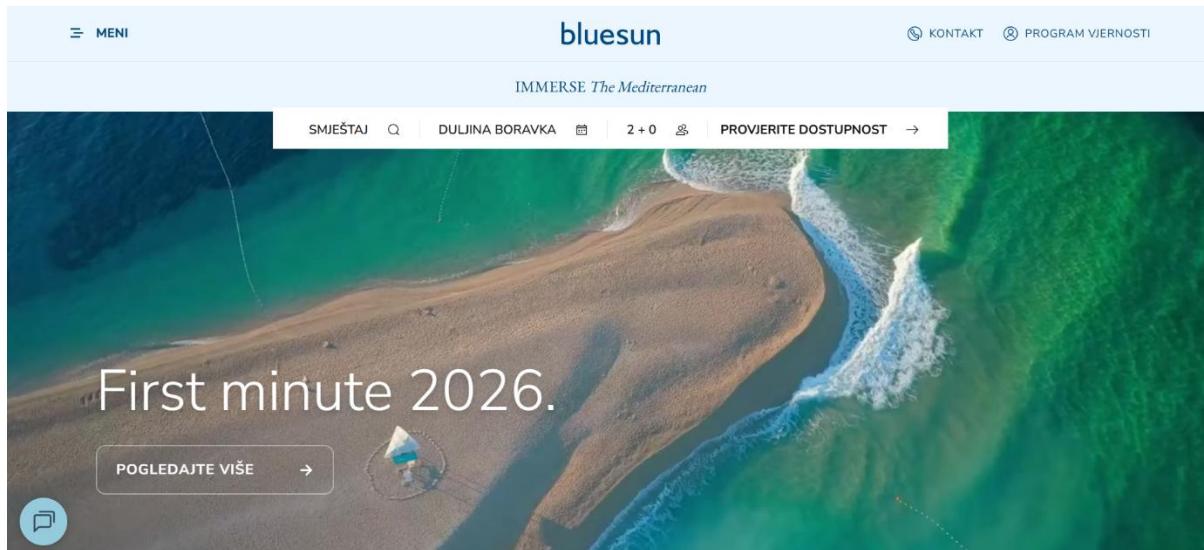
- Autentifikacija i sigurnost: Platforma Quantum Hotel korisnicima nudi preglednu početnu stranicu s jasnim opcijama za registraciju ili prijavu. Kako bi se osigurala fleksibilnost i sigurnost, korisnici mogu odabrati prijavu putem svog postojećeg Google računa, koristeći moderni OAuth2 protokol, ili se registrirati klasičnim putem, upisivanjem svoje e-mail adrese i kreiranjem lozinke. Radi sigurnosti i provjere autentičnosti, nakon registracije e-mailom, korisnicima se automatski šalje potvrda za verifikaciju računa. U slučaju da korisnik zaboravi lozinku, sustav nudi i jednostavnu opciju oporavka lozinke putem e-maila.
- Upravljanje korisničkim profilima: Prijavljeni korisnici imaju pristup svojim korisničkim profilima koje mogu samostalno uređivati. Omogućena je izmjena osobnih podataka poput imena, spola, datuma rođenja, grada, lozinke te e-mail adrese. Platforma definira različite uloge korisnika; dok obični korisnici (gosti) mogu pregledavati i mijenjati isključivo vlastite podatke, ovlašteni djelatnici i administratori imaju uvid u podatke svih korisnika u sustavu, sukladno svojim poslovnim zaduženjima.
- Pregled smještaja i usluga: Glavna funkcionalnost za goste je pregled smještajnih kapaciteta. Korisnici mogu jednostavno pregledavati sve dostupne smještajne jedinice, filtrirati ih prema svojim kriterijima te detaljno proučiti što svaka jedinica nudi. Uz pregled soba i apartmana, korisnicima je na raspolaganju i jasan popis svih dodatnih usluga koje hotel nudi, kao što su rezervacija parking mjesta, opcija doručka, korištenje brze Wi-Fi veze i slično.

- **Sustav rezervacija:** Nakon odabira željene smještajne jedinice, korisnici prelaze na proces rezervacije. Sustav ih vodi kroz odabir željenog termina (datuma dolaska i odlaska), unos broja osoba te odabir ranije spomenutih dodatnih usluga koje žele koristiti tijekom svog boravka (poput parkinga, doručka ili Wi-Fi-ja). Platforma u stvarnom vremenu provjerava dostupnost i omogućuje korisniku da sigurno dovrši svoju rezervaciju.
- **Administrativni panel:** Za potrebe hotelskog osoblja, platforma nudi robustan administrativni panel. Administratori i ovlašteni djelatnici imaju ovlasti za upravljanje cjelokupnim sadržajem hotela. To uključuje mogućnost unosa novih smještajnih jedinica, ažuriranje postojećih te njihovo brisanje. Prilikom unosa, osoblje detaljno definira svaku jedinicu, uključujući kategorizaciju smještaja, cijenu, broj kreveta i svu pripadajuću opremu. Iste ovlasti vrijede i za upravljanje dodatnim uslugama.
- **Integracija s Google Maps:** Kako bi se gostima olakšalo planiranje dolaska i snalaženje, sustav je integriran s Google Maps servisom. Ova funkcionalnost omogućuje korisnicima da na interaktivnoj karti vide točnu lokaciju hotela.
- **Poslovna analitika i izvještavanje:** Poseban segment platforme posvećen je poslovnoj analitici i izvještavanju. Administratori imaju pristup naprednoj statistici koja pruža ključne uvide u poslovanje. Mogu pratiti statistiku o zauzeću smještajnih jedinica, analizirati strukturu gostiju (primjerice, prema zemlji ili gradu dolaska) te dobiti uvid u najtraženije i najprofitabilnije dodatne usluge. Svi ovi ključni podaci mogu se jednostavno preuzeti u standardiziranim formatima (PDF, XML ili XLSX) za daljnju analizu ili arhiviranje.

Usporedba s rješenjima na tržištu

Kao jedan od istaknutih primjera na domaćem tržištu hotelskog poslovanja možemo navesti Blue Sun Hotels. Njihova web platforma predstavlja napredno rješenje primarno usmjereno na pružanje modernog iskustva rezervacije za krajnjeg korisnika, odnosno gosta.

Platforma nudi niz funkcionalnosti koje su postale standard u industriji: korisnicima je omogućen detaljan odabir termina boravka te broja odraslih i djece već pri samom pretraživanju. Budući da se radi o lancu hotela, sustav nudi jasan prikaz svih lokacija i pojedinačnih hotela (npr. u Bolu, Brelima, Tučepima) te omogućuje jednostavno filtriranje po željenoj destinaciji. Korisnicima je na raspolaganju jasan pregled različitih kategorija smještaja, od standardnih soba do ekskluzivnih apartmana, te jednostavna web-forma za slanje upita.



Slika 1.1: Naslovica Blue Sun Hotels WE 1

Iako je platforma Blue Sun Hotels izvrstan primjer modernog booking sustava i služi kao uzor za korisničko sučelje, ona se u svojoj srži fundamentalno razlikuje od ciljeva projekta Quantum Hotel. Ključna razlika je u tome što je Blue Sun sustav zatvoreno, "custom" rješenje razvijeno specifično za potrebe i upravljanje jednog velikog lanca hotela.

S druge strane, Quantum Hotel je realiziran kao univerzalna i cjelovita platforma (informacijski sustav) koju može implementirati bilo koji hotel, neovisno o svojoj veličini. Naš cilj je bio digitalizirati i unaprijediti poslovanje upravo onih hotela koji se još uvijek oslanjaju na tradicionalne i ručne načine vođenja rezervacija, a koji si ne mogu priuštiti razvoj vlastitog, skupog sustava po mjeri.

2. Analiza zahtjeva sustava

2.1. Funkcionalni zahtjevi

ID zahtjeva	Opis	Prioritet	Izvor	Kriteriji prihvaćanja
F-001	Sustav omogućuje registraciju korisnika (Google račun, vlastiti mail ili korisničko ime).	Visok	Zahtjev dionika	Korisnik se može registrirati postojećim Google računom, svojom mail adresom, ili korisničkim imenom. Nakon registracije dobiva potvrdu na e-mail adresu.
F-002	Sustav omogućuje brisanje vlastitog korisničkog računa.	Visok	Zahtjev dionika	Korisnik može zatražiti brisanje vlastitog računa.
F-003	Sustav omogućuje pregled/promjenu korisničkih podataka.	Visok	Zahtjev dionika	Moguća je promjena korisničkih podataka (ime, spol, dob, grad). Svaki korisnik može promjeniti samo svoje vlastite podatke. Admin vidi podatke svih korisnika
F-004	Sustav omogućuje unos/brisanje korisnika.	Visok	Zahtjev dionika (admina)	Admin (vlasnik sustava) ima mogućnost stvaranja novih računa (za djelatnika) i brisanja postojećih korisničkih računa.
F-005	Sustav ima integriranu uslugu „Google Maps“.	Visok	Zahtjev dionika	Korisnik unutar sustava može koristiti uslugu „Google Maps“.
F-006	Sustav omogućuje korisniku odabir željenog termina i broja osoba.	Visok	Zahtjev dionika	Korisnik unutar sustava mora moći odabrati željeni termin i broj osoba. Sustav korisniku tada prikazuje dostupne smještajne jedinice koje zadovoljavaju ove uvjete
F-007	Sustav omogućuje pregled dostupnih smještajnih jedinica.	Visok	Zahtjev dionika	Korisnik dobiva pregled dostupnih smještajnih jedinica na temelju odabranog termina i broja osoba.
F-008	Sustav omogućuje rezerviranje odabranog termina uz odabir dodatnih usluga.	Visok	Zahtjev dionika	Korisnik mora moći odabrati neku od ponuđenih opcija,

				odabrati dodatne usluge ako ih želi i rezervirati odabranu jedinicu, termin i usluge.
F-009	Sustav omogućuje pregled/unos/uređivanje svih smještajnih jedinica.	Visok	Postojeći sustav	Korisnik može pregledavati postojeće smještanje jedinice uz mogućnost sortiranja po vremenskom razdoblju. Moguć je unos nove vrste smještajnih jedinica i uređivanje neke postojeće.
F-010	Sustav omogućuje korisnicima pretragu/stvaranje/brisanje/uređivanje „članaka“ o dodatnim aktivnostima.	Nizak	Postojeći sustav	Korisnik unutar sustava može pregledavati članke o dodatnim aktivnostima hotela. Djelatnik hotela može kreirati nove stavke te uređivati i brisati postojeće
F-011	Sustav omogućuje pregled/stvaranje/uređivanje kategorija smještajnih jedinica.	Srednji	Postojeći sustav	Korisnik može pretraživati postojeće kategorije smještajnih jedinica. Korisnik također može stvarati i ažurirati kategorije smještajnih jedinica (cijena noćenja, broj kreveta...)
F-012	Sustav omogućuje pregled/unos/ažuriranje dodatnih usluga.	Nizak	Postojeći sustav	Korisnik može pretraživati postojeće dodatne usluge. Korisnik može stvoriti i ažurirati dodatne usluge koje se nude prilikom rezervacije smještajne jedinice (parking, doručak...).
F-013	Sustav omogućuje pregled te potvrdu/odbijanje ili izmjenu svih rezervacija.	Srednji	Zahtjev dionika(admin, djelatnik)	Korisnik (admin ili djelatnik) može pregledavati sve aktualne rezervacije. Korisnik (admin, djelatnik) može

				promijeniti rezervaciju. Gost u slučaju potvrde, odbijanja ili izmjene dobiva obavijest e-mailom.
F-014	Sustav vodi statistiku o noćenjima.	Visok	Zahtjev dionika	Korisnik (admin) može preuzeti podatke o zauzetim sobama i broju noćenja (u željenom vremenskom razdoblju i prema kategorijama smještajnih jedinica).
F-015	Sustav vodi statistiku o strukturi gostiju.	Visok	Zahtjev dionika	Korisnik (admin) može vidjeti iz kojih gradova dolaze pojedini gosti. Korisnik (admin) može vidjeti koje dobi gostiju najviše dolaze i koje su najtraženije kategorije smještajnih jedinica.
F-016	Sustav vodi statistiku o dodatnim uslugama.	Visok	Zahtjev dionika	Korisnik (admin) može vidjeti koje su najtraženije dodatne usluge.
F-017	Sustav ima mogućnost preuzimanja statističkih podataka.	Visok	Zahtjev dionika	Korisnik (admin) može preuzeti željenu statistiku u formatima pdf, xml ili xlsx.
F-018	Sustav ima implementiranu korisničku podršku.	Srednji	Zahtjev korisnika (gosta)	Korisnik ima dostupne odgovore na često postavljana pitanja vezana uz korištenje sustava te ima mogućnost slanja vlastitih pitanja (preko forme se šalje mail djelatnicima, oni odgovaraju mailom).
F-019	Sustav primjenjuje optimizirano rezerviranje smještajnih jedinica.	Visok	Zahtjev dionika	Sustav mora voditi računa o preklapanju termina i minimizirati broj dana kada su smještajne jedinice slobodne.
F-020	Sustav omogućuje dodjelu uloga korisnicima.	Srednji	Zahtjev dionika (admina)	Admin svakom prijavljenom može promijeniti ulogu i dati prava pristupa.

2.2. Nefunkcionalni zahtjevi

- NF = ne-funkcionalni zahtjevi
- NF 3.1 = zahtjevi performansi
- NF 3.2 = zahtjevi sigurnosti
- NF 3.3 = zahtjevi pouzdanosti
- NF 3.4 = zahtjevi skalabilnosti
- NF 3.5 = zahtjevi održivosti
- NF 3.6 = zahtjevni prenosivosti
- NF 3.7 = zahtjevi upotrebljivosti
- NF 3.8 = zahtjevi dostupnosti
- NF 3.9 = zahtjevi interoperabilnosti
- NF 3.10 = zahtjevi zakonitosti i usklađenosti

ID zahtjeva	Opis	Prioritet
NF-3.1.1	Sustav treba biti oblikovan tako da omogućuje jednostavno održavanje i nadogradnju.	Visok
NF-3.1.2	Vrijeme učitavanja i korištenja stranice / aplikacije mora biti prihvatljivo.	Visok
NF-3.3.1	Sustav mora podržavati velik broj istovremenih korisnika bez značajnog narušavanja performansi i funkcionalnosti.	Visok
NF-3.5.1	Sustav treba imati dovoljnu dokumentaciju.	Srednji
NF-3.2.1	Osjetljivi podaci korisnika moraju biti zaštićeni pristupnim kontrolama i dostupni samo ovlaštenim korisnicima.	Visok
NF-3.7.1	Korisničko sučelje mora biti intuitivno i jednostavno za upotrebu.	Srednji
NF-3.7.2	Web stranica mora imati responzivan dizajn i biti prilagođena za mobilne uređaje.	Visok
NF-3.8.1	Sustav mora biti konstantno dostupan za upotrebu (barem 99% vremena).	Srednji
NF-3.8.2	Održavanje se provodi u neaktivnim satovima radi minimizacije nedostupnosti sustava.	Nizak

2.3. Analiza zahtjeva prema ulogama u sustavu

Zahtjev	Gost (neprijavljeni)	Gost (prijavljeni)	Djelatnik	Admin	Napomena
Administriranje korisničkih računa					
Registracija s Google računom	+	+		+	OAuth2
Registracija s vlastitim računom	+	+		+	
Registracija s dodijeljenim korisničkim imenom			+	+	Admin svakom djelatniku (zaposleniku) dodjeljuje korisničko ime i lozinku koju onda on koristi za prijavu u sustav.
Odjava iz sustava		+	+	+	
Brisanje vlastitog računa		+	+	+	
Promjena korisničkih podataka (ime, prezime, mjesto stanovanja, dob, spol, lozinka)		+	+	+	Nakon prve prijave korisnik unosi sve podatke koji se onda spremaju za ubuduće. Djelatnik može promijeniti lozinku koju mu je dodijelio admin.
Stvaranje korisnika (djelatnika)				+	
Uređivanje korisnika (djelatnika)				+	
Dodjela uloge djelatnika				+	
Brisanje korisnika				+	
Dodavanje uloga				+	Admin može korisniku dodijeliti ili maknuti ulogu i ovisno o tome korisnik može vidjeti različite stvari na stranici (3 moguće uloge su opisane u ovoj tablici).
Hotel					
Interakcija s Google Maps	+	+	+	+	
Prikaz lokacije hotela	+	+	+	+	
Unos „članaka“ o dodatnim aktivnostima			+	+	Korisnik unosi informacije o dodatnom sadržaju u hotelu (npr. wellness, sport, rekreacija, obližnje atrakcije...).
Pregled „članaka“	+	+	+	+	

Odabir željenog termina, broja osoba	+	+	+	+	Korisnik odabire željeni termin, broj osoba. Sustav tada traži dostupne sobe za korisnika.
Pregled odgovora na često postavljena pitanja (FAQ)	+	+	+	+	Korisnik na FAQ stranici može vidjeti odgovore na često postavljena pitanja.
Postavljanje vlastitih pitanja		+	+	+	Korisnik putem forme može poslati pitanje vezano uz korištenje sustava. Forma se mailom šalje djelatniku. Djelatnik mailom odgovara na postavljeno pitanje.
Pregled dostupnih smještajnih jedinica	+	+	+	+	Na temelju odabranih opcija, korisniku se prikazuju dostupne smještajne jedinice koje odgovaraju upitu.
Rezervacija		+	+	+	Gost mora biti prijavljen kako bi mogao izvršiti rezervaciju. Korisnik odabire neku ponuđenu opciju. Odabire i neke dodatne opcije (npr. doručak). Automatski mu se dodjeljuje soba. Kada admin ili djelatnik potvrdi, na mail mu stiže potvrda o rezervaciji.
Hotelsko poslovanje					
Pregled smještajnih jedinica			+	+	Korisniku se prikazuju sve postojeće smještajne jedinice. Korisnik može vidjeti koje smještajne jedinice su zauzete u kojem vremenskom periodu. Za svaku smještajnu jedinicu može se vidjeti i njezina kategorija.

Uređivanje često postavljenih pitanja i pripadajućih odgovora			+	+	Korisnik dodaje nova pitanja i odgovarajuće odgovore na FAQ dijelu.
Unos/ažuriranje smještajnih jedinica			+	+	Korisnik unosi nove sobe i dodjeljuje im kategoriju. Korisnik može mijenjati status održavanja sobe (ispravno/u kvaru) i status čišćenja (prljavo/čisto).
Pregled kategorija smještajnih jedinica			+	+	Korisnik može vidjeti kojim sve vrstama soba hotel trenutno raspolaže. Ovo se ne odnosi na gosta koji vidi samo dostupne kategorije za svoje odabранo razdoblje, nego na zaposlene u hotelu.
Unos/ažuriranje kategorija smještajnih jedinica			+	+	Korisnik unosi npr. novu sobu i ispunjava karakteristike (cijena, koliko kreveta...). Podaci se mogu ažurirati. Ovo se događa u slučaju ako se uvodi npr. nova 5-krevetna soba.
Pregled dodatnih usluga	+	+	+	+	Korisniku se nudi sučelje za pregled i unos dodatnih usluga. One će se gostu prikazati tek kod odabere neke konkretnе sobe (kao što je navedeno u zahtjevu F-008)
Unos/ažuriranje dodatnih usluga			+	+	Korisnik unosi koje će se dodatne usluge ponuditi gostu pri rezervaciji sobe (parking, wifi, ručak, večera). Uz svaku uslugu unosi se cijena.
Pregled vlastitih rezervacija		+	+	+	Korisnik može vidjeti sve rezervacije za svoj račun.

Pregled svih rezervacija			+	+	Korisnik može vidjeti sve rezervacije u nekom razdoblju (od svih gostiju).
Izmjena/potvrda/odbijanje rezervacija			+	+	Admin i djelatnik mogu odbiti/potvrditi neku rezervaciju ili promijeniti vrijeme odlaska i dolaska. Gost dobiva obavijest mailom.
Hotelska statistika					
Izvješće o noćenjima			+		Korisnik odabire razdoblje i dobiva podatke o zauzetim sobama. Može vidjeti broj noćenja u sobama svake kategorije u tom razdoblju.
Izvješće o strukturi gostiju			+		Korisnik može vidjeti iz kojih gradova dolazi najviše gostiju, gosti koje dobi najviše dolaze i koju kategoriju soba uzimaju.
Izvješće o dodatnim uslugama			+		Korisnik može vidjeti koje su najtraženije dodatne usluge.
Mogućnost preuzimanja podataka			+		Odabrani podaci mogu se prikazati u PDF, XML, XLSX formatu i izvesti na računalo.

Napomena: Smještajna jedinica je jedna soba (npr. soba broj 5). Podatak o zauzetosti pregledavamo/mijenjamo za konkretnu smještajnu jedinicu, ne za kategoriju. Za svaku smještajnu jedinicu veže se njezina kategorija. Gost odabire kategoriju sobe.

2.4. Dionici projekta

Gost (korisnik)

Glavni korisnik sustava, čiji je cilj korištenje sustava za pronalaženje odgovarajućeg smještaja u hotelu u željenom terminu.

Glavne funkcionalnosti za gosta uključuju:

- Mogućnost registracije, prijave i odjave (F-001)
- Mogućnost promjene podataka i brisanje računa (F-002, F-003)
- Korištenje integracije „Google Maps“ (F-005)
- Pregledavanje članaka o dodatnim aktivnostima (F-010)
- Biranje željenih termina, broja osoba, pregledavanje dostupnih smještajnih jedinica i vršenje rezervacija (F-006, F-007, F-008)
- Ima dostupnu korisničku podršku (FAQ + postavljanje vlastitih pitanja) (F-018)
- Gost mora biti prijavljen da bi imao mogućnost rezerviranja smještaja i manipulacije vlastitog računa

Djelatnik (korisnik)

Djelatnik je korisnik kojem admin (vlasnik sustava) dodjeljuje ovlasti za obavljanje različitih zadataka unutar aplikacije. Djelatnici su zaduženi za:

- Održavanje stranice i pomoći adminu
- Održavanje korisničke podrške (FAQ) i odgovaranje na postavljena pitanja korisnika mailom (F-018)
- Prijavljanje preko dodijeljenog korisničkog računa (F-001, F-002)
- Stvaranje novih članaka o dodatnim aktivnostima (F-010)
- Pregledavanje, stvaranje i ažuriranje kategorija smještajnih jedinica (F-011)
- Pregledavanje, unos i ažuriranje smještajnih jedinica (F-019)
- Pregledavanje, unos i ažuriranje dodatnih usluga (F-012)
- Pristup pregledu svih rezervacija (F-013)

Admin (korisnik)

Admin je vlasnik sustava, s punim pristupom svim funkcionalnostima sustava. Admin ima mogućnost:

- Dodijeliti ulogu djelatnika korisnicima
- Potvrditi i izmijeniti rezervacije (F-013)
- Pristupiti svim statističkim podacima sustava i preuzeti te podatke (F-014, F-015, F-016, F-017)

Razvojni tim

Razvojni tim primjenjuje znanje programskog inženjerstva kako bi razvio zadalu aplikaciju prema specifikacijama i zahtjevima.

3. Specifikacija zahtjeva sustava

3.1. Opis obrazaca uporabe

UC1: Registracija u sustav

- Glavni aktor: gost, djelatnik, admin
- Cilj: Uči u aplikaciju i započeti korištenje
- Preduvjet: Imati postojeći i ispravan Google račun (ako se prijavljuje preko OAuth2)
- Opis osnovnog tijeka:
 1. Korisnik se prijavljuje s Google računom preko OAuth2 ili korisničkim imenom
 2. Nakon prve prijave korisnik dobiva potvrdu na e-mail adresu
- Opis mogućih odstupanja:
 - 1.a Korisnik nema e-mail ili je unio krive podatke
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC1.1: Slanje e-mail potvrde

- Glavni aktor: gost, djelatnik, admin
- Cilj: Osigurati da je korisnik ispravno prijavljen
- Preduvjet: Korisnik obavlja prvu prijavu putem Google računa
- Opis osnovnog tijeka:
 1. Sustav šalje korisniku potvrdu na e-mail
- Opis mogućih odstupanja: -

UC2: Upravljanje podacima korisničkog računa

- Glavni aktor: prijavljeni gost, djelatnik, admin
- Cilj: Korisnik želi ažurirati svoje osobne podatke ili izbrisati svoj račun
- Preduvjet: Korisnik ima prijavljeni i potvrđeni račun
- Opis osnovnog tijeka:
 1. Korisnik otvara svoj profil
 2. Korisnik uređuje dostupne i željene podatke (ime, prezime, spol, ...), a ako su uneseni podaci ispravni podaci se uspješno ažuriraju ili bira gumb za brisanje računa
- Opis mogućih odstupanja:
 - 2.a Korisnik je unio pogrešne podatke
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC3: Upravljanje korisničkim računima

- Glavni aktor: admin
- Cilj: Upravljati korisnicima sustava (djelatnicima i gostima) kroz dodjelu, ažuriranje i brisanje računa i uloga
- Preduvjet: Admin je prijavljen u sustav i ima aktivna administrativna prava
- Opis osnovnog tijeka:
 1. Admin pristupa upravljačkom sučelju za korisnike
 2. Admin može odabratи opciju za stvaranje novog korisnika, uređivanje ili brisanje postojećeg
 3. Spremanje izmjena
- Opis mogućih odstupanja:
 - 3.a Neuspješno dohvaćanje ili spremanje podataka u bazu

UC3.1: Administracija djelatnika

- Glavni aktor: admin
- Cilj: Dodati novog ili ukloniti korisnika sa statusom djelatnika
- Preduvjet: Admin je prijavljen i ima pristup upravljanju korisnicima
- Opis osnovnog tijeka:
 1. Admin otvara obrazac za dodavanje novog korisnika i unosi potrebne podatke (ime, prezime, e-mail, ...) ili odabire onog kojeg želi izbrisati i klikne na "Izbriši"
 2. Novi korisnički račun se kreira i automatski aktivira ili se izbriše zajedno sa svim povezanim podacima
- Opis mogućih odstupanja:
 - 2.a Uneseni e-mail već postoji u sustavu, krivo popunjeni podaci
 1. Sustav javlja korisniku o pogrešci

UC3.2: Ažuriranje uloga

- Glavni aktor: admin
- Cilj: Promijeniti ulogu postojećem korisniku
- Preduvjet: Admin je prijavljen i ima pristup upravljanju korisnicima
- Opis osnovnog tijeka:
 1. Admin otvara profil korisnika i odabire novu ulogu za korisnika
 2. Sustav ažurira korisničku ulogu i prikazuje poruku o uspjehu
- Opis mogućih odstupanja:
 - 2.a Neispravno uneseni podaci
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC4: Pregled informacija

- Glavni aktor: gost, djelatnik, admin
- Cilj: Pregledati informativne sadržaje sustava
- Preduvjet: -
- Opis osnovnog tijeka:
 1. Sustav prikazuje dostupne aktivne sadržaje
 2. Korisnik odabire željeni sadržaj za detaljan pregled
 3. Sustav prikazuje traženi sadržaj
- Opis mogućih odstupanja:
 - 3.a Nema dostupnog sadržaja
 1. Sustav javlja korisniku da nema dostupnog sadržaja

UC4.1: Interakcija s Google Maps

- Glavni aktor: gost, djelatnik, admin
- Cilj: Pregledati lokaciju objekta na interaktivnoj karti
- Preduvjet: U sustav je integrirana Google Maps usluga
- Opis osnovnog tijeka:
 1. Korisnik odabire gumb za lokaciju
 2. Sustav učitava interaktivnu Google kartu.
- Opis mogućih odstupanja: -

UC4.2: FAQ

- Glavni aktor: gost, djelatnik, admin
- Cilj: Dobiti odgovor na često postavljena pitanja
- Preduvjet: FAQ sadržaji postoje u sustavu
- Opis osnovnog tijeka:
 1. Korisnik otvara stranicu s FAQ
 2. Korisnik bira pitanje za prošireni prikaz odgovora
- Opis mogućih odstupanja:
 - 2.a Ne postoji odgovor na određeno pitanje

UC4.3: Pregled članka

- Glavni aktor: gost, djelatnik, admin
- Cilj: Pregledati članke o dodatnim aktivnostima
- Preduvjet: U sustavu postoje članci
- Opis osnovnog tijeka:
 1. Korisnik odabire kategoriju „Članci“ i sustav prikazuje listu članaka
 2. Korisnik čita članke
- Opis mogućih odstupanja:
 - 1.a Nema objavljenih članaka

UC5: Odabir termina i broja osoba

- Glavni aktor: gost, djelatnik, admin
- Cilj: Omogućiti korisniku odabir želenog termina i broja osoba za rezervaciju
- Preduvjet: Korisnik je prijavljen
- Opis osnovnog tijeka:
 1. Korisnik otvara sučelje za rezervaciju.
 2. Unosi željeni datum dolaska i odlaska, odabire broj osoba.
 3. Sustav provjerava dostupnost smještajnih jedinica
- Opis mogućih odstupanja:
 - 3.a Nema dostupnih soba za odabrani termin
 1. Sustav javlja korisniku da nema dostupnih soba

UC5.1: Pregled dostupnih smještajnih jedinica

- Glavni aktor: gost, djelatnik, admin
- Cilj: Prikazati dostupne smještajne jedinice na odabrani termin i broj osoba
- Preduvjet: Unos termina, broja osoba
- Opis osnovnog tijeka:
 1. Sustav prikazuje listu smještajnih jedinica dostupnih za odabrani termin
 2. Korisnik može filtrirati i pregledati detalje jedinica
- Opis mogućih odstupanja:
 - 1.a Nema dostupnih jedinica
 1. Sustav javlja korisniku da nema dostupnih jedinica

UC5.2: Rezervacija

- Glavni aktor: gost, djelatnik, admin
- Cilj: Kreirati novu rezervaciju za odabranu jedinicu, termin i broj osoba
- Preduvjet: Odabrana dostupna smještajna jedinica i termin
- Opis osnovnog tijeka:
 1. Korisnik odabire jedinicu i potvrđuje rezervaciju
 2. Unosi potrebne podatke
 3. Sustav sprema rezervaciju i šalje potvrdu korisniku
- Opis mogućih odstupanja: -

UC5.3: Odabir dodatnih usluga

- Glavni aktor: gost, djelatnik, admin
- Cilj: Omogućiti korisniku da uz rezervaciju odabere dodatne usluge
- Preduvjet: Kreiranje rezervacije ili u tijeku rezervacijskog procesa
- Opis osnovnog tijeka:
 1. Sustav prikazuje dostupne dodatne usluge
 2. Korisnik odabire željene usluge
 3. Sustav ažurira i potvrđuje izbor
- Opis mogućih odstupanja:
 - 3.a Odabrana usluga više nije dostupna
 1. Sustav javlja korisniku da odabrana usluga više nije dostupna

UC6: Upravljanje smještajnim jedinicama

- Glavni aktor: djelatnik, admin
- Cilj: Omogućiti pregled, unos i ažuriranje podataka o smještajnim jedinicama
- Preduvjet: Korisnik je prijavljen s odgovarajućim pravima (djelatnik ili admin)
- Opis osnovnog tijeka:
 1. Korisnik pristupa sučelju za upravljanje smještajem
 2. Može pregledavati i uređivati postojeće smještajne jedinice ili unositi nove jedinice
 3. Sustav validira i sprema izmjene
- Opis mogućih odstupanja:
 - 3.a Korisnik nema odgovarajuće ovlasti
 1. Sustav javlja korisniku da nema ovlasti

UC6.1: Pregled smještajnih jedinica

- Glavni aktor: djelatnik, admin
- Cilj: Pregledati sve ili filtrirane smještajne jedinice
- Preduvjet: Korisnik ima pristup sustavu i odgovarajuću ulogu
- Opis osnovnog tijeka:
 1. Korisnik otvara stranicu s popisom smještajnih jedinica
 2. Može koristiti sortiranje: po cijeni, održavanju, vremenskom razdoblju... Rezultati se prikazuju u obliku tablica
- Opis mogućih odstupanja:
 - 3.a Nema dostupnih jedinica za zadane filtere
 1. Sustav javlja korisniku da nema dostupnih jedinica

UC6.2: Održavanje smještajnih jedinica

- Glavni aktor: djelatnik, admin
- Cilj: Dodati novu ili ažurirati postojeću smještajnu jedinicu
- Preduvjet: Korisnik ima odgovarajuću ulogu i pristup uređivanju
- Opis osnovnog tijeka:
 1. Korisnik otvara obrazac za unos nove jedinice ili uređivanje postojeće i unosi ili mijenja podatke (naziv jedinice, kategorija (soba, apartman...), održavanje, čišćenje...)
 2. Sustav sprema jedinicu u bazu
- Opis mogućih odstupanja:
 - 2.a Podaci nisu ispravno popunjeni
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC6.3: Upravljanje kategorijama jedinica

- Glavni aktor: djelatnik, admin
- Cilj: Upravljati kategorijama smještajnih jedinica
- Preduvjet: Korisnik je prijavljen s ulogom djelatnika ili administratora
- Opis osnovnog tijeka:
 1. Korisnik otvara modul za upravljanje kategorijama
 2. Pregledava postojeće kategorije i može dodati novu kategoriju ili ažurirati postojeću
 3. Sustav validira i sprema podatke
- Opis mogućih odstupanja:
 - 3.a Neispravni ili nepotpuni podaci
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC7: Upravljanje informacija

- Glavni aktor: djelatnik, admin
- Cilj: Upravljati sadržajem za korisničku podršku
- Preduvjet: Prijava u sustav s odgovarajućim ovlastima
- Opis osnovnog tijeka:
 1. Korisnik bira između FAQ, dodatnih usluga i članaka i uređuje jednog od njih
- Opis mogućih odstupanja: -

UC7.1: Uređivanje FAQ

- Glavni aktor: djelatnik, admin
- Cilj: Upravljati sadržajem sekcije FAQ kako bi korisnicima bila dostupna ažurirana pomoć
- Preduvjet: Korisnik je prijavljen s odgovarajućim ovlastima
- Opis osnovnog tijeka:
 1. Korisnik otvara sučelje za uređivanje FAQ-a i uređuje pitanja i odgovore
 2. Nakon spremanja sustav ažurira prikazani sadržaj
- Opis mogućih odstupanja:
 - 2.a Pogrešno ispunjeni podaci
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC7.2: Upravljanje dodatnim uslugama

- Glavni aktor: djelatnik, admin
- Cilj: Upravljati dodatnim uslugama koje gosti mogu uključiti uz rezervaciju
- Preduvjet: Korisnik ima pristup administraciji ili djelatničkom sučelju
- Opis osnovnog tijeka:
 1. Korisnik može pregledati, uređivati i izbrisati postojeće ili unijeti nove usluge
 2. Sustav validira i spremi podatke.
- Opis mogućih odstupanja:
 - 2.a Neispravni ili nepotpuni podaci
 1. Sustav javlja korisniku da provjeri podatke koje je upisao

UC7.3: Upravljanje člancima o dodatnim aktivnostima

- Glavni aktor: djelatnik, admin
- Cilj: Uređivanje članaka o dodatnim aktivnostima
- Preduvjet: Korisnik je prijavljen s odgovarajućim ovlastima
- Opis osnovnog tijeka:
 1. Korisnik može kreirati nove i uređivati postojeće članke o dodatnim aktivnostima kako bi pružio bolju uslugu
- Opis mogućih odstupanja: -

UC8: Upravljanje rezervacijama

- Glavni aktor: djelatnik, admin
- Cilj: Upravljati rezervacijama gostiju, osigurati njihovu validaciju i pravovremenu obradu
- Preduvjet: Korisnik je prijavljen s odgovarajućim ovlastima
- Opis osnovnog tijeka:
 1. Korisnik pristupa modulu rezervacija.
 2. Korisnik može pregledati detalje, potvrditi, odbiti ili izmijeniti rezervacije
 3. Sustav šalje obavijest gostu putem e-maila o promjeni statusa rezervacije
- Opis mogućih odstupanja: -

UC9: Pristup statistici hotela

- Glavni aktor: admin
- Cilj: Pristupiti glavnom pregledniku svih dostupnih statistika sustava
- Preduvjet: Admin je prijavljen u sustav
- Opis osnovnog tijeka:
 1. Admin odabire modul „Statistika“
 2. Sustav prikazuje glavne kategorije statistike
- Opis mogućih odstupanja:
 - 1.a Statistički podaci privremeno nedostupni
 1. Sustav javlja korisniku da su podaci privremeno nedostupni

UC9.1: Pregled statistike noćenja

- Glavni aktor: admin
- Cilj: Pregledati broj noćenja i zauzetost u određenom vremenskom razdoblju
- Preduvjet: Admin ima pristup statistici
- Opis osnovnog tijeka:
 1. Admin odabire „Rezervacije“, unosi vremenski raspon
 2. Sustav prikazuje broj rezervacija i prihod po kategoriji smještaja
- Opis mogućih odstupanja:
 - 1.a Nema podataka za traženi period
 1. Sustav javlja korisniku da nema podataka za traženi period

UC9.2: Pregled statistike o strukturi gostiju

- Glavni aktor: admin
- Cilj: Pregledati demografske i geografske statistike gostiju
- Preduvjet: Admin ima pristup statistici
- Opis osnovnog tijeka:
 1. Admin odabire gumb „Gosti“
 2. Sustav prikazuje statistike po državi, gradu, dobnu strukturu i po spolu
- Opis mogućih odstupanja:
 - 2.a Zbog nedovoljno podataka prikazana statistika nije od velike važnosti

UC9.3: Pregled statistike o dodatnim uslugama

- Glavni aktor: admin
- Cilj: Pregledati koje dodatne usluge gosti najčešće koriste
- Preduvjet: Admin ima pristup statistici
- Opis osnovnog tijeka:
 1. Admin odabire gumb „Usluge“
 2. Sustav prikazuje listu najtraženijih dodatnih usluga i broj korištenja
- Opis mogućih odstupanja:
 - 2.a Nema podataka o određenim uslugama
 1. Sustav javlja korisniku da nema podataka o određenim uslugama

UC9.4: Preuzimanje statističkih podataka

- Glavni aktor: admin
- Cilj: Preuzeti statističke podatke u različitim formatima
- Preduvjet: Postoje odabrani i generirani statistički podaci
- Opis osnovnog tijeka:
 1. Admin odabire odabire željeni format u kojem želi preuzeti podatke
 2. Sustav generira datoteku i nudi je za preuzimanje
- Opis mogućih odstupanja: -

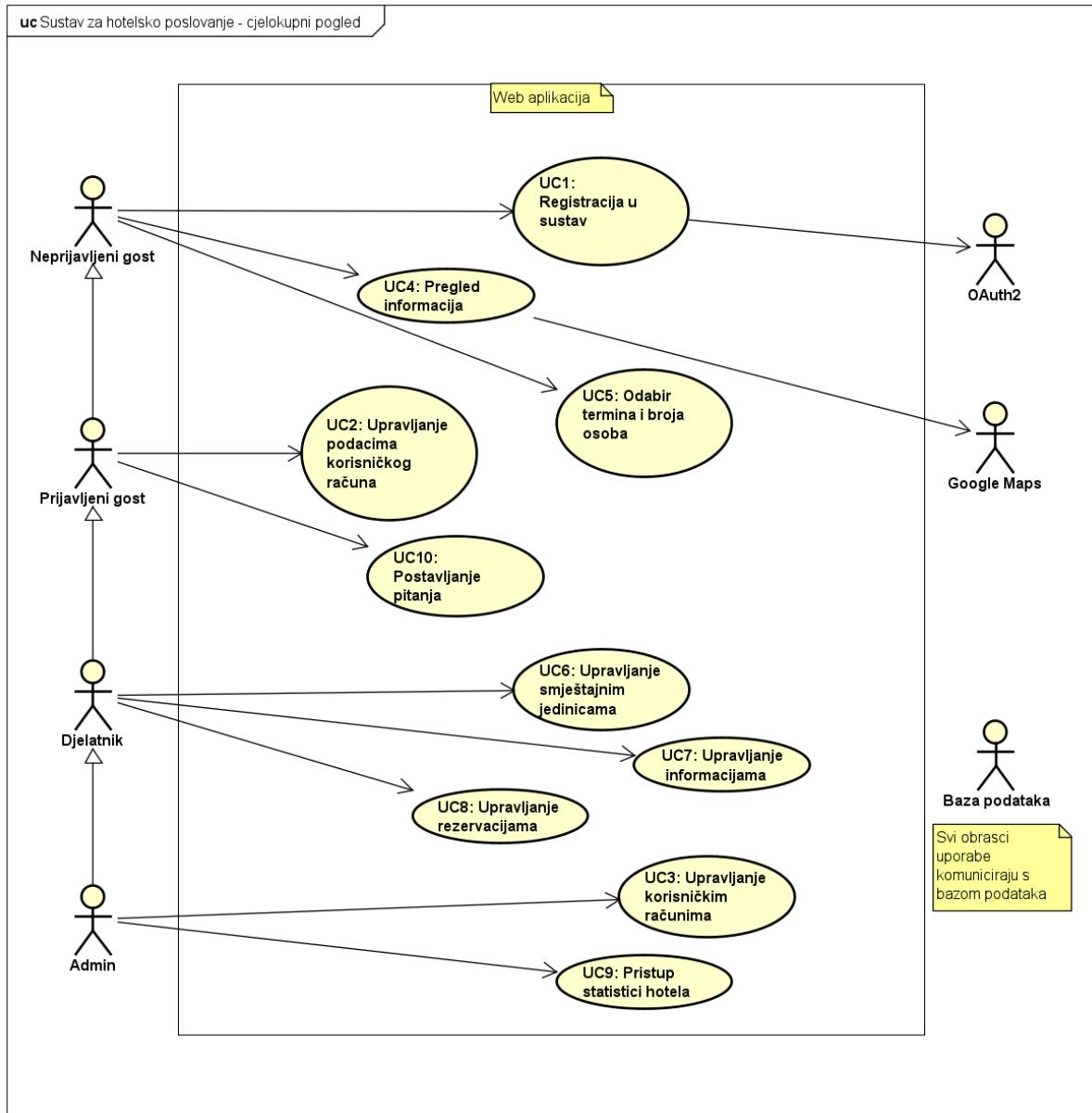
UC10: Postavljanje pitanja

- Glavni aktor: djelatnik, prijavljeni gost
- Cilj: Omogućiti korisnicima da postave pitanje i dobiti povratnu informaciju
- Preduvjet: Korisnik je prijavljen u sustav
- Opis osnovnog tijeka:
 1. Korisnik šalje pitanje u formi
 2. Korisnik dobiva potvrdu da je njegovo pitanje uspješno postavljeno
 3. Pitanje se proslijeđuje nekom od djelatnika
- Opis mogućih odstupanja: -

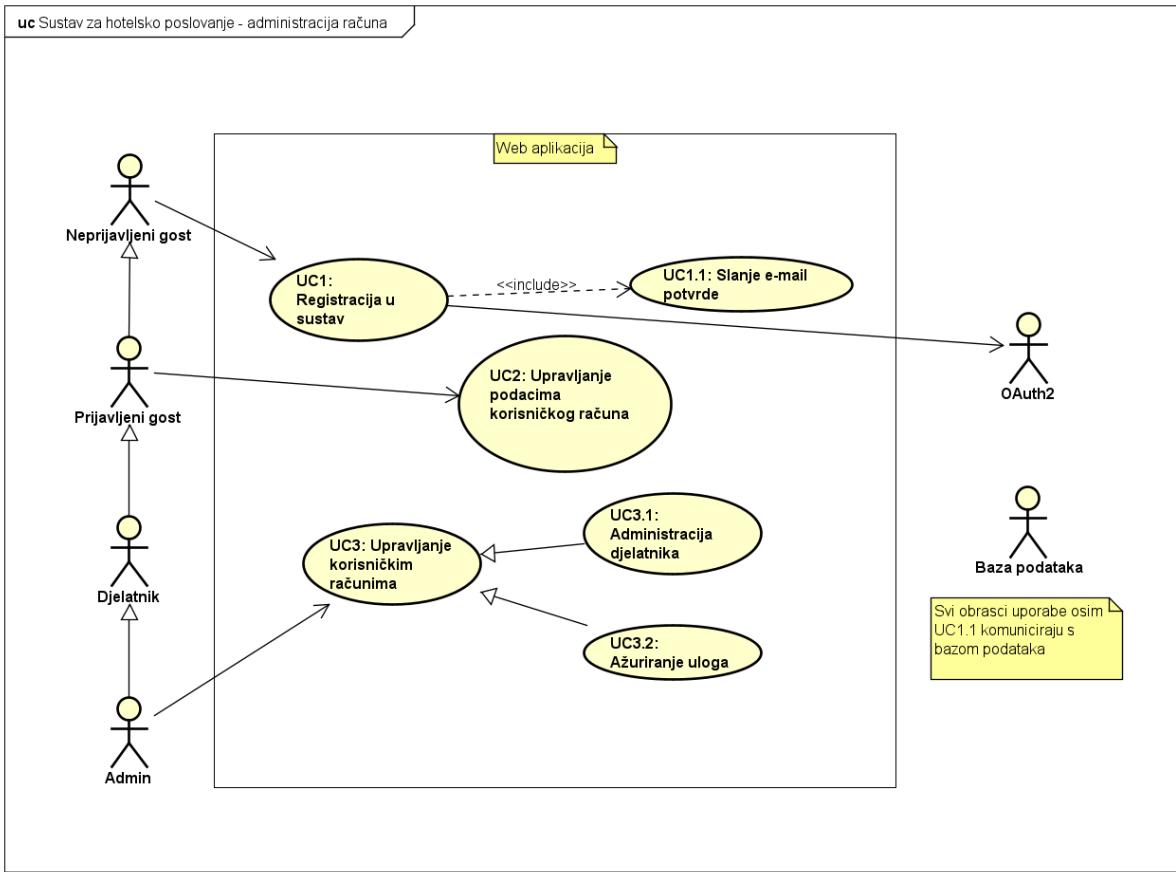
UC10.1: Odgovor na pitanje

- Glavni aktor: djelatnik, prijavljeni gost
- Cilj: Korisnik dobiva odgovor na postavljeno pitanje
- Preduvjet: Korisnik je postavio pitanje
- Opis osnovnog tijeka:
 1. Djelatnik dobiva i obradi upit
 2. Djelatnik šalje korisniku odgovor na postavljeno pitanje
- Opis mogućih odstupanja: -

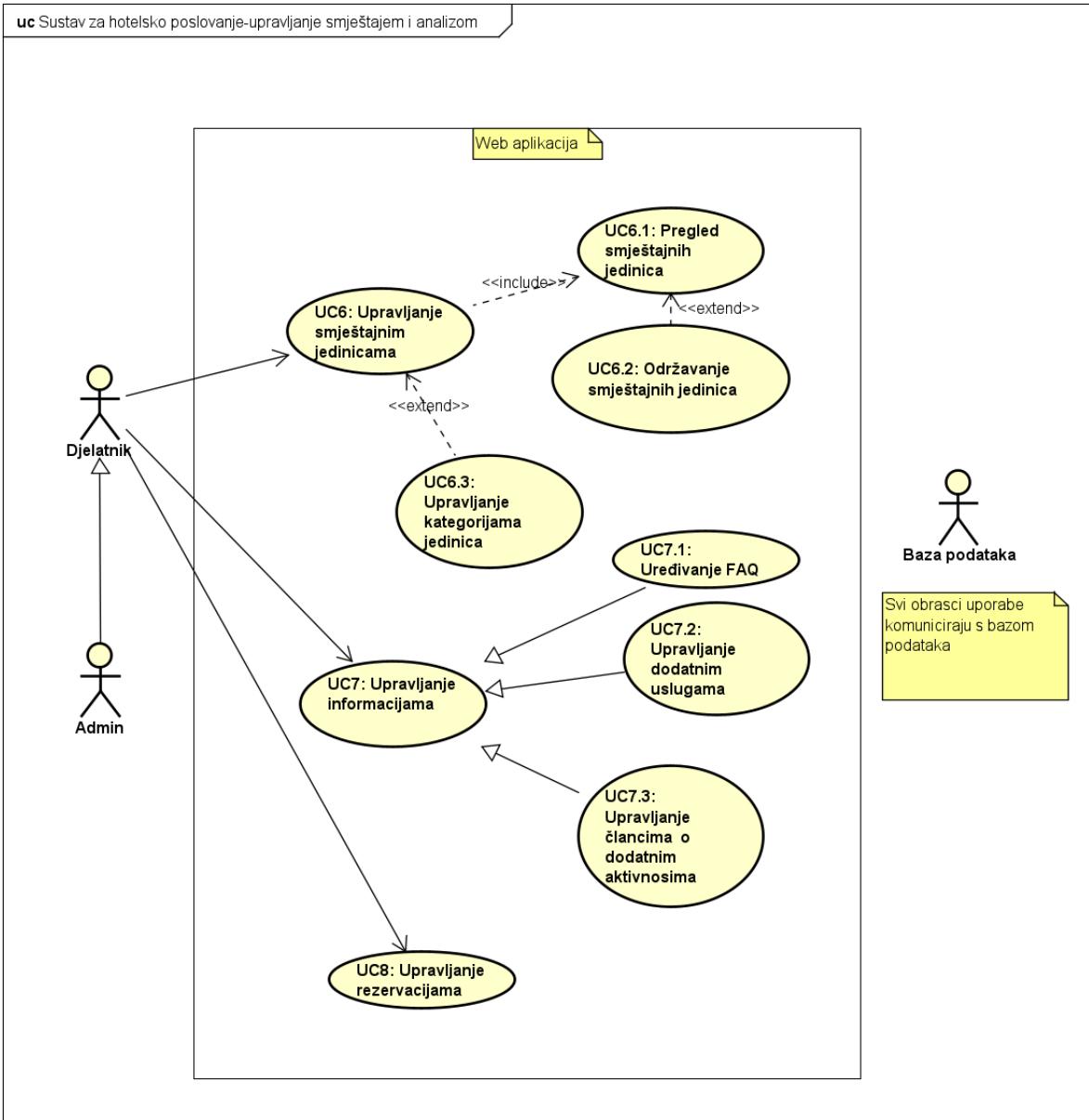
3.2. Dijagrami obrazaca uporabe



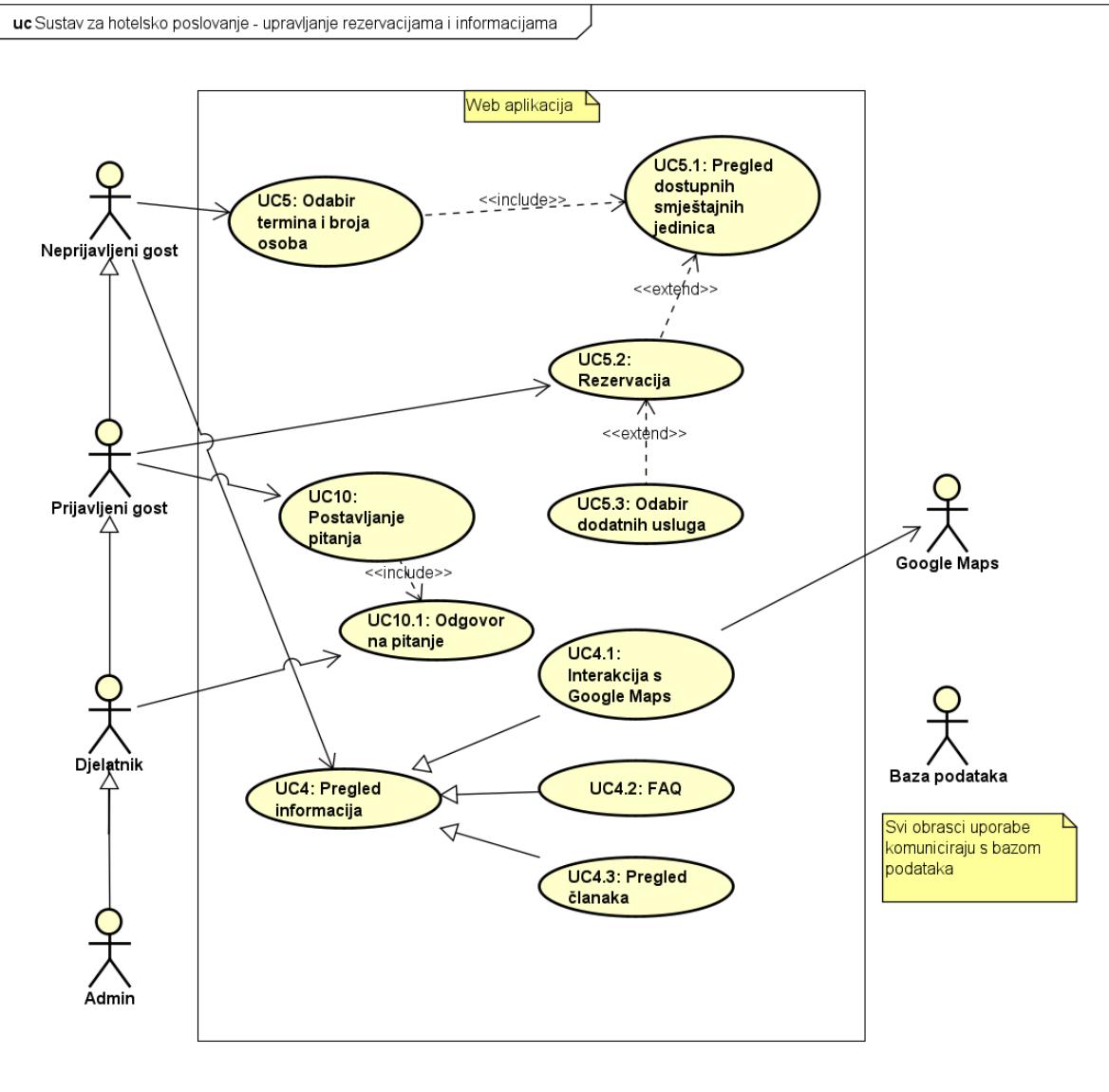
Slika 3.1: Sustav za hotelsko upravljanje – cjelokupni pogled



Slika 3.2: Sustav za hotelsko poslovanje – administracija računa

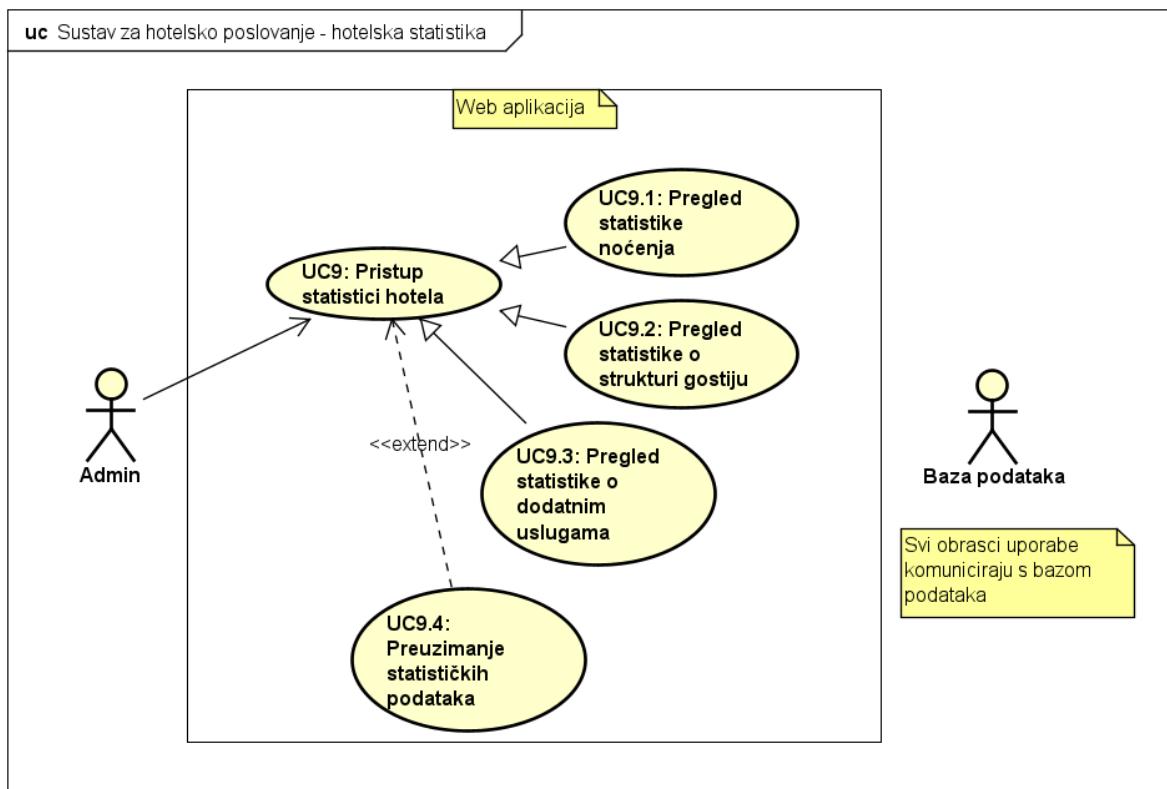


Slika 3.3: Sustav za hotelsko poslovanje – upravljanje smještajem i analizom



Slika 3.4: Sustav za hotelsko poslovanje – upravljanje rezervacijama i informacijama

uc Sustav za hotelsko poslovanje - hotelska statistika



Slika 3.5: Sustav za hotelsko poslovanje – hotelska statistika

3.3. Sekvencijski dijagrami

3.3.1. Registracija u sustav

Sekvencijski dijagram "UC1: Registracija u sustav" prikazuje tok interakcije između četiri ključna sudionika: nepovezanog gosta (korisnika koji još nije prijavljen), web aplikacije, vanjskog OAuth2 servisa i baze podataka sustava.

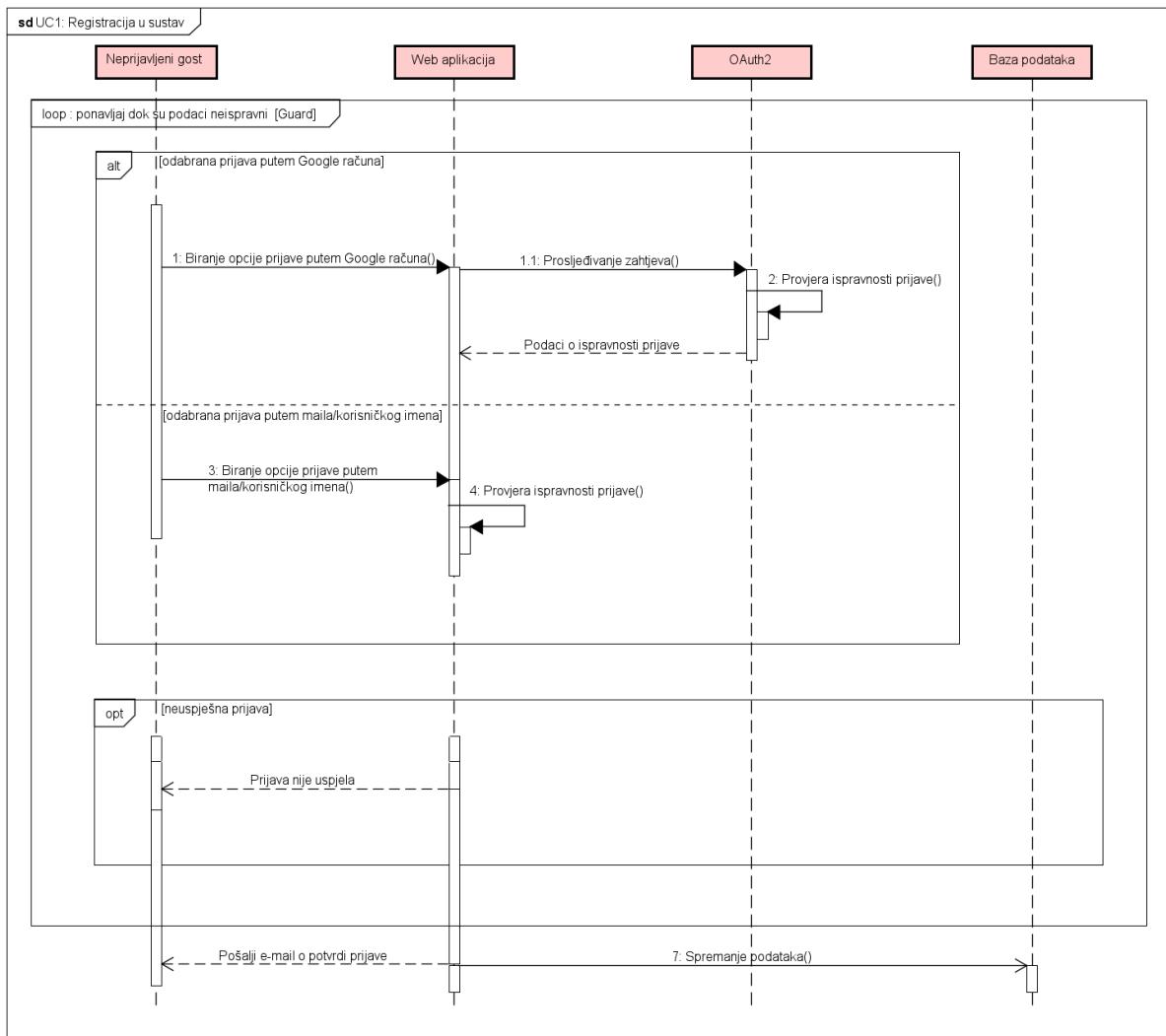
Proces započinje kada neprijavljeni gost inicira prijavu putem Googlea, maila ili korisničkog imena. Ako je odabrana prijava putem Googlea web aplikacija interpretira to kao zahtjev za autentifikaciju i prosljeđuje ga OAuth2 servisu. OAuth2 obrađuje zahtjev i vraća odgovor koji sadrži autorizacijski token ili kod, čime se potvrđuje uspješna autentifikacija korisnika.

Ako korisnik umjesto Google autentifikacije odabere prijavu putem e-mail adrese ili korisničkog imena, tada web aplikacija sama preuzima odgovornost za provjeru vjerodajnica bez uključivanja vanjskih servisa. Proces započinje kada neprijavljeni gost unese svoje podatke u odgovarajuće polje za prijavu – to uključuje e-mail adresu ili korisničko ime te lozinku. Web aplikacija zatim lokalno validira unesene podatke prema unaprijed definiranim pravilima (npr. e-mail mora biti u ispravnom formatu, korisničko ime ne smije sadržavati nedozvoljene znakove, lozinka mora imati minimalnu duljinu i barem jedan broj ili specijalni znak) aplikacija inicira sesiju i korisnik se uspješno registrira u sustav.

U slučaju da podaci nisu ispravni ili korisnik ne postoji, aplikacija vraća poruku o pogrešnoj prijavi i može ponuditi opciju ponovne registracije. Ovaj tok se odvija potpuno interno, bez potrebe za vanjskim servisima.

Ako je registracija uspješno provedena, web aplikacija šalje poruku Spremanje podataka() prema bazi podataka, čime se trajno pohranjuju korisnički podaci – uključujući e-mail adresu, korisničko ime, hash lozinke (i eventualne dodatne informacije poput datuma registracije ili uloge korisnika). Aplikacija također inicira slanje potvrde korisniku putem e-maila ukoliko je registracija uspješno provedena.

Dijagram jasno prikazuje glavni tok registracije, uključujući autentifikaciju preko vanjskog servisa, provjeru ispravnosti prijave lokalno, pohranu podataka te završnu potvrdu korisniku. Uključivanje uvjetnih i ponovljenih tokova omogućuje modeliranje scenarija neuspjeha i ponovne registracije, čime se osigurava pouzdan proces registracije.



Slika 3.6: Sustav za hotelsko upravljanje – UC1 Registracija u sustav

3.3.2. Postupak rezervacije soba

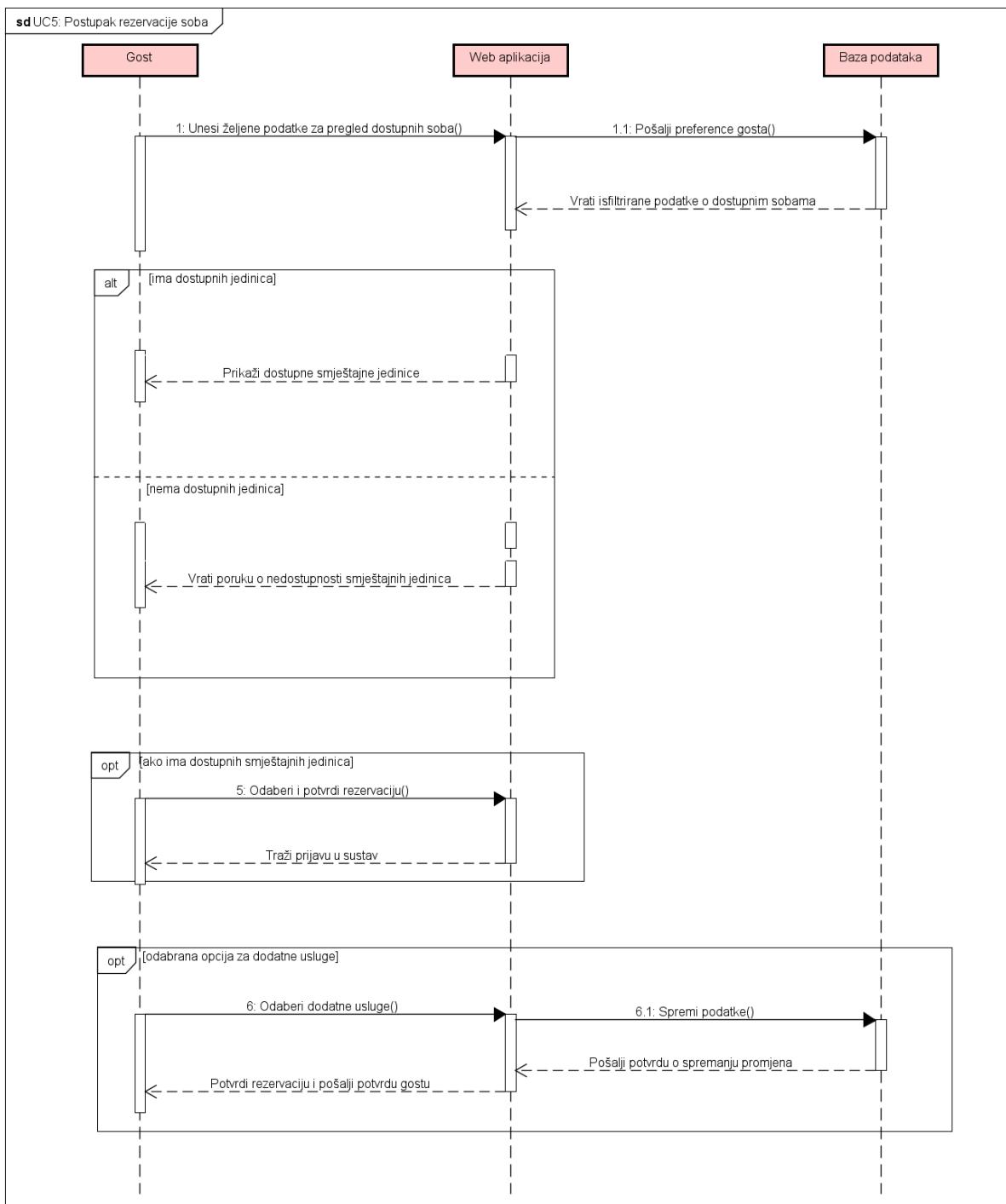
Sekvencijski dijagram "UC5: Postupak rezervacije sobe" prikazuje interakciju između gosta, web aplikacije i baze podataka tijekom procesa rezervacije smještaja.

Proces započinje kada gost inicira pregled dostupnih soba po kriterijima termina i broja osoba, što se u dijagramu prikazuje porukom prema web aplikaciji. Web aplikacija zatim šalje upit bazi podataka kako bi dohvatila filtrirane informacije o slobodnim sobama, a baza vraća rezultate koji se prikazuju gostu. Ako u tom rezultatu ima dostupnih smještajnih jedinica iste se prikazuju gostu, a ako se ništa ne poklapa s gostovim preferencijama gostu se prikazuje poruka o nedostupnosti.

Zajedno s prikazom dostupnih soba gost dobiva opciju da prihvati rezervaciju (za što je nužno biti prijavljen u sustav, znači ta mogućnost se prikazuje jedino ako se radi o prijavljenom korisniku).

Nakon potvrđene rezervacije gostu se prikazuje mogućnost dodatnih opcija vezanih uz sobu – primjerice, tip kreveta, pogled, klima uređaj ili druge pogodnosti. Gost zatim odabire dodatne usluge, poput doručka, parkinga ili wellness paketa, a web aplikacija bilježi te izbore. U sljedećem koraku aplikacija kreira rezervaciju, što se prikazuje porukom prema bazi podataka da se podaci spreme. U bazi podataka se svi relevantni podaci – identifikator gosta, odabrana soba, dodatne usluge, datumi boravka – trajno pohranjuju.

Dijagram prikazuje kako gost putem web aplikacije pregledava dostupne sobe, odabire sobu i dodatne usluge, te potvrđuje rezervaciju. Web aplikacija provjerava dostupnost u bazi podataka i pohranjuje sve relevantne podatke o rezervaciji, uključujući gosta, odabranoj sobu, dodatne usluge i datume boravka



Slika 3.6: Sustav za hotelsko upravljanje – UC5 Postupak rezerviranja sobe

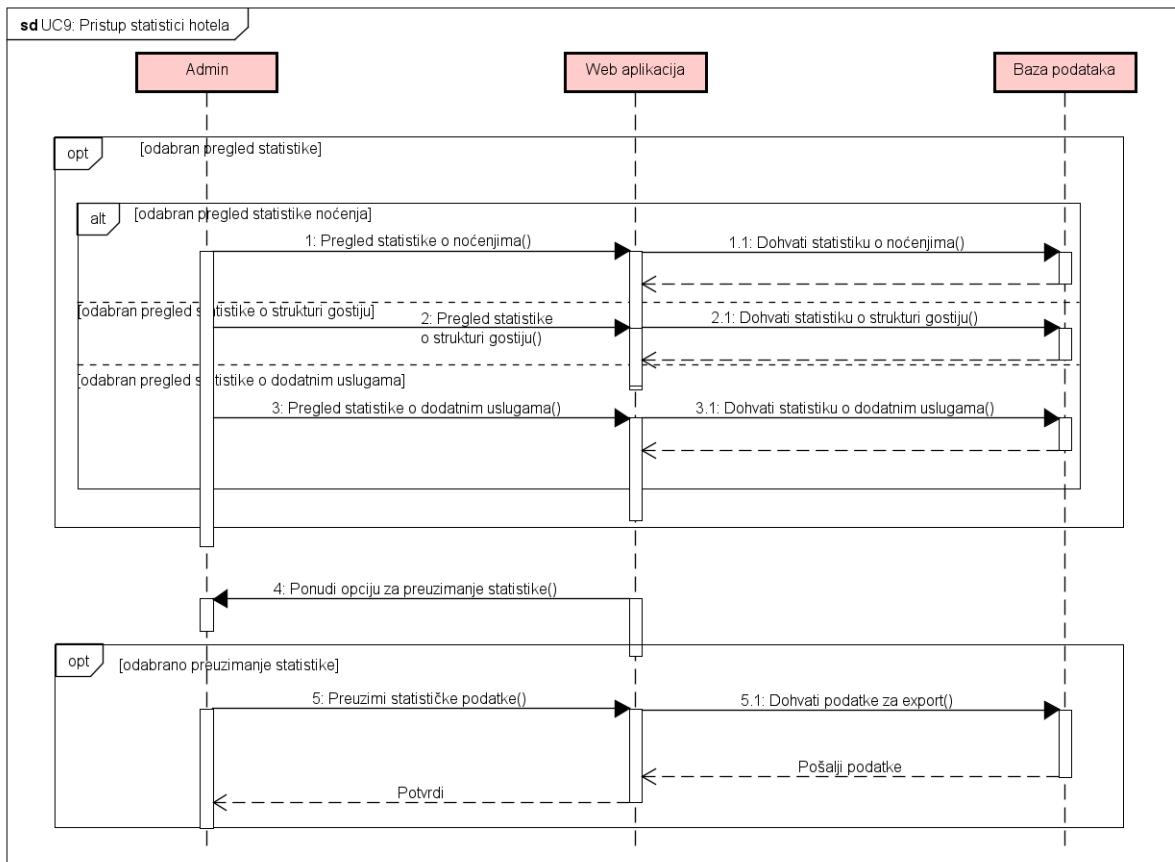
3.3.3 Pristup statistici hotela

Sekvencijski dijagram "UC9: Pristup statistici hotela" prikazuje tok interakcije između administratora, web aplikacije i baze podataka tijekom pristupa hotelskim statističkim podacima.

Proces započinje kada administrator odabere jednu od dostupnih kategorija statistike - statistiku noćenja, strukturu gostiju ili dodatne usluge. Izbor koja od tih dostupnih statistika se želi pregledati je unutar zasebnog opt bloka, čime se jasno naznačuje da se radi o optionalnoj funkcionalnosti koja se aktivira ovisno o korisničkom odabiru.

Nakon odabira statistike koju želimo pregledati aplikacija šalje zahtjev prema bazi podataka kako bi dohvatala željene podatke te ih prikazala adminu. Aplikacija također nudi administratoru mogućnost preuzimanja podataka, što je prikazano kao zasebna poruka. Ako administrator odabere opciju za izvoz, aplikacija od baze traži podatke i administratora još jednom pita da potvrди download.

Ovaj dijagram jasno prikazuje kako sustav omogućuje fleksibilan pristup različitim vrstama statistike, uz mogućnost izvoza podataka.



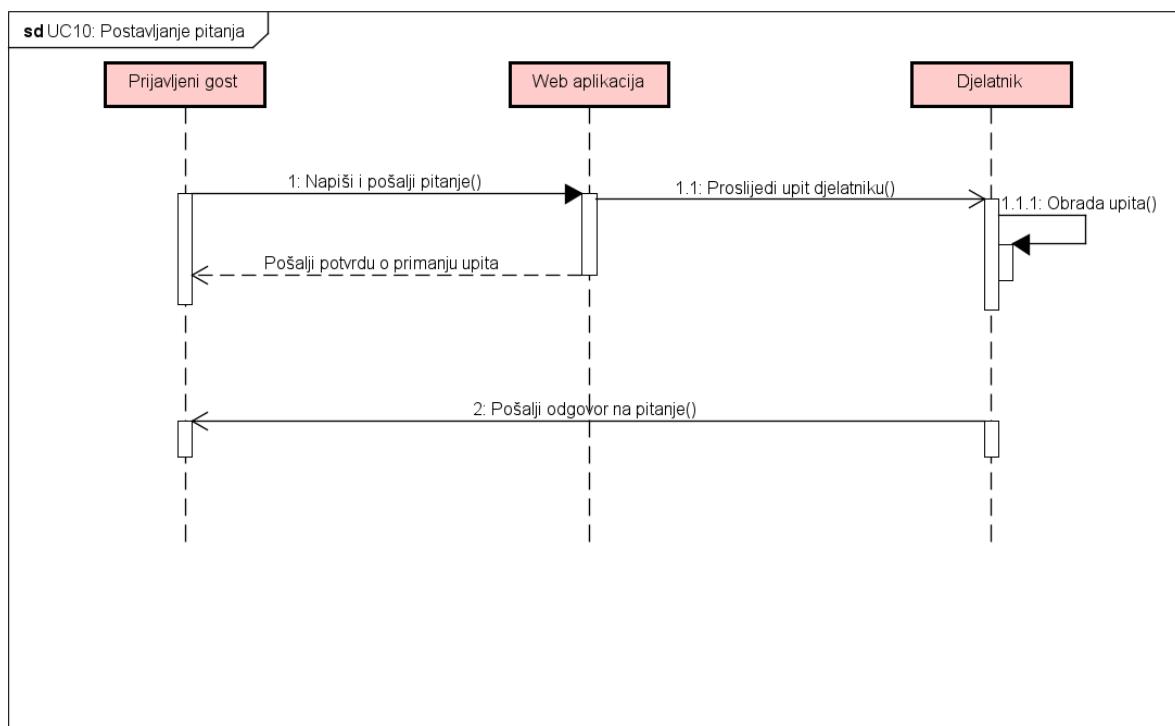
Slika 3.8: Sustav za hotelsko upravljanje - UC9 Pristup statistici hotela

3.3.4 Postavljanje pitanja

Sekvencijski dijagram "UC10: Postavljanje pitanja" prikazuje tok komunikacije između prijavljenog gosta, web aplikacije i djelatnika hotela tijekom procesa slanja upita i primanja odgovora.

Proces započinje kada prijavljeni gost napiše i pošalje pitanje putem web aplikacije. Taj korak uključuje automatsku potvrdu o primitku upita, čime se korisniku signalizira da je njegov zahtjev uspješno zaprimljen. Web aplikacija zatim proslijeđuje upit djelatniku, koji je zadužen za njegovu obradu. Unutar tog koraka prikazan je podproces obrade upita, koji može uključivati čitanje, klasifikaciju, traženje odgovora ili formulaciju odgovora prema vrsti pitanja. Nakon što djelatnik obradi upit, web aplikacija šalje odgovor natrag prijavljenom gostu.

Sekvencijski dijagram prikazuje tok slanja upita i daljnje komunikacije od prijavljenog gosta do djelatnika hotela i povratnog odgovora putem web aplikacije.



Slika 3.9: Sustav za hotelsko upravljanje – UC10 Postavljanje pitanja

3.4. Provjera uključenosti ključnih funkcionalnosti u obrasce uporabe

ID Obrasca Uporabe	Naziv Obrasca Uporabe	Povezani Funkcionalni Zahtjevi
UC1	Registracija u sustav	F-001
UC1.1	Slanje e-mail potvrde	F-001
UC2	Upravljanje podacima korisničkog računa	F-002, F-003
UC3	Upravljanje korisničkim računima	F-004, F-020
UC3.1	Administracija djelatnika	F-004
UC3.2	Ažuriranje uloga	F-020
UC4	Pregled informacija	F-005, F-018, F-010
UC4.1	Interakcija s Google Maps	F-005
UC4.2	FAQ	F-018
UC4.3	Pregled članaka	F-010
UC5	Odabir termina i broja osoba	F-006
UC5.1	Pregled dostupnih smještajnih jedinica	F-007
UC5.2	Rezervacija	F-008
UC5.3	Odabir dodatnih usluga	F-008
UC6	Upravljanje smještajnim jedinicama	F-009
UC6.1	Pregled smještajnih jedinica	F-011
UC6.2	Održavanje smještajnih jedinica	F-019
UC6.3	Upravljanje kategorijama jedinica	F-012
UC7	Upravljanje informacija	F-018, F-012, F-010
UC7.1	Uređivanje FAQ	F-018
UC7.2	Upravljanje dodatnim uslugama	F-012
UC7.3	Upravljanje člancima o dodatnim aktivnostima	F-010
UC8	Upravljanje rezervacijama	F-013
UC9	Pristup statistici hotela	F-014, F-015, F-016, F-017
UC9.1	Pregled statistike noćenja	F-014
UC9.2	Pregled statistike o strukturi gostiju	F-015
UC9.3	Pregled statistike o dodatnim uslugama	F-016
UC9.4	Preuzimanje statističkih podataka	F-017
UC10	Postavljanje pitanja	F-018
UC10.1	Odgovor na pitanje	F-018

4. Arhitektura i dizajn sustava

4.1. Opis arhitekture

Stil arhitekture

Quantum Hotel sustav koristi klijent-poslužitelj (client-server) arhitektonski stil. Prema tome koristi mrežne protokole u ulozi konektora (HTTP, SQL upiti) da bi povezao komponente (klijenta i poslužitelja). Ograničenje ovakvog sustava su da nema međusobne komunikacije između klijenta, već se sva komunikacija izvodi s poslužiteljem. Ovaj arhitektonski stil omogućuje centralizaciju logike i podataka (u bazi) te je pogodan za razvoj web i mobilnih aplikacija. Također, primjenom ovog modela osigurana je skalabilnost i fleksibilnost sustava i jasna podjela odgovornosti među timovima.

Podsustavi

Glavni podsustavi Quantum Hotel aplikacije uključuju:

- Autentifikacijski podsustav: Upravljanje registracijom i prijavom uz mogućnost prijave Google OAuth2 servisom ili korisničkim podacima
- Podsustav za upravljanje profilima: Omogućuje promjenu korisničkih podataka, brisanje i stvaranje korisničkih računa
- Podsustav za hotelsko poslovanje: Omogućuje pregled i upravljanje smještajnih jedinica, njihovih kategorija i dodatnih usluga, upravljanje i stvaranje rezervacija te upravljanje ostalim funkcionalnostima ključnim za hotelsko poslovanje
- Podsustav za hotelsku statistiku: Omogućuje uvid o statistici noćenja, strukturi gostiju i dodatnim uslugama te omogućuje preuzimanje tih podataka u zadanim formatima

Preslikavanje na radnu platformu

Za implementaciju hotelskog informacijskog sustava odabran je cloud pristup s naglaskom na korištenje kontejnera pomoću Dockera. Ovakav pristup omogućava visoku dostupnost, sigurnost, skalabilnost i fleksibilnost potrebnu za moderno hotelsko poslovanje. Spremista podataka U svrhu pohranjivanja podataka sustav će koristiti PostgreSQL relacijsku bazu podataka. Ovaj pristup je jednostavan i pogodan za razvojne potrebe sustava te omogućuje učinkovitu pohranu strukturiranih podataka poput podataka o korisnicima, kategorijama smještajnih jedinica i rezervacijama.

Mrežni protokoli

Komunikacija između klijenta i poslužitelja odvijat će se korištenjem HTTP protokola. Ovaj protokol omogućuje jednostavan prijenos podataka korištenjem GET, POST, PUT, DELETE i PATCH metoda te osigurava stabilnost pri velikom broju zahtjeva. Također su moguća proširenja u slučaju potrebe zaštite podataka (npr. HTTPS protokol).

Globalni upravljači tok

Korisnik započinje interakciju sa sustavom putem registracije ili prijave. Nakon što se korisnik prijavi u sustav, ovisno o svojoj ulozi (gost, djelatnik, admin) ima različite ovlasti. Generalno korisnik može vidjeti lokaciju hotela, članke o dodatnim aktivnostima u hotelu, pretraživati dostupne smještanje jedinice i stvarati rezervacije. Svi potrebni podaci za ove interakcije unutar sustava šalju se preko HTTP protokola uz odgovarajuće komuniciranje s PostgreSQL bazom podataka.

Sklopoškoprogramske zahtjevi

Tehnički zahtjevi za sustav Quantum Hotel uključuju podršku za operativne sustave na poslužiteljskoj strani (Linux, Windows Server) i klijentskoj strani (Windows, macOS, iOS, Android). Server treba imati dovoljno RAM memorije (preporučeno najmanje 16 GB) i brzi procesor (barem osam jezgre) za podršku istovremenog rada više korisnika. Za PostgreSQL bazu podataka i aplikacijski server potrebno je osigurati diskovni prostor od minimalno 100 GB za pohranu podataka o rezervacijama, gostima, sobama i statistici hotela. Za optimalne performanse potrebna je stabilna i dovoljno brza mreža.

4.2. Obrazloženje odabira arhitekture

Obrazloženje izbora arhitekture

Odabir klijent-poslužiteljskog arhitektonskog stila za sustav Quantum Hotel temelji se na potrebi za centraliziranim upravljanjem podacima i logikom sustava, kao i na zahtjevu za dostupnošću putem različitih uređaja (računalo, tablet, mobilni uređaj). Ovaj arhitektonski pristup omogućuje jasnu podjelu odgovornosti između poslužiteljske i klijentske strane, što pojednostavljuje održavanje, razvoj i nadogradnju sustava. Poslužiteljski dio zadužen je za poslovnu logiku, autentifikaciju korisnika, upravljanje podacima i komunikaciju s bazom PostgreSQL, dok klijentski omogućuje responzivno i intuitivno korisničko sučelje. Korištenje Docker kontejnera dodatno doprinosi prenosivosti i skalabilnosti sustava, što omogućuje jednostavno pokretanje i održavanje u različitim okruženjima, uključujući cloud infrastrukturu.

Ključni čimbenici izbora arhitekture

Pri odabiru arhitekture uzeti su u obzir sljedeći čimbenici:

- Skalabilnost i fleksibilnost: Klijent-poslužitelj model i kontejnerizacija omogućuju horizontalno i vertikalno skaliranje sustava, čime se lako može prilagoditi povećanom broju korisnika i zahtjevima hotela.
- Održavanje i proširivost: Jasna podjela između klijenta, poslužitelja i baze podataka omogućuje neovisne nadogradnje bez prekida rada sustava.
- Sigurnost: Centralizirano upravljanje autentifikacijom putem OAuth2 (Google prijava) i sigurno spremanje u bazi podataka osiguravaju zaštitu osjetljivih korisničkih podataka.
- Performanse i dostupnost: Arhitektura omogućuje optimalno iskorištavanje resursa, dok kontejnerizacija i cloud pristup osiguravaju visoku dostupnost (99% vremena rada).

Principi oblikovanja arhitekture

Prilikom oblikovanja sustava primjenjeni su ključni principi dobrog arhitektonskog dizajna:

- Visoka kohezija: Svaki podsustav (autentifikacija, profili, hotelsko poslovanje, statistika) obavlja jasno definiranu funkciju unutar svog područja odgovornosti.
- Niska povezanost: Komunikacija među podsustavima ostvaruje se putem standardiziranih REST API poziva, što omogućuje njihovu neovisnost i jednostavnu zamjenu ili nadogradnju.

4.3. Organizacija sustava na visokoj razini

Sustav je organiziran u nekoliko ključnih komponenti na višoj razini apstrakcije koje su zajednički odgovorne za funkcionalnost i ispravan rad sustava

Klijent-poslužitelj

Quantum Hotel sustav organiziran je u klijent-poslužitelj arhitekturi koja omogućava centralizaciju logike i podataka te fleksibilnost i skalabilnost sustava. Podaci se spremaju i obrađuju na poslužitelju, a klijenti uslugama pristupaju preko vlastitih uređaja. Klijentski dio sustava predstavlja interaktivna aplikacija u kojima korisnici mogu rezervirati smještaj u hotelu uz podrazumijevane funkcionalnosti (prijava, upravljanje računom, pregled sadržaj i dostupnih smještaja...) Poslužiteljski dio zadužen je za poslovnu logiku aplikacije (hotelsko poslovanje), obrađuje zahtjeve klijenata i komunicira s bazom podataka u svrhu upravljanja podacima.

Baza podataka

Quantum Hotel sustav koristi relacijsku bazu podataka PostgreSQL koja omogućava pohranu svih potrebnih podataka za ispravan rad sustava. Baza sadrži podatke o korisnicima, smještajnim jedinicama, statistici hotela i ostale potrebne informacije. Relacijska baza nudi prednosti poput dosljednosti podataka i nudi mogućnosti kreiranja složenijih upita. Također, korištenje PostgreSQL sustava široko je podržan pristup koji komplimentira ostale komponente sustava (Spring Boot).

Datotečni sustav

Zbog potreba izvoza dokumenata (o statistici) u različite formate i slanja potvrda o rezervacijama korisnicima, sustav će dinamički generirati dokumente bez lokalnog pohranjivanja na disk. Ovaj pristup je jednostavniji za održavanje i u skladu s kontejnerizacijom (Docker).

Grafičko sučelje

Korisničko sučelje sustava Quantum Hotel implementirano je kao web aplikacija dostupna putem internetskog preglednika. Sučelje je responzivno i omogućuje pristup s različitih uređaja (mobitel, tablet, laptop, stolno računalo). Sama aplikacija pruža intuitivno sučelje za korištenje glavnih usluga sustava poput prijave/registracije, pretraživanja dostupnih smještaja te stvaranja rezervacija.

4.4. Organizacija aplikacije

Aplikacija Quantum Hotel organizirana je prema višeslojnoj arhitekturi, koja osigurava modularnost, lakše održavanje i mogućnost proširenja sustava. Glavne komponente sustava čine frontend sloj (klijentski dio) i backend sloj (poslužiteljski dio), koji međusobno komuniciraju putem REST API sučelja.

Frontend sloj

Njegova glavna uloga je omogućiti korisnicima intuitivno, responzivno i moderno grafičko korisničko sučelje za korištenje aplikacije. Prema tome korisnicima pruža sve već navedene usluge pri čemu komunicira s backendom putem HTTP zahtjeva. Važno je napomenuti da sam frontend sloj nema izravan pristup bazi podataka već svi zahtjevi prolaze kroz backend sloj (osigurava validaciju i sigurnost). Frontend sloj u svrhu implementacije koristi React i Next.js tehnologije.

Backend sloj

Ovaj sloj zadužen je za obradu korisničkih zahtjeva, svu poslovnu logiku sustava i za rad s podacima (komunicira s bazom podataka). Implementira funkcionalnosti autentifikacije i autorizacije korisnika, upravljanja korisničkim profilima, dohvata podataka o smještajnim jedinicama, unos novih kategorija smještajnih jedinica, uvid u statistiku i ostale ključne funkcionalnosti. Backend sloj razvijen je s pomoću tehnologija Spring Boot okvira pri čemu s pomoću REST API-ja komunicira s frontend slojem. Organiziranost sloja definirana je MVC (Model – View – Controller) arhitekturom koja je objašnjena u nastavku.

MVC Arhitektura

Arhitektura sustava Quantum Hotel 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)

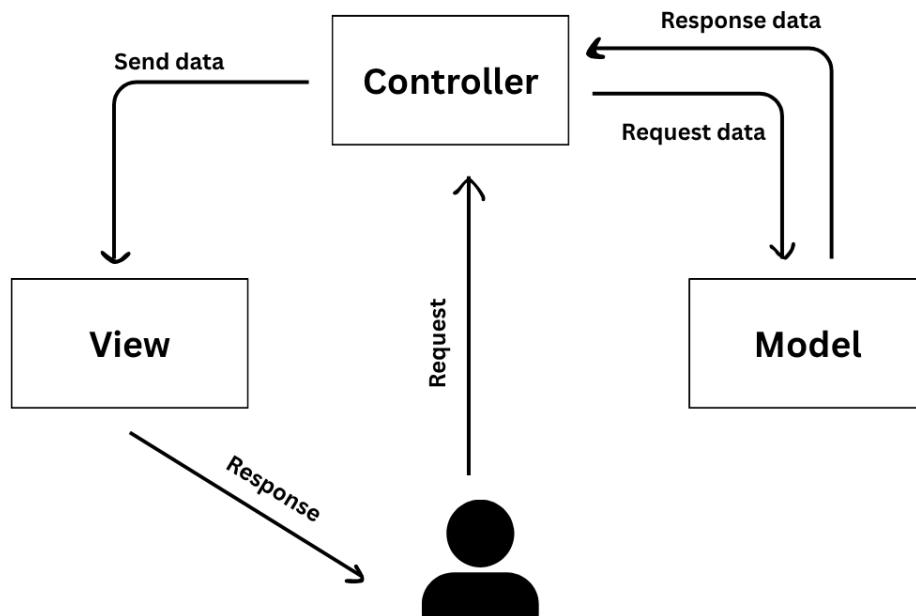
Model sloj predstavlja temelj aplikacije, zadužen za upravljanje podacima i poslovnim pravilima. U ovom sloju definiraju se klase koje odgovaraju entitetima u bazi podataka primjerice korisnik, rezervacija, smještajna jedinica, kategorija i dodatna usluga. Model komunicira s bazom podataka (PostgreSQL) te omogućuje dohvaćanje, pohranu i ažuriranje informacija na siguran i strukturiran način. Osim pristupa podacima, model može sadržavati i osnovnu poslovnu logiku koja se odnosi na te entitete (npr. provjera dostupnosti smještaja ili izračun cijene rezervacije).

Pogled (View sloj)

View sloj odgovoran je za vizualni prikaz podataka korisniku i interakciju s aplikacijom. U sustavu Quantum Hotel ulogu View sloja preuzima frontend aplikacija razvijena s pomoću React/Next.js tehnologija. Frontend dohvata podatke putem REST API-ja koje pruža backend te ih prikazuje u obliku responzivnog web sučelja prilagođenog različitim uređajima. Zadaća ovog sloja je prikazati podatke na jasan i intuitivan način, omogućujući korisnicima pregled dostupnih soba, stvaranje rezervacija i pregled profila, bez potrebe za izravnim kontaktom s poslužiteljem ili bazom podataka.

Upravljač (Controller sloj)

Controller sloj djeluje kao posrednik između pogleda i modela. On prima korisničke zahtjeve (npr. prijava, rezervacija, dohvati statistiku) s frontend aplikacije, proslijeđuje ih odgovarajućim servisima u model sloju te nakon obrade vraća rezultate natrag View sloju. Quantum Hotel sustav kontrolere implementira unutar Spring Boot okvira i definira API rute koje frontend koristi. Ovakva struktura omogućuje jasnu razdvojenost poslovne logike od prezentacijskog sloja, što znatno pojednostavljuje testiranje i nadogradnju aplikacije.



Slika 4.1: Prikaz MVC modela

4.5. 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 attribute koji ga karakteriziraju. U našoj bazi definirali smo 7 entiteta za potrebe uspješnog vođenja evidencije poslovanja i upravljanja podacima. Ti entiteti su redom:

- Users
- AccommodationCategory
- AccommodationUnit
- Reservation
- Amenities
- Articles
- FAQ

Osim entiteta, u bazi podataka postoji i tablica:

- ReservationAmenity

koja je nastala zbog n:n veze pa nije entitet, ali jest pomoćna tablica.

4.5.1 Opis tablica

Users

Ovaj entitet sadržava sve važne informacije o korisniku aplikacije. Objedinjuje uloge gosta, zaposlenika i administratora kroz atribut usr_role. Sadrži sljedeće attribute: usr_id, usr_first_name, usr_last_name, usr_username, usr_email, usr_password, usr_image_url, usr_enabled, usr_account_lock, usr_provider_id, usr_provider, usr_phone_number, usr_date_of_birth, usr_gender, usr_city, usr_role, usr_created, usr_last_updated, usr_last_update_id, usr_require_password_change, usr_email_verified, usr_deleted_at. Entitet ima One-to-many vezu sa samim sobom preko atributa usr_last_update_id zbog uređivanja profila.

Atribut	Tip podataka	Opis Varijable
usr_id	SERIAL	Jedinstveni identifikator korisnika (primarni ključ)
usr_first_name	VARCHAR(100)	Ime korisnika
usr_last_name	VARCHAR(100)	Prezime korisnika
usr_username	VARCHAR(64)	Korisničko ime
usr_email	VARCHAR(255)	E-mail adresa korisnika
usr_password	VARCHAR(100)	Korisnikova lozinka

usr_image_url	VARCHAR(100)	Slika profila korisnika
usr_enabled	BOOLEAN	Status aktivacije korisničkog računa
usr_account_lock	BOOLEAN	Zaključavanje u slučaju krive lozinke
usr_provider_id	VARCHAR(128)	Jedinstveni identifikator OAuth pružatelja
usr_provider	VARCHAR(128)	Naziv OAuth pružatelja
usr_phone_number	VARCHAR(50)	Broj mobitela korisnika
usr_date_of_birth	DATE	Datum rođenja korisnika
usr_gender	VARCHAR(16)	Spol korisnika (M, F, OTHER)
usr_city	VARCHAR(128)	Grad u kojem korisnik stanuje
usr_role	VARCHAR(50)	Uloga (Gost, Djelatnik, Administrator)
usr_created	TIMESTAMP	Trenutak kreiranja korisničkog profila
usr_last_updated	TIMESTAMP	Trenutak zadnjeg ažuriranja korisničkog profila
usr_last_update_id	INT	Identifikator korisnika koji je učinio zadnju promjenu (Strani ključ prema Users)
usr_require_password_change	BOOLEAN	Zahtjev za promjenom lozinke
usr_email_verified	BOOLEAN	Označava je li e-mail potvrđen
usr_deleted_at	TIMESTAMP	Vrijeme logičkog brisanja korisnika

AccommodationCategory

Ovaj entitet sadrži sve bitne informacije o kategorijama smještaja u hotelu. Ima sljedeće attribute: cat_id, cat_name, cat_units_number, cat_people_num, cat_twin_beds, cat_price, cat_check_in, cat_check_out, cat_image. Entitet ima One-to-many vezu s entitetom AccommodationUnit preko atributa cat_id i One-to-many vezu s entitetom Reservation također preko atributa cat_id.

Atribut	Tip podataka	Opis Varijable
cat_id	SERIAL	Jedinstveni identifikator kategorije smještaja (primarni ključ)
cat_name	VARCHAR(100)	Naziv kategorije
cat_units_number	INT	Broj soba koje se nude u kategoriji
cat_people_num	INT	Broj ljudi koje soba može primiti
cat_twin_beds	BOOLEAN	True = soba sadrži odvojene krevete, False = soba sadrži bračni krevet
cat_check_in	TIME	Standardno vrijeme prijave
cat_check_out	TIME	Standardno vrijeme odjave
cat_price	NUMERIC(10,5)	Cijena jednog noćenja
cat_image	VARCHAR(255)	Putanja do slike kategorije sobe

AccommodationUnit

Ovaj entitet sadrži sve bitne informacije o pojedinim smještajnim jedinicama. Ima sljedeće attribute: un_id, un_number, un_floor, un_status, cat_id. Entitet ima Many-to-one vezu s entitetom AccommodationCategory preko atributa cat_id te One-to-many vezu sa entitetom Reservation preko atributa un_id.

Atribut	Tip podataka	Opis Varijable
un_id	SERIAL	Jedinstveni identifikator smještajne jedinice (primarni ključ)
un_number	INT	Broj sobe
un_floor	INT	Kat na kojem se soba nalazi
un_status	BOOLEAN	Dostupnost sobe (false -> soba se ne nudi)
cat_id	INT	Strani ključ prema AccommodationCategory

Reservation

Ovaj entitet ovisi o korisniku (gostu) i trenutku kada je kreiran zahtjev. Ima sljedeće attribute: res_id, res_date_from, res_date_to, res_created, res_processed, res_check_in, res_check_out, res_status, guest_id, employee_id, cat_id, un_id. Entitet mora imati jedinstvenu kombinaciju (guest_id, res_created). Ima Many-to-one veze sa entitetima Users (za gosta i zaposlenika), AccommodationCategory i AccommodationUnit.

Atribut	Tip podataka	Opis Varijable
res_id	SERIAL	Jedinstveni identifikator rezervacije (primarni ključ)
res_date_from	DATE	Zatraženi datum početka rezervacije
res_date_to	DATE	Zatraženi datum završetka rezervacije
res_created	TIMESTAMP	Trenutak u kojem je kreiran zahtjev
res_processed	TIMESTAMP	Trenutak kada je zaposlenik obradio zahtjev
res_check_in	TIME	Vrijeme kada se prijavio gost
res_check_out	TIME	Vrijeme kada je gost napustio hotel
res_status	INT	Status (1=Prihvaćena, 0=U obradi, -1=Odbijena)
guest_id	INT	Identifikator gosta (Strani ključ prema Users)
employee_id	INT	Identifikator zaposlenika koji obrađuje (Strani ključ prema Users)
cat_id	INT	Identifikator kategorije (Strani ključ prema AccommodationCategory)
un_id	INT	Identifikator smještajne jedinice (Strani ključ prema AccommodationUnit)

Amenities

Ovaj entitet bilježi sve dodatne opcije koje gost može odabrati. Ima sljedeće atribute: amn_id, amn_name, amn_price. Ima Many-to-many vezu sa entitetom Reservation preko tablice ReservationAmenity.

Atribut	Tip podataka	Opis Varijable
amn_id	SERIAL	Jedinstveni identifikator dodatne opcije (primarni ključ)
amn_name	VARCHAR(100)	Naziv dodatne opcije
amn_price	NUMERIC(10,5)	Cijena dodatne opcije po noćenju

ReservationAmenity

Ovo je tablica koja opisuje Many-to-many vezu između entiteta Reservation i Amenities. Ima sljedeće atribute: res_id, amn_id i quantity.

Atribut	Tip podataka	Opis Varijable
res_id	INT	Jedinstveni identifikator rezervacije (strani ključ prema Reservation)
amn_id	INT	Jedinstveni identifikator dodatne opcije (strani ključ prema Amenities)
quantity	INT	Količina zahtijevane dodatne usluge

Articles

Ovaj entitet sadrži sve važne informacije o člancima koji se objavljaju na stranici. Sadrži atribute: art_id, art_title, art_created, art_edited, art_description, usr_id. Ima Many-to-one vezu sa entitetom Users preko usr_id koji predstavlja autora ili zadnjeg urednika.

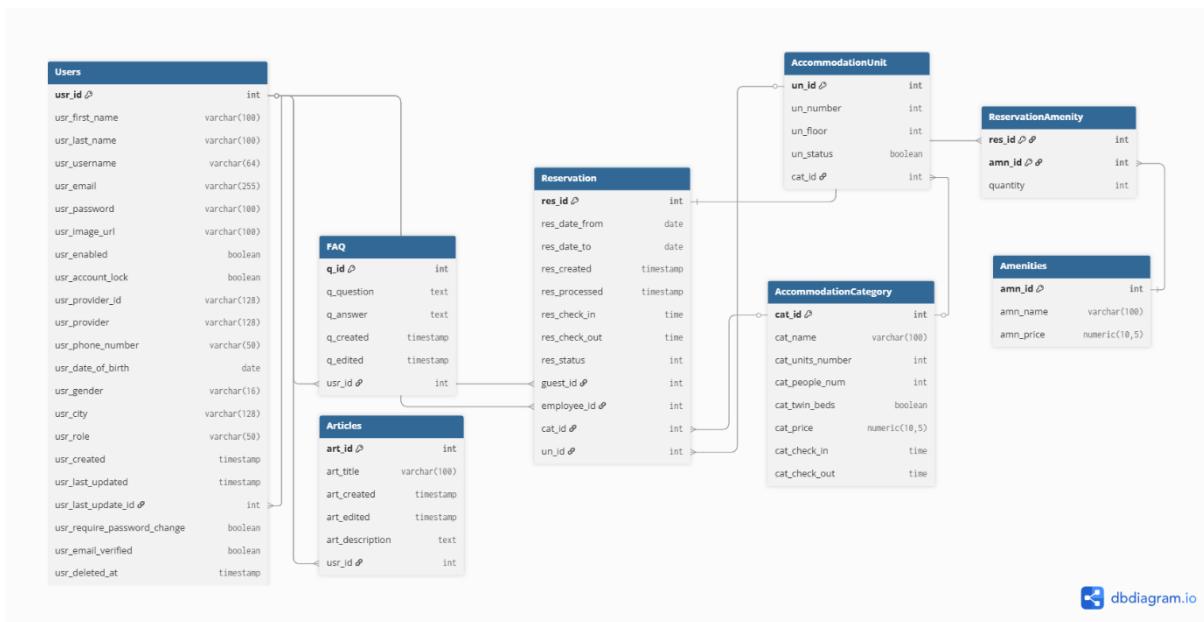
Atribut	Tip podataka	Opis Varijable
art_id	SERIAL	Jedinstveni identifikator članka (primarni ključ)
art_title	VARCHAR(100)	Naslov članka
art_created	TIMESTAMP	Trenutak objave članka
art_edited	TIMESTAMP	Trenutak zadnje promjene
art_description	TEXT	Tekst članka
usr_id	INT	Identifikator autora/urednika (Strani ključ prema Users)

FAQ

Ovaj entitet sadrži pitanja i odgovore za korisničku podršku. Sadrži sljedeće atribute: q_id, q_question, q_answer, q_created, q_edited, usr_id. Ima Many-to-one vezu sa entitetom Users preko usr_id.

Atribut	Tip podataka	Opis Varijable
q_id	SERIAL	Jedinstveni identifikator pitanja (primarni ključ)
q_question	TEXT	Tekst pitanja
q_answer	TEXT	Tekst odgovora
q_created	TIMESTAMP	Trenutak objave
q_edited	TIMESTAMP	Trenutak zadnje promjene
usr_id	INT	Identifikator zaposlenika (Strani ključ prema Users)

4.5.2. Dijagram baze podataka



Slika 4.2: Relacijski model baze podataka

4.6. Dijagram razreda



Slika 4.3: Dijagram razreda

Opis dijagrama

Na slici je prikazana skupina klasa i enumeracija koje zajedno čine osnovu sustava za autentifikaciju, autorizaciju i upravljanje korisnicima. Model korisnika predstavljen je klasom **User**, koja sadrži sva zajednička obilježja potrebna za rad s korisnicima neovisno o njihovoj ulozi u sustavu. Uloga korisnika definirana je enumeracijom **Role**, dok se način prijave razlikuje uz pomoć enumeracije **AuthProvider** koja omogućava razlikovanje lokalne prijave i prijave putem OAuth2 servisa (Google). Dodatno, spol korisnika definiran je enumeracijom **Gender**.

Prijava, registracija, verifikacija e-mail adrese i reset lozinke provode se putem **AuthController** klase. Ona komunicira s bazom podataka preko **UserRepository** te koristi **EmailService** za slanje verifikacijskih i reset poruka. Kako bi se osigurala sigurnost i kontrola pristupa, konfiguracija sigurnosnih pravila nalazi se u **SecurityConfig** klasi, dok je rad s vanjskim OAuth2 identitetima omogućen putem **CustomOidcUserService**.

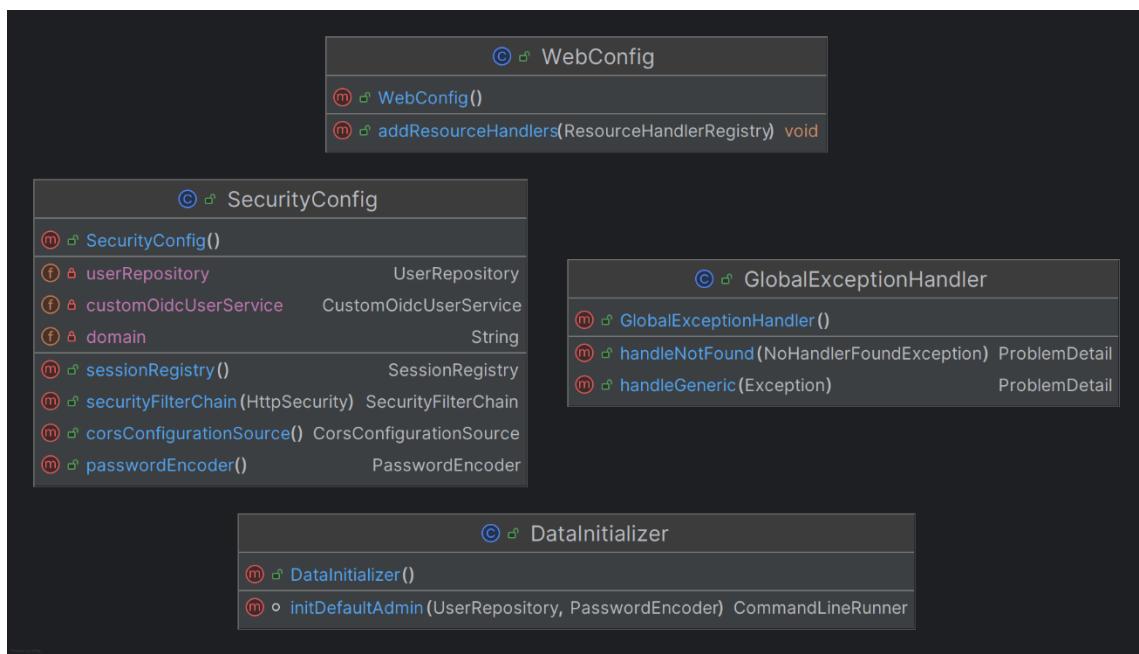
Nakon uspješne prijave, korisnik može pristupiti vlastitim podacima uz pomoć **UserController** klase koja vraća podatke u obliku DTO objekta **UserDto**. Dodatne funkcionalnosti nad korisnicima kao što su reset lozinke i kreiranje osoblja omogućene su preko **UserService** klase, koja koristi **UserRepository** za spremanje i dohvaćanje podataka iz PostgreSQL baze.

Administrator sustava ima pristup posebnim mogućnostima opisanima u **AdminController** klasi, poput kreiranja novih korisnika osoblja ili resetiranja njihovih lozinki. Za potrebe prikaza stranica unutar web sučelja privremeno se koriste metode unutar **PageController** klase. U slučaju pogrešaka u radu aplikacije, **CustomErrorController** i **GlobalExceptionHandler** pružaju jasne odgovore korisniku.

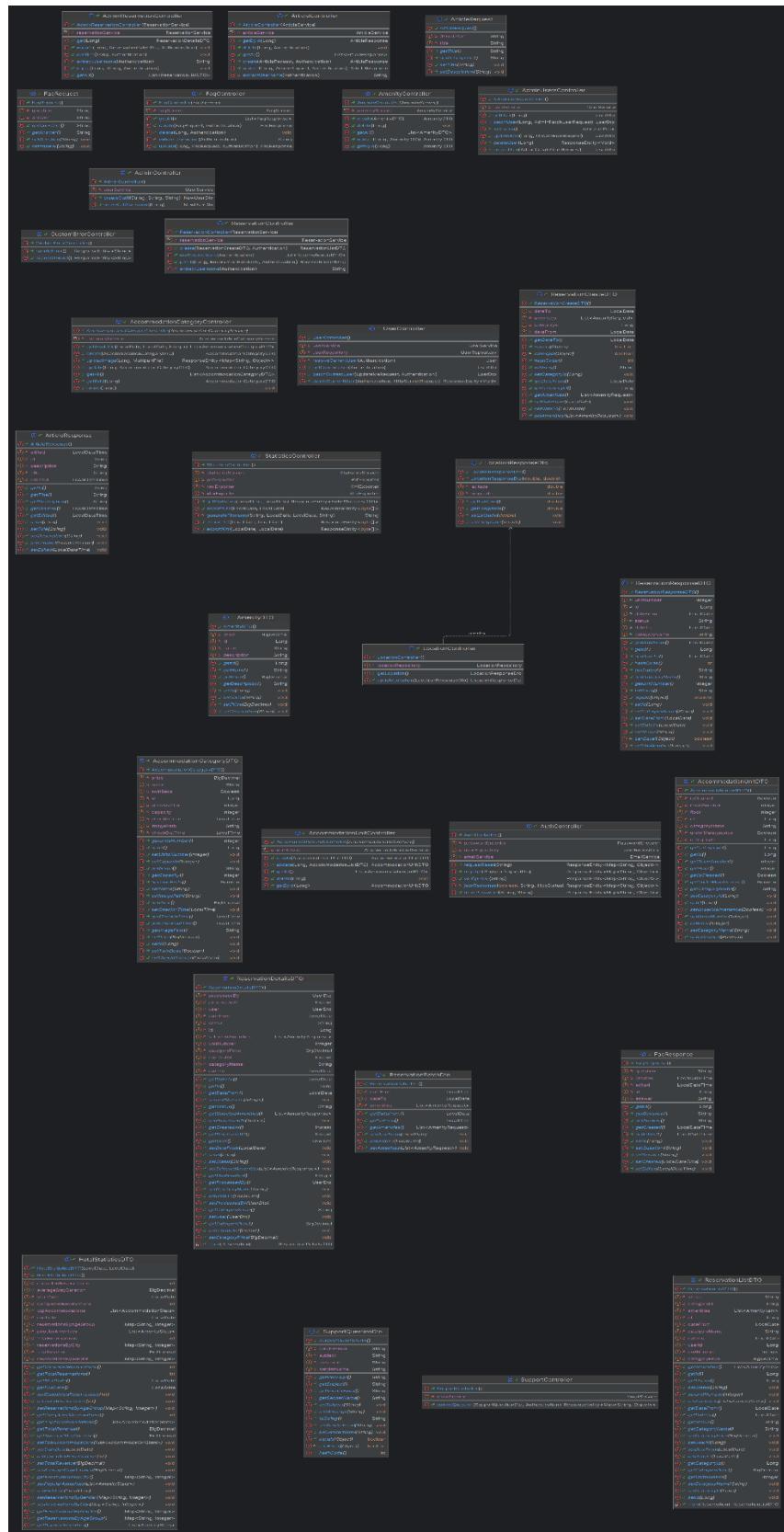
Kako bi se sustav inicijalno pripremio, definirana je klasa **DataInitializer** koja kreira zadane administrativne korisnike prilikom prvog pokretanja aplikacije. Konfiguracija resursa aplikacije (poput statičkih datoteka) nalazi se u **WebConfig** klasi.

Komunikacija između klijenta i poslužitelja olakšana je korištenjem DTO objekata, poput **RegisterRequestDto** za registraciju te **NewUserDto** za rad s korisnicima osoblja. Na taj se način osigurava čitljivost, sigurnost i jasan prijenos podataka unutar sustava.

Dijagram razreda prema funkcijama:



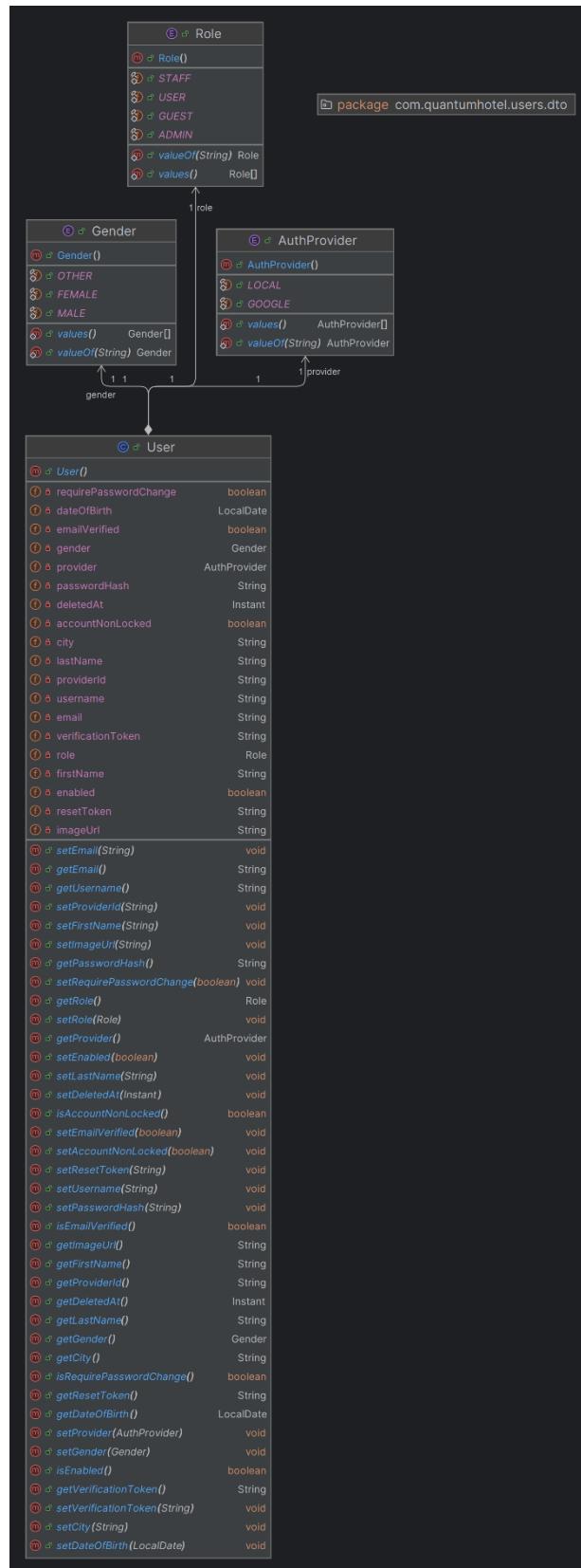
Slika 4.4: Dijagram razreda – konfiguracija



Slika 4.5: Dijagram razreda - kontroleri



Slika 4.6: Dijagram razreda - servisi



Slika 4.7: Dijagram razreda - korisnik

4.7. Dinamičko ponašanje aplikacije

Kako bismo stekli dublji uvid u cjelokupnu dinamiku sustava, koristimo nekoliko različitih UML dijagrama koji nam omogućuju vizualizaciju dinamičkog ponašanja aplikacije. Ovi prikazi obuhvaćaju ključne elemente poput promjena stanja, aktivnosti, događaja i procesa donošenja odluka. Jasno razumijevanje tih promjena presudno je za ispravan rad aplikacije jer otkriva precizan način na koji korisnici i komponente međusobno komuniciraju tijekom izvođenja sustava.

UML dijagrami stanja

Cilj dijagrama stanja je vizualizirati ponašanje objekta rezervacije sobe kroz faze njezinog životnog ciklusa. Dijagram pomaže u razumijevanju dinamičkog ponašanja sustava (ovisno o događajima i odlukama djelatnika) te u identifikaciji prijelaza i stanja koja zahtijevaju dodatnu obradu.

Dijagram stanja - Rezervacija sobe

Stanja

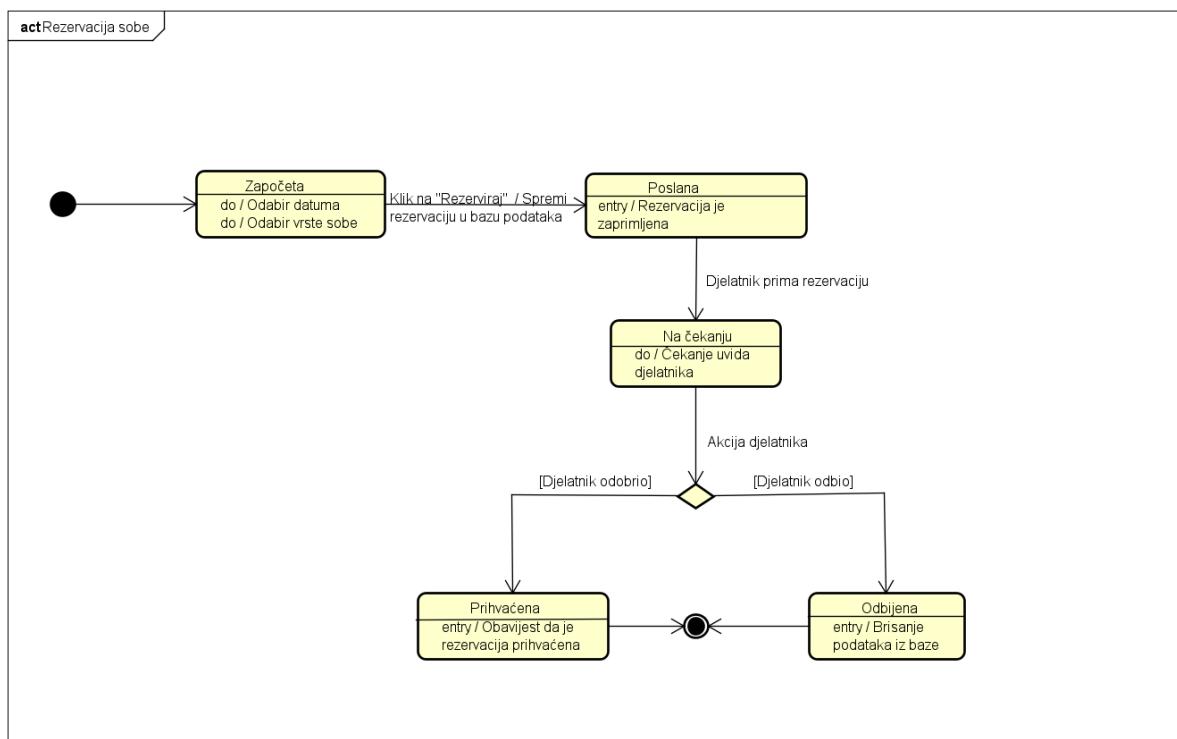
- **Započeta** – korisnik odabire datume i vrstu sobe.
- **Poslana** – rezervacija je zaprimljena (podaci su spremljeni u bazu).
- **Na čekanju** – rezervacija čeka uvid/odluku djelatnika.
- **Prihvaćena** – rezervacija je odobrena; korisniku se šalje obavijest.
- **Odbijena** – rezervacija je odbijena; podaci se brišu iz baze.

Prijelazi

- **Započeta → Poslana**
Potaknuto klikom na gumb „**Rezerviraj**“ / *Spremi rezervaciju u bazu podataka*.
- **Poslana → Na čekanju**
Potaknuto time da **djelatnik primi rezervaciju** (postaje dostupna za pregled).
- **Na čekanju → Prihvaćena**
Potaknuto odlukom **djelatnika da odobri rezervaciju** [Djelatnik odobrio].
- **Na čekanju → Odbijena**
Potaknuto odlukom **djelatnika da odbije rezervaciju** [Djelatnik odbio].

Opis životnog tijeka

Sljedeći dijagram prikazuje životni tijek **rezervacije sobe**. Proces započinje u stanju **Započeta**, gdje korisnik odabire datume i vrstu sobe. Klikom na gumb „**Rezerviraj**“ zahtjev se šalje, a podaci se spremaju u bazu podataka, čime rezervacija prelazi u stanje **Poslana** i smatra se zaprimljrenom. Nakon što djelatnik primi zahtjev, rezervacija prelazi u stanje **Na čekanju**, gdje ostaje sve dok djelatnik ne doneše odluku. Ovisno o odluci, rezervacija prelazi u stanje **Prihvaćena** (uz obavijest korisniku da je rezervacija odobrena) ili u stanje **Odbijena** (uz brisanje podataka o rezervaciji iz baze).



Slika 4.8: Dijagram stanja - Rezervacija sobe

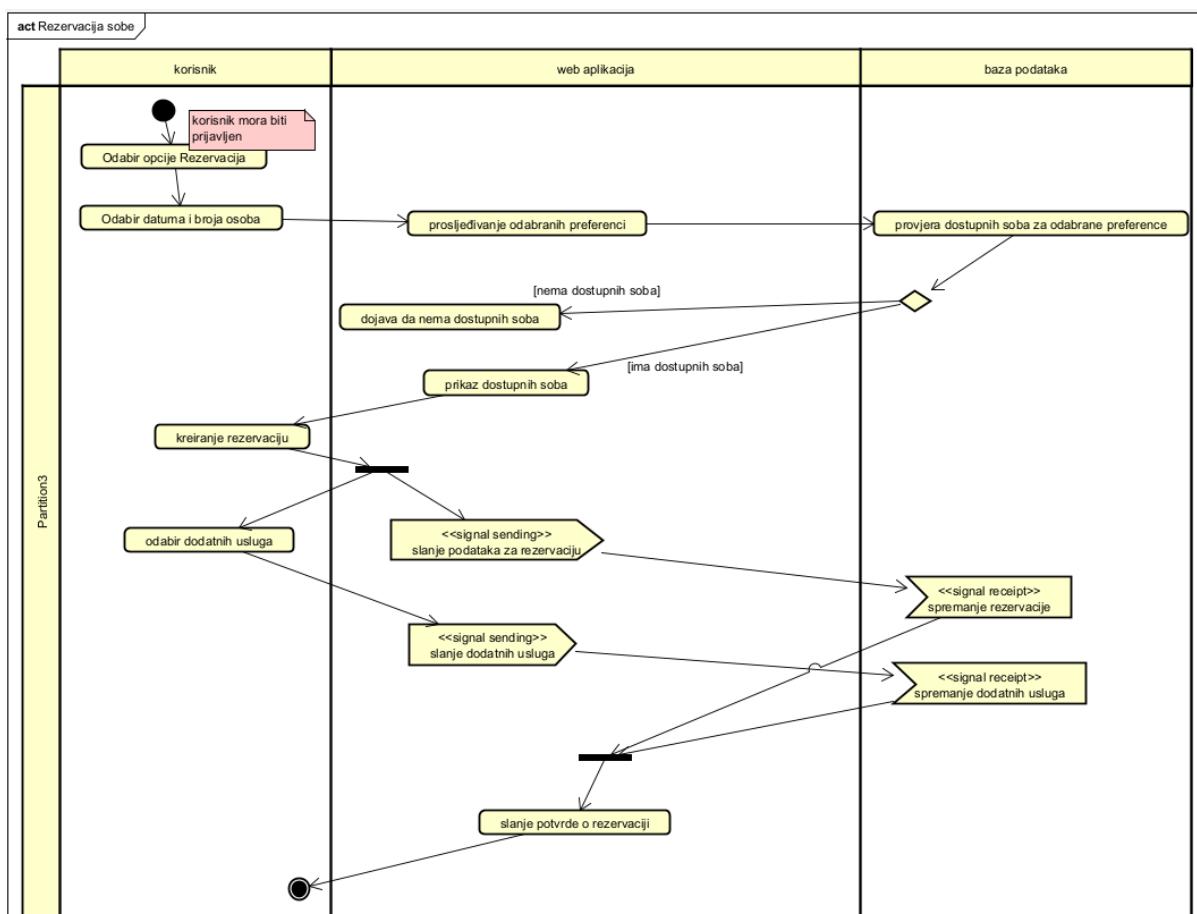
UML dijagrami aktivnosti

Dijagram aktivnosti služi za preciznu vizualizaciju tijeka upravljanja i podataka unutar određenog procesa. U nastavku teksta detaljno je prikazan i pojašnjen dijagram koji modelira redoslijed koraka prilikom rezervacije sobe.

Dijagram aktivnosti – Rezervacija sobe

Ovaj dijagram aktivnosti prikazuje tijek **rezervacije sobe u web aplikaciji hotela**, s jasno razdijeljenim odgovornostima između korisnika, web aplikacije i baze podataka. Proces započinje korisnikovom interakcijom sa sustavom i završava slanjem potvrde o uspješno kreiranoj rezervaciji. Na početku procesa korisnik mora biti **prijavljen u sustav**, što je preduvjet za nastavak rezervacije. Nakon toga korisnik odabire opciju *Rezervacija*, unosi željene datume boravka i broj osoba. Web aplikacija zatim prosljeđuje odabrane preferencije bazi podataka, gdje se provodi **provjera dostupnih**

soba za zadane kriterije. Ako u bazi podataka **ne postoje dostupne sobe**, web aplikacija obavještava korisnika da trenutno nema raspoloživih opcija. U suprotnom, ako su sobe dostupne, web aplikacija prikazuje popis dostupnih soba korisniku. Korisnik potom odabire željenu sobu i započinje proces **kreiranja rezervacije**. Nakon odabira sobe, korisnik ima mogućnost odabrati i **dodatne usluge** (npr. doručak, parking, wellness). Web aplikacija šalje podatke o rezervaciji i dodatnim uslugama bazi podataka putem odgovarajućih signala. Baza podataka po zaprimanju signala spremi podatke o rezervaciji i dodatnim uslugama. Po uspješnom spremanju svih potrebnih podataka, web aplikacija korisniku šalje **potvrdu o rezervaciji**, čime se proces rezervacije završava.

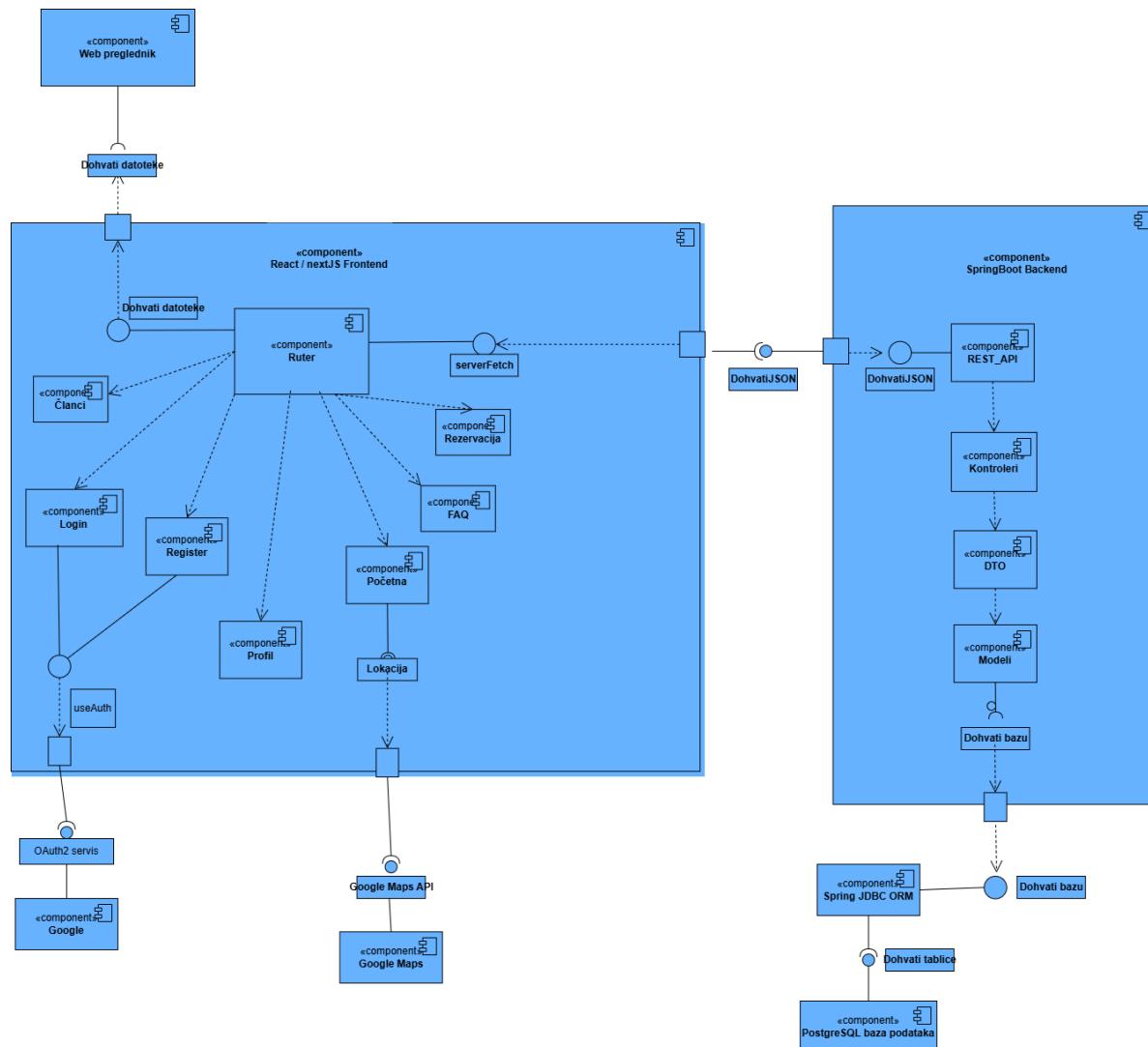


Slika 4.9: Dijagram aktivnosti – Rezervacija sobe

5. Arhitektura komponenata i razmještaja

5.1. Dijagram komponenata

Dijagram komponenti predstavlja strukturni i statički UML dijagram koji čini dio specifikacije arhitekture programske potpore. Prikazuje internu strukturu implementacijskih komponenti i njihove odnose s vanjskim sustavima. Glavni elementi ovog dijagrama su komponente (enkapsulirane cjeline programske potpore), sučelja (imenovani skup javno vidljivih atributa i apstraktnih operacija) te poveznice (spojnica, delegacija i ovisnost).



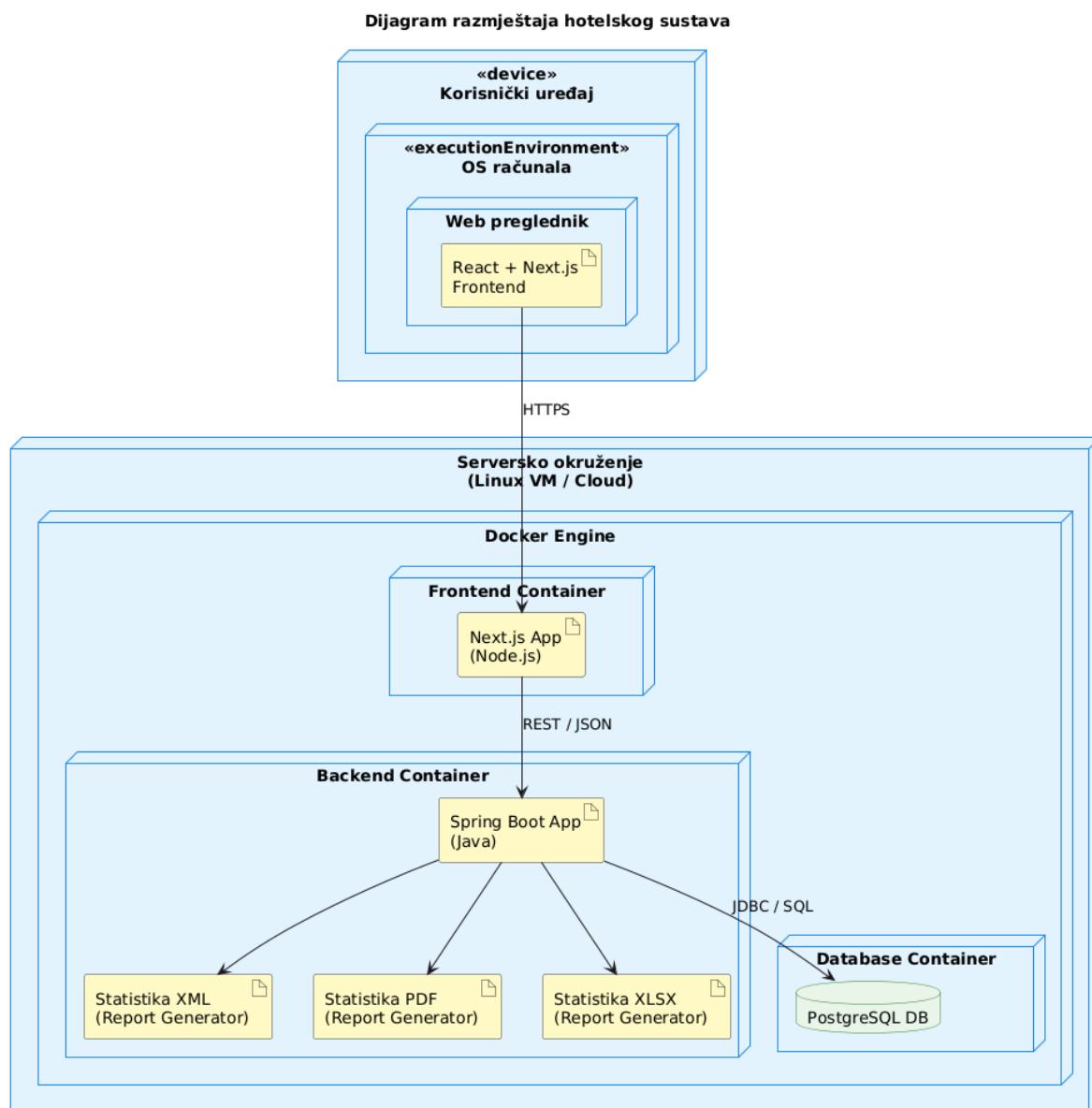
Slika 5.1 Dijagram komponenti

Dijagram prikazuje arhitekturu aplikacije koja se sastoji od frontenda (React+NextJS) i backenda (SpringBoot) te njihovu interakciju s vanjskim servisima i bazom. Web preglednik dohvaća datoteke potrebne za prikaz korisničkog sučelja. React ruter upravlja glavnim rutama unutar aplikacije (prijava, registracija, rezervacija, FAQ, profil, početna...) Kod prijave i registracije moguće je korištenje Google OAuth2 servisa te je na početnoj stranici omogućen prikaz lokacije (Google Maps) zbog čega su prikazane interakcije s odgovarajućim vanjskim servisima. Putem serverFetch poziva dohvaćaju se JSON podaci preko REST API-ja. Kontroleri upravljaju zahtjevima frontenda i proslijeđuju ih dalje u aplikaciju. Za prijenos podataka između

kontrolera i modela koristiti se DTO (Data Transfer Object), a modeli (entiteti) predstavljaju strukturu podataka koja se koristi u aplikaciji. Interakciju s bazom podataka nam omogućava Spring radni okvir, a kao relacijsku bazu za pohranu aplikacijskih podataka koristimo PostgreSQL bazu podataka.

5.2. Dijagram razmještaja

Dijagram razmještaja je strukturni statički UML dijagram koji opisuje topologiju sustava i usredotočen je na odnos sklopovskih i programskih dijelova. Prikazuje sklopovske komponente, komunikacijske puteve, smještaj i izvođenje programskih artefakata. Osnovni elementi ovog tipa dijagrama su čvorovi (npr. uređaj ili okolina izvođenja), artefakti (realizacije programskih komponenti, npr. dokument) i ovisnosti (prikazuju odnos između artefakata).



Slika 5.2: Dijagram razmještaja

Dijagram prikazuje fizičku arhitekturu hotelske aplikacije s klijentskim i poslužiteljskim okruženjem, koja komunicira putem HTTP/HTTPS protokola. Klijentsko okruženje uključuje korisnički uređaj s web preglednikom za izvršavanje frontend aplikacije razvijene u React i Next.js. Klijentski uređaj šalje zahtjeve poslužiteljskom okruženju, koje radi unutar Docker okruženja. Poslužitelj se sastoji od kontejnera za frontend (Next.js), backend (Spring Boot) i PostgreSQL bazu podataka. Spring Boot backend obrađuje REST API zahtjeve, upravlja poslovnom logikom i pristupa bazi podataka putem Spring Data JPA. Poslužitelj generira i statističke izvještaje u XML, PDF i XLSX formatima. Ova arhitektura omogućuje jasnu podjelu odgovornosti, lakše održavanje te skalabilnost i portabilnost sustava kroz Docker kontejnerizaciju.

6. Ispitivanje programskog rješenja

6.1. Ispitivanje komponenti

Ispitni slučaj 1: Testiranje funkcije servisa ArticleService

Opis: U ovom testu koristit ćemo paket Mockito za simuliranje rada repozitorija članaka (ArticleRepository) kako bismo mogli bez interakcije sa stvarnom bazom testirati logiku, odnosno funkcionalnost servisa članaka.

Ulazni podatci: Stvaramo testne instance objekata Article (a1 i a2) koje nam pomažu u simuliranju rada repozitorija članaka, pri čemu koristimo paket Mockito. Postavljamo atribute poput id, title, description i created za svaki članak.

Očekivani rezultat: Očekujemo da ćemo pri korištenju funkcije ArticleService-a getAll() dobiti listu odgovora članaka (List) te stoga vrijednost varijable result neće biti null. Dodatno očekujemo da lista sadrži točno 2 članka s odgovarajućim naslovima "Article 1" i "Article 2".

Rezultat: Provjera testa provjerava ispravnost poziva funkcije getAll() iz klase ArticleService na način da provjerava da rezultat nije null, da lista sadrži 2 elementa te da naslovi članaka odgovaraju očekivanim vrijednostima.

```

class ArticleServiceTest {
    @Mock 1 usage
    private ArticleRepository articleRepository;
    @Mock no usages
    private UserRepository userRepository;
    @InjectMocks 1 usage
    private ArticleService articleService;
    @BeforeEach new *
    void setUp() { MockitoAnnotations.openMocks( testClass: this); }
    @Test new *
    void shouldReturnAllArticles() {
        Article a1 = new Article();
        a1.setId(1L);
        a1.setTitle("Article 1");
        a1.setDescription("Desc 1");
        a1.setCreated(LocalDateTime.now());
        Article a2 = new Article();
        a2.setId(2L);
        a2.setTitle("Article 2");
        a2.setDescription("Desc 2");
        a2.setCreated(LocalDateTime.now());
        when(articleRepository.findAll()).thenReturn(List.of(a1, a2));
        List<ArticleResponse> result = articleService.getAll();
        assertNotEqual(result);
        assertEquals( expected: 2, result.size());
        assertEquals( expected: "Article 1", result.get(0).getTitle());
        assertEquals( expected: "Article 2", result.get(1).getTitle());
    }
}

```

Slika 6.1.1. ArticleServiceTest

Ispitni slučaj 2: Testiranje entiteta Faq

Opis: U ovom testu provjeravamo ispravnost postavljanja i dohvaćanja svojstava entiteta Faq (često postavljana pitanja). Radi se o unit testu koji provjerava ispravnost rada gettera i settera bez interakcije s bazom podataka.

Ulazni podatci: Stvaramo testnu instancu objekta Faq s postavljenim atributima: id (1L), question ("What is Quantum Hotel?"), answer ("A futuristic hotel."), createdAt, editedAt te povezanog korisnika (User) s korisničkim imenom "janedoe" koji je kreirao FAQ.

Očekivani rezultat: Očekujemo da će sve postavljene vrijednosti biti ispravno pohranjene i dohvaćene putem getter metoda. Objekt faq ne smije biti null, a svi atributi moraju odgovarati postavljenim vrijednostima.

Rezultat: Provedba testa provjerava da objekt faq nije null te da svi atributi (id, question, answer, createdAt, editedAt i createdBy) vraćaju točne vrijednosti koje su prethodno postavljene. Posebno se provjerava da korisničko ime kreatora FAQ-a odgovara "janedoe".

```

class FaqTest { new *

    private Faq faq; 14 usages
    private User user; 3 usages

    @BeforeEach new *
    void init() {
        user = new User();
        user.setUsername("janedoe");

        faq = new Faq();
        faq.setId(1L);
        faq.setQuestion("What is Quantum Hotel?");
        faq.setAnswer("A futuristic hotel.");
        faq.setCreatedAt(Instant.now());
        faq.setEditedAt(Instant.now());
        faq.setCreatedBy(user);
    }

    @Test new *
    void testFaqProperties() {
        assertNotNull(faq);
        assertEquals(expected: 1L, faq.getId());
        assertEquals(expected: "What is Quantum Hotel?", faq.getQuestion());
        assertEquals(expected: "A futuristic hotel.", faq.getAnswer());
        assertNotNull(faq.getCreatedAt());
        assertNotNull(faq.getEditedAt());
        assertEquals(expected: "janedoe", faq.getCreatedBy().getUsername());
    }
}

```

Slika 6.1.2: FaqTest

Ispitni slučaj 3: Svojstva entiteta AccommodationCategory

Opis: Jedinični test nad klasom AccommodationCategory kojim se provjerava ispravnost modela podataka i rad getter/setter metoda.

Ulagni podatci: Stvaramo objekt AccommodationCategory i ručno postavljamo ID (1L), naziv ("Luxury Suite"), cijenu (500.00) i kapacitet (4).

Očekivani rezultat: Sve get metode moraju vratiti točno one vrijednosti koje su prethodno proslijedene.

Rezultat: Test potvrđuje da se entitet ispravno sprema i da se ispravno obrađuju osnovni atributi.

```
@Test
void testCategoryProperties() {
    AccommodationCategory category = new AccommodationCategory();
    category.setId(1L);
    category.setName("Luxury Suite");
    category.setPrice(new BigDecimal("500.00"));
    category.setCapacity(4);

    assertEquals(expected: 1L, category.getId());
    assertEquals(expected: "Luxury Suite", category.getName());
    assertEquals(new BigDecimal("500.00"), category.getPrice());
    assertEquals(expected: 4, category.getCapacity());
}
```

Slika 6.1.3: AccommodationCategoryTest

Ispitni slučaj 4: Svojstva entiteta AccommodationUnit

Opis: Koristeći paket Mockito, simuliramo rad repozitorija kako bismo testirali funkcionalnost ispravnost modela podataka i rad getter/setter metoda te spremanje ispravnog modela u bazu.

Ulagni podatci: Stvaramo AccommodationUnit objekt s brojem sobe 101 i ID-em kategorije 1L što ručno upisujemo.

Očekivani rezultat: Očekujemo da će sve get() metode vratiti točno ove vrijednosti koje smo naveli kao ulazne podatke..

Rezultat: Test potvrđuje da se entitet ispravno sprema i da se ispravno obrađuju osnovni atributi.

```
@Test
void testUnitProperties() {
    AccommodationUnit unit = new AccommodationUnit();
    assertTrue(unit.getIsCleaned()); // default true
    assertFalse(unit.getUnderMaintenance()); // default false

    AccommodationCategory category = new AccommodationCategory();
    category.setName("Double Room");

    unit.setRoomNumber(101);
    unit.setCategory(category);

    assertEquals(expected: 101, unit.getRoomNumber());
    assertEquals(expected: "Double Room", unit.getCategory().getName());
}
```

Slika 6.1.4: AccommodationUnitTest

Ispitni slučaj 5: Dohvat svih kategorija smještaja

Opis: Testiramo AccommodationCategoryService kako bismo osigurali ispravnu transformaciju podataka iz baze u DTO objekte za prikaz.

Ulagni podatci: Pomoću Mockito okvira simuliramo da categoryRepository.findAll() vraća listu s jednom kategorijom ("Double Room").

Očekivani rezultat: Metoda getAll() mora vratiti listu koja nije prazna, sadrži točno jedan element i ispravno mapirano ime kategorije.

Rezultat: Provđena potvrđuje točnost mapiranja entiteta u DTO pri dohvatu svih unesenih kategorija.

```
    @Test new *
void getAllCategories() {
    AccommodationCategory category = new AccommodationCategory();
    category.setId(1L);
    category.setName("Double Room");

    when(categoryRepository.findAll()).thenReturn(List.of(category));

    List<AccommodationCategoryDTO> result = categoryService.getAll();

    assertFalse(result.isEmpty());
    assertEquals(expected: 1, result.size());
    assertEquals(expected: "Double Room", result.get(0).getName());
}
```

Slika 6.1.5: AccommodationCategoryGetAll

Ispitni slučaj 6: Dohvat nepostojeće kategorije

Opis: Test izazivanja pogreške radi provjere otpornosti sustava na nepostojeće identifikatore.

Ulagni podatci: Simuliramo poziv za nepostojeći ID 99L za koji repozitorij vraća Optional.empty().

Očekivani rezultat: Očekujemo da će servis baciti EntityNotFoundException.

Rezultat: Sustav ispravno detektira nedostatak podataka i obavještava o pogrešci putem iznimke.

```
    @Test new *
void getById_NotFound() {
    when(categoryRepository.findById(99L)).thenReturn(Optional.empty());

    assertThrows(EntityNotFoundException.class, () -> categoryService.getById(99L));
}
```

Slika 6.1.6: AccommodationCategoryInvalid

Ispitni slučaj 7: Kreiranje sobe za nepostojeću kategoriju

Opis: Ovaj test služi za izazivanje pogreške (*exception testing*) kako bismo provjerili otpornost sustava pri unosu neispravnih identifikatora.

Ulagni podatci: Kreiramo AccommodationUnitDTO s ID-jem kategorije 999L. Simuliramo repozitorij tako da za taj ID vrati prazan Optional.empty().

Očekivani rezultat: Očekujemo da će sustav baciti EntityNotFoundException. Dodatno, očekujemo da se metoda save() nikada ne pozove (never()).

Rezultat: Provedba testa osigurava da sustav ispravno sprječava unos soba s nepostojećim relacijama.

```
@Test
void create_InvalidCategory() {
    AccommodationUnitDTO dto = new AccommodationUnitDTO();
    dto.setCategoryId(999L);

    when(categoryRepository.findById(999L)).thenReturn(Optional.empty());

    assertThrows(EntityNotFoundException.class, () -> unitService.create(dto));
    verify(unitRepository, never()).save(any());
}
```

Slika 6.1.7: AccommodationUnitInvalidIdT

Ispitni slučaj 8: Ažuriranje statusa čišćenja

Opis: Testiramo metodu update unutar AccommodationUnitService kako bismo provjerili parcijalno ažuriranje podataka nad postojećom sobom.

Ulagni podatci: Kreiramo postojeću jedinicu kojoj je status isCleaned postavljen na false. Šaljemo updateDto u kojem taj status mijenjamo u true.

Očekivani rezultat: Očekujemo da će novonastali objekt imati status true te da će repozitorij primiti objekt na spremanje s ažuriranim podatkom.

Rezultat: Test potvrđuje da servisna logika uspješno modificira traženo polje i spremi promjene.

```

    void update_Status() {
        Long unitId = 1L;
        AccommodationUnit existingUnit = new AccommodationUnit();
        existingUnit.setId(unitId);
        existingUnit.setIsCleaned(false);

        AccommodationCategory category = new AccommodationCategory();
        category.setId(1L);
        existingUnit.setCategory(category);

        AccommodationUnitDTO updateDto = new AccommodationUnitDTO();
        updateDto.setIsCleaned(true);

        when(unitRepository.findById(unitId)).thenReturn(Optional.of(existingUnit));
        when(unitRepository.save(any())).thenAnswer(invocationOnMock -> invocation.getArgument(0));

        AccommodationUnitDTO result = unitService.update(unitId, updateDto);

        assertTrue(result.getIsCleaned());
        verify(unitRepository).save(existingUnit);
    }
}

```

Slika 6.1.8: AccommodationUnitStatusTest

Ispitni slučaj 9: Testiranje AdminReservationController kontrolera

Opis: U ovom testu koristit ćemo @WebMvcTest anotaciju i MockMvc za testiranje REST API endpointa administratorskog kontrolera rezervacija. Paket mockito koristimo za simuliranje rada servisa rezervacija (ReservationService) kako bismo testirali HTTP zahtjeve i odgovore bez pokretanja cijele aplikacije.

Ulagni podatci: Koristimo @MockBean za stvaranje mock instance ReservationService servisa. Testiramo različite endpoint-e: dohvaćanje svih rezervacija (GET /api/admin/reservations), dohvaćanje rezervacije po ID-u (GET /api/admin/reservations/{id}), potvrđivanje rezervacije (POST /api/admin/reservations/{id}/confirm), odbijanje rezervacije (POST /api/admin/reservations/{id}/reject) te testiranje neautoriziranog pristupa. Za sve testove koji zahtijevaju autentifikaciju koristimo @WithMockUser anotaciju s ADMIN ulogom.

Očekivani rezultat: Očekujemo da će svi endpointi vratiti odgovarajuće HTTP status kodove: 200 OK za uspješne zahtjeve i 3xx redirekciju za neautorizirane korisnike. Također očekujemo da će servisne metode biti pozvane s ispravnim parametrima.

Rezultat: Provedba testa provjerava ispravnost HTTP odgovora koristeći mockMvc.perform() i andExpect() metode te verificira da su odgovarajuće servisne metode pozvane pomoću verify() metode.

```

@WebMvcTest(AdminReservationController.class)
class AdminReservationControllerTest {
    @Autowired
    private MockMvc mockMvc;
    @MockBean 6 usages
    private ReservationService reservationService;
    @Test
    @WithMockUser(roles = {"ADMIN"})
    void shouldReturnAllReservations() throws Exception {
        when(reservationService.adminGetAll()).thenReturn(List.of(new ReservationListDTO()));
        mockMvc.perform(get( uriTemplate: "/api/admin/reservations"))
            .andExpect(status().isOk());
        verify(reservationService).adminGetAll();
    }
    @Test
    @WithMockUser(roles = {"ADMIN"})
    void shouldReturnReservationById() throws Exception {
        ReservationDetailsDTO dto = new ReservationDetailsDTO();
        when(reservationService.adminGet( id: 1L)).thenReturn(dto);
        mockMvc.perform(get( uriTemplate: "/api/admin/reservations/1"))
            .andExpect(status().isOk());
        verify(reservationService).adminGet( id: 1L);
    }
    @Test
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void shouldConfirmReservation() throws Exception {
        mockMvc.perform(post( uriTemplate: "/api/admin/reservations/1/confirm")
            .with(csrf()))
            .andExpect(status().isOk());
        verify(reservationService).confirm( id: 1L, username: "admin");
    }
}

```

Slika 6.1.9: AdminReservationController

Ispitni slučaj 10: Testiranje ArticleController kontrolera

Opis: U ovom testu koristit ćemo @SpringBootTest i @AutoConfigureMockMvc anotacije za integracijsko testiranje kontrolera članaka

Ulagni podatci: U @BeforeEach metodi pripremamo korisnika "admin" u bazi podataka ako već ne postoji. Stvaramo ArticleRequest objekt s naslovom "Test Article" i opisom "Test Description" te ga šaljemo kao JSON putem POST zahtjeva na endpoint /api/articles. Koristimo @WithMockUser s ulogom STAFF za simuliranje autentifikacije.

Očekivani rezultat: Očekujemo HTTP status 200 OK i da će odgovor sadržavati JSON objekt s točnim vrijednostima atributa title i description koje smo poslali u zahtjevu.

Rezultat: Provedba testa provjerava da je članak uspješno kreiran u bazi podataka i da odgovor sadrži očekivane vrijednosti koristeći jsonPath() metode za validaciju JSON odgovora.

```

class ArticleControllerTest {

    @BeforeEach
    void setup() {
        if (userRepository.findByUsername("admin").isEmpty()) {
            User user = new User();
            user.setUsername("admin");
            userRepository.save(user);
        }
    }

    @Test
    @WithMockUser(username = "admin", roles = {"STAFF"})
    void shouldCreateArticle() throws Exception {
        ArticleRequest request = new ArticleRequest();
        request.setTitle("Test Article");
        request.setDescription("Test Description");

        mockMvc.perform(post( UriTemplate: "/api/articles")
                .with(csrf())
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath( expression: "$.title").value( expectedValue: "Test Article"))
                .andExpect(jsonPath( expression: "$.description").value( expectedValue: "Test Description"));
    }
}

```

Slika 6.1.10: ArticleControllerTest

Ispitni slučaj 11: Testiranje FaqController kontrolera

Opis: U ovom testu koristit ćemo @SpringBootTest i @AutoConfigureMockMvc anotacije za integracijsko testiranje kontrolera često postavljanih pitanja (FAQ). Test provjerava funkcionalnost stvaranja novog FAQ unosa kroz REST API endpoint.

Ulagni podatci: Stvaramo FaqRequest objekt s pitanjem "What is Quantum Hotel?" i odgovorom "A futuristic hotel." koji šaljemo kao JSON putem POST zahtjeva na endpoint /api/faq. Koristimo @WithMockUser s ulogom ADMIN jer samo administratori mogu kreirati FAQ unose.

Očekivani rezultat: Očekujemo HTTP status 200 OK i da će JSON odgovor sadržavati točne vrijednosti atributa question i answer koje smo poslali u zahtjevu.

Rezultat: Provedba testa provjerava da je FAQ unos uspješno kreiran i da odgovor sadrži ispravne vrijednosti koristeći jsonPath() metode za validaciju JSON strukture.

```

class FaqControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void shouldCreateFaqAsAdmin() throws Exception {
        FaqRequest request = new FaqRequest();
        request.setQuestion("What is Quantum Hotel?");
        request.setAnswer("A futuristic hotel.");

        mockMvc.perform(post(uriTemplate("/api/faq")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(request)))
                .andExpect(status().isOk())
                .andExpect(jsonPath(expression: "$.question").value(expectedValue: "What is Quantum Hotel?"))
                .andExpect(jsonPath(expression: "$.answer").value(expectedValue: "A futuristic hotel.")));
    }

}

```

Slika 6.1.11: FaqControllerTest

Ispitni slučaj 12: Testiranje ReservationController kontrolera

Opis: U ovom testu koristit ćemo @WebMvcTest anotaciju i MockMvc za testiranje REST API endpointa kontrolera rezervacija. Paket Mockito koristimo za simuliranje rada servisa rezervacija kako bismo testirali dohvaćanje korisničkih rezervacija bez interakcije sa stvarnom bazom.

Ulazni podatci: Koristimo @MockBean za stvaranje mock instance ReservationService servisa. Stvaramo testni ReservationListDTO objekt s kategorijom "Deluxe" i statusom "PENDING". Simuliramo dohvaćanje rezervacija za korisnika "john" putem GET zahtjeva na endpoint /api/reservations/me. Koristimo @WithMockUser za simuliranje autentifikacije korisnika "john".

Očekivani rezultat: Očekujemo HTTP status 200 OK i da će servisna metoda findMine() biti pozvana s korisničkim imenom "john" te vratiti listu rezervacija.

Rezultat: Provedba testa provjerava da endpoint uspješno vraća korisnički specifične rezervacije koristeći mockMvc.perform() i andExpect() metode.

```

@WebMvcTest(ReservationController.class)
class ReservationControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean no usages
    private ReservationService reservationService;

    @Test
    @WithMockUser(username = "john")
    void shouldReturnMyReservations() throws Exception {
        ReservationListDTO dto = new ReservationListDTO();
        dto.setCategoryName("Deluxe");
        dto.setStatus("PENDING");

        when(reservationService.findMine(username: "john"))
            .thenReturn(List.of(dto));

        mockMvc.perform(get(uriTemplate: "/api/reservations/me"))
            .andExpect(status().isOk());
    }
}

```

Slika 6.1.12: ReservationControllerTest

Ispitni slučaj 13: Testiranje StatisticsController kontrolera

Opis: U ovom testu koristit ćemo @WebMvcTest anotaciju i MockMvc za testiranje REST API endpointa kontrolera statistike hotela. Testiramo dohvaćanje statistike te eksportiranje u različite formate (XML, PDF, XLSX) koristeći mock instance servisa i exportera.

Ulagni podatci: Koristimo @MockBean za stvaranje mock instanci StatisticsService, XmlExporter, PdfExporter i XlsxExporter komponenti. Testiramo četiri scenarija: (1) dohvaćanje statistike za period od 2024-01-01 do 2024-01-31, (2) eksportiranje u XML format, (3) eksportiranje u PDF format, (4) eksportiranje u XLSX format. Za sve testove koristimo @WithMockUser s ADMIN ili STAFF ulogama.

Očekivani rezultat: Očekujemo HTTP status 200 OK za sve zahtjeve te odgovarajuće Content-Type headere za svaki format izvoza: APPLICATION_XML za XML, APPLICATION_PDF za PDF i APPLICATION_OCTET_STREAM za XLSX format.

Rezultat: Provedba testa provjerava da svi endpoint-i vraćaju odgovarajuće HTTP odgovore i Content-Type headere te da su servisne i exporter metode pozvane s ispravnim parametrima.

```

class StatisticsControllerTest {
    @Test
    @WithMockUser(roles = "ADMIN")
    void shouldReturnStatistics() throws Exception {
        HotelStatisticsDTO dto = new HotelStatisticsDTO();
        when(statisticsService.generateStatistics(
            LocalDate.parse( text: "2024-01-01"),
            LocalDate.parse( text: "2024-01-31"))
        ).thenReturn(dto);

        mockMvc.perform(get( uriTemplate: "/api/statistics")
                .param( name: "startDate", ...values: "2024-01-01")
                .param( name: "endDate", ...values: "2024-01-31"))
                .andExpect(status().isOk());
    }

    @Test
    @WithMockUser(roles = {"ADMIN", "STAFF"})
    void shouldExportXml() throws Exception {
        HotelStatisticsDTO dto = new HotelStatisticsDTO();
        byte[] xmlData = "xml-content".getBytes();

        when(statisticsService.generateStatistics(any(), any()))
            .thenReturn(dto);

        when(xmlExporter.export(dto)).thenReturn(xmlData);

        mockMvc.perform(get( uriTemplate: "/api/statistics/export/xml")
                .param( name: "startDate", ...values: "2024-01-01")
                .param( name: "endDate", ...values: "2024-01-31"))
                .andExpect(status().isOk())
                .andExpect(content().contentType(MediaType.APPLICATION_XML));
    }
}

```

Slika 6.1.13: StatisticsControllerTest

Ispitni slučaj 14: Testiranje entiteta Reservation

Opis: U ovom testu provjeravamo ispravnost postavljanja i dohvaćanja svojstava entiteta Reservation (rezervacija). Radi se o unit testu koji provjerava ispravnost rada gettera i setтера bez interakcije s bazom podataka.

Ulagni podatci: U @BeforeEach metodi stvaramo testnu instancu objekta User s korisničkim imenom "testuser" te instancu objekta Reservation s postavljenim atributima: id (1L), dateFrom (trenutni datum), dateTo (trenutni datum + 2 dana), status (PENDING) i povezanog korisnika.

Očekivani rezultat: Očekujemo da će sve postavljene vrijednosti biti ispravno pohranjene i dohvaćene putem getter metoda. Objekt reservation ne smije biti null, a svi atributi moraju odgovarati postavljenim vrijednostima.

Rezultat: Provedba testa provjerava da objekt reservation nije null te da svi atributi (id, status, user, createdAt) vraćaju točne vrijednosti koje su prethodno postavljene. Posebno se provjerava da status rezervacije odgovara PENDING te da korisničko ime povezanog korisnika odgovara "testuser".

```

class ReservationTest {

    private Reservation reservation; no usages
    private User user; no usages

    @BeforeEach
    void setup() {
        user = new User();
        user.setUsername("testuser");

        reservation = new Reservation();
        reservation.setId(1L);
        reservation.setDateFrom(LocalDate.now());
        reservation.setDateTo(LocalDate.now().plusDays( daysToAdd: 2));
        reservation.setUser(user);
        reservation.setStatus(ReservationStatus.PENDING);
    }

    @Test
    void testReservationProperties() {
        assertNotNull(reservation);
        assertEquals( expected: 1L, reservation.getId());
        assertEquals(ReservationStatus.PENDING, reservation.getStatus());
        assertEquals( expected: "testuser", reservation.getUser().getUsername());
        assertNotNull(reservation.getCreatedAt());
    }
}

```

Slika 6.1.14: ReservationTest

Napomena:

Za sve sljedeće testove koristi se **base user** prikazan na slici ispod, koji služi kao osnovna testna instanca korisnika.

```

private User baseUser() { 5 usages new *
    User u = new User();
    u.setUsername("user1");
    u.setEmail("user@test.com");
    u.setRole(Role.USER);
    u.setEnabled(true);
    u.setDateOfBirth(LocalDate.of( year: 1995, month: 1, dayOfMonth: 1));
    return u;
}

```

Slika 6.1.15: Base user korišten u svim UserService testovima

Ispitni slučaj 15: Djelomično ažuriranje vlastitog profila

Opis:

Testira se PATCH funkcionalnost metode updateMe, gdje se ažuriraju samo poslani atributi.

Ulazni podatci:

Korisnik i zahtjev koji sadrži novo ime "Ivana" i grad "Zagreb".

Očekivani rezultat:

Ažurirani atributi moraju biti promijenjeni, dok ostali ostaju neizmijenjeni.

Rezultat:

Test potvrđuje da su ime i grad uspješno ažurirani u vraćenom UserDto objektu.

```
@Test new *
void updateMe_partialUpdate_success() {
    User user = baseUser();
    UserService.UpdateMeRequest req = new UserService.UpdateMeRequest();
    req.firstName = "Ivana";
    req.city = "Zagreb";

    when(userRepository.save(user)).thenReturn(user);

    UserDto dto = userService.updateMe(user, req);

    assertEquals( expected: "Ivana", dto.firstName());
    assertEquals( expected: "Zagreb", dto.city());
}
```

Slika 6.1.16: Djelomično ažuriranje vlastitog profila

Ispitni slučaj 16: Povreda jedinstvenosti pri ažuriranju profila

Opis:

Ovim testom provjerava se rukovanje greškom pri kršenju jedinstvenih ograničenja (email/username).

Ulazni podatci:

Zahtjev s duplicitiranim emailom.

Očekivani rezultat:

Baca se ResponseStatusException s HTTP statusom 409 (CONFLICT).

Rezultat:

Test potvrđuje da se ispravna HTTP greška vraća klijentu.

```
@Test new *
void updateMe_uniqueConstraintViolation_throws409() {
    User user = baseUser();
    UserService.UpdateMeRequest req = new UserService.UpdateMeRequest();
    req.email = "duplicate@mail.com";

    when(userRepository.save(user))
        .thenThrow(DataIntegrityViolationException.class);

    ResponseStatusException ex = assertThrows(
        ResponseStatusException.class,
        () -> userService.updateMe(user, req)
    );

    assertEquals(HttpStatus.CONFLICT, ex.getStatusCode());
}
```

Slika 6.1.17: Povreda jedinstvenosti pri ažuriranju profila

Ispitni slučaj 17: Brisanje vlastitog korisničkog računa (soft delete)

Opis:

Provjerava se soft delete funkcionalnost metode deleteMe.

Ulazni podatci:

Postojeći korisnik.

Očekivani rezultat:

Korisnik se onemogućuje (enabled = false) i postavlja se vrijeme brisanja.

Rezultat:

Test potvrđuje da je korisnik onemogućen i da je deletedAt ispravno postavljen.

```
@Test new *
void deleteMe_softDelete() {
    User user = baseUser();

    userService.deleteMe(user);

    assertFalse(user.isEnabled());
    assertNotNull(user.getDeletedAt());
}
```

Slika 6.1.18: Brisanje vlastitog korisničkog računa (soft delete)

Ispitni slučaj 18: Admin kreira LOCAL korisnika bez lozinke

Opis:

Testira se validacija obavezne lozinke za LOCAL korisnike.

Ulazni podatci:

Zahtjev bez lozinke i s providerom LOCAL.

Očekivani rezultat:

Baca se ResponseStatusException s HTTP statusom 400 (BAD REQUEST).

Rezultat:

Test potvrđuje ispravnu validaciju ulaznih podataka.

```
@Test new *
void adminCreateUser_localWithoutPassword_throws400() {
    UserService.AdminCreateUserRequest req =
        new UserService.AdminCreateUserRequest();
    req.username = "test";
    req.provider = AuthProvider.LOCAL;

    ResponseStatusException ex = assertThrows(
        ResponseStatusException.class,
        () -> userService.adminCreateUser(req)
    );

    assertEquals(HttpStatus.BAD_REQUEST, ex.getStatusCode());
}
```

Slika 6.1.19: Admin kreira LOCAL korisnika bez lozinke 1

Ispitni slučaj 19: Admin uspješno kreira korisnika

Opis:

Provjerava se ispravno kreiranje korisnika putem administratorske metode.

Ulazni podatci:

Korisničko ime "admin1", lozinka "secret" i uloga ADMIN.

Očekivani rezultat:

Korisnik se spremi i vraća se odgovarajući UserDto.

Rezultat:

Test potvrđuje da su korisničko ime i uloga ispravno postavljeni.

```
@Test new *
void adminCreateUser_success() {
    UserService.AdminCreateUserRequest req =
        new UserService.AdminCreateUserRequest();
    req.username = "admin1";
    req.password = "secret";
    req.role = Role.ADMIN;

    when(userRepository.save(any(User.class)))
        .thenAnswer(InvocationOnMock invocation -> invocation.getArgument(0));

    UserDto dto = userService.adminCreateUser(req);

    assertEquals(expected: "admin1", dto.username());
    assertEquals(expected: "ADMIN", dto.role());
}
```

Slika 6.1.20: Admin uspješno kreira korisnika

Ispitni slučaj 20: Admin djelomično ažurira korisnika

Opis:

Testira se PATCH ažuriranje korisnika od strane administratora.

Ulazni podatci:

Zahtjev s novim gradom "Split" i spolom FEMALE.

Očekivani rezultat:

Atributi korisnika se ispravno ažuriraju.

Rezultat:

Test potvrđuje da vraćeni DTO sadrži ažurirane vrijednosti.

```
@Test new *
void adminPatchUser_updatesFields() {
    User user = baseUser();
    user.setId(1L);

    UserService.AdminPatchUserRequest req =
        new UserService.AdminPatchUserRequest();
    req.city = "Split";
    req.gender = Gender.FEMALE;

    when(userRepository.findById(1L))
        .thenReturn(Optional.of(user));

    when(userRepository.save(user))
        .thenReturn(user);

    UserDto dto = userService.adminPatchUser( id: 1L, req);

    assertEquals( expected: "Split", dto.city());
    assertEquals( expected: "FEMALE", dto.gender());
}
```

Slika 6.1.21: Admin djelomično ažurira korisnika

Ispitni slučaj 21: Admin mijenja ulogu korisnika

Opis:

Provjerava se mogućnost promjene uloge korisnika.

Ulazni podatci:

ID korisnika i nova uloga ADMIN.

Očekivani rezultat:

Uloga korisnika se ažurira i vraća u DTO objektu.

Rezultat:

Test potvrđuje uspješnu promjenu uloge.

```
@Test new *
void adminUpdateRole_success() {
    User user = baseUser();
    user.setId(1L);

    when(userRepository.findById(1L))
        .thenReturn(Optional.of(user));
    when(userRepository.save(user))
        .thenReturn(user);

    UserDto dto = userService.adminUpdateRole( id: 1L, Role.ADMIN);

    assertEquals( expected: "ADMIN", dto.role());
}
```

Slika 6.1.22: Admin mijenja ulogu korisnika

Ispitni slučaj 22: Admin mijenja ulogu bez zadane vrijednosti

Opis:

Testira se validacija obaveznog parametra role.

Ulazni podatci:

Vrijednost uloge je null.

Očekivani rezultat:

Baca se ResponseStatusException s HTTP statusom 400 (BAD REQUEST).

Rezultat:

Test potvrđuje da se neispravan zahtjev pravilno odbacuje.

```
@Test new *
void adminUpdateRole_nullRole_throws400() {
    assertThrows(ResponseStatusException.class,
        () -> userService.adminUpdateRole( id: 1L, role: null));
}
```

Slika 6.1.23: Admin mijenja ulogu bez zadane vrijednosti

6.2. Ispitivanje sustava

Alati za ispitivanje sustava:

Selenium WebDriver

- Omogućuje pisanje automatiziranih end-to-end testova (npr. Java + JUnit).
- Podržava rad s više preglednika (Chrome/Firefox) i integraciju s CI okruženjem.

Testno okruženje

- OS: Windows 11
- WebDriver: ChromeDriver (verzija: 144.0.7559.96)
- Jezik/okvir: Java + JUnit 5
- Pokretanje sustava: Docker Compose / lokalno
- URL aplikacije (frontend): <http://localhost:3000>
- URL backend API: <http://localhost:8080>

SYS-01 – Prijava putem Google OAuth2 (redovni slučaj)

```
@Test
void SYS01_googleOAuth2Redirect_shouldNavigateToGoogleOrOauthEndpoint() {
    openApp(FrontendRoutes.login());

    click(By.cssSelector(".google-button2"));

    String url = driver.getCurrentUrl();

    boolean ok = url.contains("accounts.google.com")
        || url.contains("/oauth2/authorization/google")
        || url.toLowerCase().contains("oauth2");

    assertTrue(ok, "Očekivan OAuth2 redirect. Dobiveno: " + url);
}
```

Slika 6.2.1. SYS-01

Cilj: Provjeriti da klik na prijavu pokreće OAuth2 tok i da se korisnik nakon uspješne prijave vraća u aplikaciju kao prijavljen korisnik.

Ulazi:

- Test Google račun (npr. test.user@gmail.com)
- Korisnička akcija: klik na gumb “Prijava s Google”

Koraci ispitivanja (Selenium):

1. Otvoriti početnu stranicu aplikacije: BASE_URL.
2. Pričekati da se učita početni UI (npr. hero sekcija ili header).
3. Kliknuti na gumb “Prijava s Google”.
4. Verificirati da je korisnik preusmjeren na Google login domenu (accounts.google.com).
5. (Ako se test izvodi kao “pravi E2E”): unijeti email i lozinku test računa i potvrditi prijavu.
6. Verificirati povratak u aplikaciju i prikaz korisničkog stanja (npr. avatar, “Odjava”, “Moj profil”, ili token-cookie prisutan).

Očekivani izlaz:

- Preusmjerenje na Google OAuth2 prijavu.
- Nakon uspješne prijave korisnik je prijavljen i vidi elemente za prijavljenog korisnika.

Dobiveni izlaz:

Test set: com.quantumhotel.SYS01_OAuth2RedirectTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.986 s -- in
com.quantumhotel.SYS01_OAuth2RedirectTest (**PASS**)

SYS-02 — Kreiranje rezervacije s dodatnim uslugama (redovni slučaj)

```
@Test
void SYS02_createReservation_withExtras_shouldSucceed() throws InterruptedException {
    ensureAuthenticatedIfPossible();

    openApp(FrontendRoutes.reservations());

    By dateFrom = By.id("dateFrom");
    By dateTo   = By.id("dateTo");
    By persons  = By.id("persons");

    WebElement from = visible(dateFrom);
    from.click();
    from.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    from.sendKeys("06-02-2026");
    WebElement to = visible(dateTo);
    to.click();
    to.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    to.sendKeys("07-02-2026");
    WebElement person = visible(persons);
    person.click();
    person.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    person.sendKeys("2");

    click(By.cssSelector("#dostupneSobeBtn"));
    click(By.cssSelector("#room-1"));
    click(By.cssSelector("#addon-1"));
    click(By.cssSelector("#rezervirajBtn"));

    boolean successToast = !driver.findElements(By.cssSelector("#resultMessage")).contains("Rezervacija uspješno kreirana!");
    assertTrue(successToast, "Očekivana potvrda rezervacije.");
}
```

Slika 6.2.2. SYS-02

Cilj: Provjeriti end-to-end tok rezervacije: od pregleda soba do potvrde rezervacije i prikaza u “Moje rezervacije”.

Preduvjeti:

- U sustavu postoje dostupne sobe i opcionalne dodatne usluge (npr. doručak).
- Priložen JSESSIONID putem -DSESSIONID=

Ulazi:

- Datum dolaska: 06-02-2026
- Datum odlaska: 07-02-2026
- Broj osoba: 2
- Dodatne usluge: Doručak (prva ponuđena)

Koraci ispitivanja (Selenium):

1. Otvoriti stranicu pregleda smještaja (/reservations).
2. Unijeti datume i broj osoba u formu (["dateFrom"], ["dateTo"], ["persons"]).
3. Kliknuti "Prikaži dostupne sobe" (["dostupneSobeBtn"]).
4. Na koraku dodatnih usluga označiti Doručak (["addon-1"]`).
5. Odabratи jednu dostupnu sobu (prvi rezultat ["room-1"]).
6. Kliknuti "Rezerviraj" (["rezervirajBtn"]).
7. Verificirati poruku uspjeha (resultMessage).

Očekivani izlaz:

- Rezervacija je kreirana bez greške.

Dobiveni izlaz:

Test set: com.quantumhotel.SYS02_CreateReservationWithExtrasTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.567 s -- in
com.quantumhotel.SYS02_CreateReservationWithExtrasTest (**PASS**)

SYS-03- Neispravan raspon datuma (rubni uvjet)

```
@Test
void SYS03_invalidDateRange_shouldShowValidationError() {
    ensureAuthenticatedIfPossible();

    openApp(FrontendRoutes.reservations());

    By dateFrom = By.id("dateFrom");
    By dateTo   = By.id("dateTo");
    By persons  = By.id("persons");

    // odabrati datum za koji znamo da vec postoji rezervacija
    WebElement from = visible(dateFrom);
    from.click();
    from.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    from.sendKeys("29-01-2026");
    WebElement to = visible(dateTo);
    to.click();
    to.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    to.sendKeys("28-01-2026");
    WebElement person = visible(persons);
    person.click();
    person.sendKeys(Keys.chord(Keys.CONTROL, "a"));
    person.sendKeys("2");

    click(By.cssSelector("#dostupneSobeBtn"));

    boolean hasError = !driver.findElements(By.cssSelector("#resultMessage")).contains("Greška pri dohvaćanju dost");
    assertTrue(hasError, "Očekivana greška rezervacije.");
}
```

Slika 6.2.3. SYS-03

Cilj: Provjeriti validaciju unosa kada je datum odlaska prije datuma dolaska (granični ulaz).

Preduvjeti:

- Priložen JSESSIONID putem -DSESSIONID=

Ulazi:

- Datum dolaska: 29-01-2026
- Datum odlaska: 28-01-2026 (namjerno neispravno)
- Broj osoba: 2

Koraci ispitivanja (Selenium):

1. Otvoriti stranicu pretrage soba (/reservations).
2. Unijeti dateFrom=29-01-2026, dateTo=28-01-2026.
3. Kliknuti "Prikaži dostupne sobe".
4. Verificirati da se:
 - prikaže validacijska poruka ("Datum odlaska mora biti nakon datuma dolaska.")

Očekivani izlaz:

- Sustav **ne** šalje zahtjev za pretragom / rezervacijom s neispravnim datumima.
- Korisnik vidi jasnu poruku o grešci.

Dobiveni izlaz:

```
-----  
Test set: com.quantumhotel.SYS03_InvalidDateRangeTest  
-----
```

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.708 s -- in  
com.quantumhotel.SYS03_InvalidDateRangeTest (PASS)
```

SYS-04 — Preklapanje rezervacije / nedostupna soba (rubni uvjet)



```
@Test  
void SYS04_reservationOverlap_shouldBeBlocked() {  
    ensureAuthenticatedIfPossible();  
  
    openApp(FrontendRoutes.reservations());  
  
    By dateFrom = By.id("dateFrom");  
    By dateTo = By.id("dateTo");  
    By persons = By.id("persons");  
  
    // odabrati datum za koji znamo da vec postoji rezervacija  
    WebElement from = visible(dateFrom);  
    from.click();  
    from.sendKeys(Keys.chord(Keys.CONTROL, "a"));  
    from.sendKeys("06-02-2026");  
    WebElement to = visible(dateTo);  
    to.click();  
    to.sendKeys(Keys.chord(Keys.CONTROL, "a"));  
    to.sendKeys("07-02-2026");  
    WebElement person = visible(persons);  
    person.click();  
    person.sendKeys(Keys.chord(Keys.CONTROL, "a"));  
    person.sendKeys("2");  
  
    click(By.cssSelector("#dostupneSobeBtn"));  
  
    boolean hasError = !driver.findElements(By.cssSelector("#resultMessage")).contains("Nema dostupnih soba za oda  
assertTrue(hasError, "Očekivano nema dostupnih soba");  
}
```

Slika 6.2.4. SYS-04

Cilj: Provjeriti da sustav sprječava preklapanje rezervacija i ne dopušta rezerviranje zauzete sobe za isti termin.

Preduvjeti:

- U bazi postoji već kreirana rezervacija za određenu sobu i termin.
- Priložen JSESSIONID putem -DSESSIONID=

Ulazi:

- Isti tip sobe i isti datum/termin kao postojeća rezervacija
- Datum dolaska: 06-02-2026
- Datum odlaska: 07-02-2026
- Broj osoba: 2

Koraci ispitivanja (Selenium):

1. Otvoriti /reservations.
2. Unijeti termin koji je već zauzet (preduvjet).
3. Kliknuti "Prikaži dostupne sobe".
4. Pokušati odabrati sobu koja je zauzeta:
 - pri potvrди rezervacije sustav vraća grešku.
5. Verificirati prikaz poruke ("Nema dostupnih soba").

Očekivani izlaz:

- Sustav sprječava preklapanje (rezervacija nije kreirana).
- Korisniku je prikazana razumljiva poruka.

Dobiveni izlaz:

Test set: com.quantumhotel.SYS04_ReservationOverlapTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.927 s -- in
com.quantumhotel.SYS04_ReservationOverlapTest (**PASS**)

SYS-05 — Poziv nepostojeće/neimplementirane funkcionalnosti

```
@Test
void SYS05_nonExistingRoute_shouldShow404OrNotFoundMessage() {
    openApp(FrontendRoutes.notFoundProbe());

    String src = driver.getPageSource().toLowerCase();
    boolean looksLike404 =
        src.contains("404")
            || src.contains("This page could not be found.")
            || driver.getCurrentUrl().contains("404");

    assertTrue(looksLike404, "Očekivan 404/not found prikaz za nepostojeću rutu.");
}
```

Slika 6.2.5. SYS-05

Cilj: Provjeriti da sustav reagira na pokušaj korištenja nepostojeće funkcionalnosti ili nepostojeće rute (nema rušenja aplikacije, korisnik dobiva obavijest ili 404 stranicu)

Ulazi (primjer 1 — nepostojeća ruta):

- Direktna navigacija: BASE_URL/admin/non-existing-feature

Koraci ispitivanja (Selenium):

1. Otvoriti BASE_URL/admin/non-existing-feature.
2. Verificirati da se prikaže:
 - 404 stranica

Očekivani izlaz:

- Aplikacija se ne ruši.
- Korisnik dobiva jasnu informaciju da funkcionalnost/stranica ne postoji.

Dobiveni izlaz (priložiti):

Test set: com.quantumhotel.SYS05_NonExistingFeatureTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.396 s -- in
com.quantumhotel.SYS05_NonExistingFeatureTest (**PASS**)

Rezultati

Pokretanje testiranja putem:

```
.\mvnw.cmd test -Dheadless=true -DSESSIONID=...
```

```
find version of CDP to use for 144.0.7559.59. You may need to include a dependency on a specific version of the CDP using something similar to `org.seleniumhq.selenium:selenium-devtools-v86:4.20.0` where the version ("v86") matches the version of the chromium-based browser you're using and the version number of the artifact is the same as Selenium's.
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.927 s -- in com.quantumhotel.SYS04_ReservationOverlapTest
[INFO] Running com.quantumhotel.SYS05_NonExistingFeatureTest
2026-01-22T19:01:09.568+01:00 INFO 28128 --- [Quantum Hotel] [           main] i.g.bonigarcia.wdm.WebDriverManager : Using chromedriver 144.0.7559.96 (resolved driver for Chrome 144)
2026-01-22T19:01:09.570+01:00 INFO 28128 --- [Quantum Hotel] [           main] i.g.bonigarcia.wdm.WebDriverManager : Exporting webdriver.chrome.driver as C:\Users\LUKAS\.cache\selenium\chromedriver\win64\144.0.7559.96\chromedriver.exe
2026-01-22T19:01:10.186+01:00 WARN 28128 --- [Quantum Hotel] [           main] o.o.selenium.devtools.CdpVersionFinder : Unable to find CDP implementation matching 144
2026-01-22T19:01:10.186+01:00 WARN 28128 --- [Quantum Hotel] [           main] o.o.selenium.chromium.ChromiumDriver : Unable to find version of CDP to use for 144.0.7559.59. You may need to include a dependency on a specific version of the CDP using something similar to `org.seleniumhq.selenium:selenium-devtools-v86:4.20.0` where the version ("v86") matches the version of the chromium-based browser you're using and the version number of the artifact is the same as Selenium's.
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.396 s -- in com.quantumhotel.SYS05_NonExistingFeatureTest
[INFO] Running com.quantumhotel.users.UserServiceTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.139 s -- in com.quantumhotel.users.UserServiceTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 38, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  27.810 s
[INFO] Finished at: 2026-01-22T19:01:11+01:00
[INFO] -----
```

Slika 6.2.6. Rezultati testiranja sustava

7. Korištene tehnologije i alati

Tehnologije korištene za izradu aplikacije

Za frontend aplikacije koristili smo React.js (uz framework Next.js) i JavaScript. Za backend koristili smo Java (uz framework Springboot). Frontend i backend povezali smo u kontenjeru Docker (koristeći docker-compose) te smo to postavili u produkciju koristeći AWS EC2. Kako bismo korisnicima omogućili pristup isključivo frontend aplikaciji te istovremeno lokalizirali backend, implementirali smo Nginx reverse proxy. Bazu podatka dizajnirali smo pomoću alata ERDplus te kreirali relacije pomoću PostgreSQL-a. Za razvoj aplikacije tj. pisanje koda korišteno je razvojno okruženje IntelliJ IDEA, namijenjeno izradi aplikacija u programskom jeziku Java. Koristili smo i tehnologiju Google Maps za prikaz lokacije.

Tehnologije korištene za izradu dokumentacije

Dokumentaciju smo pisali na Wikiju na GitHubu. Za modeliranje dijagrama (dijagrami obrazaca uporabe, sekvencijski dijagrama, klasni dijagrami) koristili smo alat sa predavanja - Astah UML. Za modeliranje ostalih dijagrama smo koristili stranice draw.io (za dijagrame komponenti) i stranicu PlantUML-a (za dijagram razmještaja). Za prikaz relacijske sheme baze podataka koristili smo stranicu dbdiagram.io. Za prikaz dijagrama razreda koristili smo integrirani dio programa IntelliJ - PlantUML Integration koji nam je automatski generirao te dijagrame. Konačna verzija dokumentacije je izvedena u PDF format pomoću programa Microsoft Word.

Ostale tehnologije

Pri testiranju aplikacija koristili smo integrirani framework u SpringBootu i Mockito (testiranje komponenti) te alat za snimanje korisničkih akcija u pregledniku i automatsko ponavljanje testova Selenium (testiranje sustava). Za kontrolu verzija izvornog koda korišten je sustav Git, pri čemu je repozitorij projekta pohranjen na platformi GitHub. Za rješavanje problema i debuggiranje korisli smo alate umjetne inteligencije - ChatGPT, Gemini i Claude. Za komunikaciju među članovima koristili smo WhatsApp i Discord. Na aplikaciji WhatsApp smo dogovarali sastanke i koristili ju za brze dogovore i eventualne promjene u projektu, a na Discordu smo redovito držali sastanke gdje smo kreirali detaljan plan dalnjeg rada na projektu, podjelu zadataka i rješavali veće probleme kojima se trebalo dublje posvetiti. Zadatke smo raspodijelili preko aplikacije Microsoft Word.

Internet poveznice za korištene tehnologije

1. [React.js](#)
2. [Node.js](#)
3. [SpringBoot](#)
4. [Docker](#)
5. [Amazon Web Service EC2](#)
6. [NGINX](#)
7. [ERDplus](#)
8. [PostgreSQL](#)
9. [IntelliJ IDEA](#)
10. [Google Maps](#)
11. [Astah UML](#)
12. [Draw.io/diagrams.net](#)
13. [PlantUML](#)
14. [dbdiagram.io](#)
15. [Mockito](#)
16. [Selenium](#)
17. [ChatGPT](#)
18. [Gemini](#)
19. [Claude](#)
20. [WhatsApp](#)
21. [Discord](#)
22. [Microsoft Word](#)
23. [Git](#)
24. [GitHub](#)

8. Upute za puštanje u pogon

Quantum Hotel – Deployment i administracija

Ovaj odjeljak dokumentacije daje smjernice za instalaciju, konfiguraciju, pokretanje i administraciju aplikacije Quantum Hotel na razvojnem i producijskom okruženju. Fokus je na AWS (Ubuntu Linux) deployu pomoću Docker Compose + Nginx reverse proxy + HTTPS (Certbot).

1. Instalacija (AWS Ubuntu)

1.1. Preduvjeti

Na serveru je potrebno instalirati Git, Docker + Docker compose

1.2. Instalacija paketa (Ubuntu)

```
sudo apt update  
sudo apt install -y git ca-certificates curl  
# Docker  
curl -fsSL https://get.docker.com | sudo sh  
sudo usermod -aG docker $USER  
# Docker Compose (plugin)  
sudo apt install -y docker-compose-plugin
```

OJAVA / prijava nakon dodavanja u docker grupu:

```
exit
```

Provjera:

```
git --version  
docker --version  
docker compose version
```

2. Preuzimanje (kloniranje repozitorija)

```
git clone https://github.com/FranBistrovic/QuantumHotel.git  
cd QuantumHotel
```

3. Konfiguracija aplikacije

3.1. Frontend konfiguracija (.env.production)

Datoteka: frontend/.env.production

Producija (javna domena) :

NEXT_PUBLIC_API_URL=https://quantumhotel.duckdns.org (URL koji browser koristi, mora biti https i domena)

API_INTERNAL_URL=http://backend:8080 (interni URL za Docker mrežu, frontend → backend servis)

3.2. Backend konfiguracija (application.properties)

Datoteka: backend/src/main/resources/application.properties

```
# --- GENERAL ---
spring.application.name=Quantum Hotel
spring.web.resources.add-mappings=false
app.support.email=redacted@gmail.com
app.domain=${DOMAIN:http://localhost:8080}
server.forward-headers-strategy=framework
server.servlet.session.cookie.secure=true
server.servlet.session.cookie.same-site=None
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB

# --- OAuth2 ---
spring.security.oauth2.client.registration.google.client-
id=redacted.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=redacted
spring.security.oauth2.client.registration.google.redirect-
uri={baseUrl}/login/oauth2/code/{registrationId}

# --- PostgreSQL Connection ---
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=${SPRING_DATASOURCE_URL:jdbc:postgresql://localhost:5432/
quantumhotel}
spring.datasource.username=${SPRING_DATASOURCE_USERNAME:quantum_user}
spring.datasource.password=${SPRING_DATASOURCE_PASSWORD:quantum-db-
password}

# --- Hibernate / JPA ---
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

# --- Email ---
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=redacted@gmail.com
spring.mail.password=redacted
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
```

Bitno za produkciju: DOMAIN se proslijeđuje kroz Docker Compose environment (vidi dolje), ako se koristi Google OAuth2 i HTTPS, redirect URI mora odgovarati javnoj domeni

4. Docker (build + run)

4.1. Dockerfile datoteke (referenca)

Frontend frontend/DockerFile:

```
FROM node:20
WORKDIR /app
COPY package*.json .
RUN npm install
COPY ..
RUN npm run build
EXPOSE 3000
CMD ["npm", "run", "start"]
```

Backend backend/DockerFile:

```
FROM maven:3.8.4-openjdk-17 AS build

WORKDIR /app
COPY ./pom.xml /app
COPY ./src /app/src
RUN mvn clean package -Dmaven.test.skip=true

FROM eclipse-temurin:17-jdk
WORKDIR /app
COPY --from=build /app/target/*jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

4.2. docker-compose.yml

services:

```
db:
  image: postgres:16
  container_name: quantumhotel-db
  restart: unless-stopped
  environment:
    POSTGRES_DB: quantumhotel
    POSTGRES_USER: quantum_user
    POSTGRES_PASSWORD: quantum-db-password
  ports:
    - "5432:5432"
  volumes:
    - pgdata:/var/lib/postgresql/data
  networks:
    - quantum-net
```

backend:

build:

```

context: ./backend
dockerfile: Dockerfile
container_name: quantumhotel-backend
restart: unless-stopped
depends_on:
  - db
environment:
  SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/quantumhotel
  SPRING_DATASOURCE_USERNAME: quantum_user
  SPRING_DATASOURCE_PASSWORD: quantum-db-password
  SPRING_JPA_HIBERNATE_DDL_AUTO: update
  SPRING_PROFILES_ACTIVE: docker
  DOMAIN: quantumhotel.duckdns.org
ports:
  - "8080:8080"
networks:
  - quantum-net

frontend:
  container_name: quantumhotel-frontend
  build:
    context: ./frontend
    dockerfile: Dockerfile
    restart: unless-stopped
  depends_on:
    - backend
  ports:
    - "3000:3000"
  environment:
    NEXT_PUBLIC_API_URL: http://backend:8080
  networks:
    - quantum-net

volumes:
  pgdata:

networks:
  quantum-net:

```

Pokretanje:

docker compose up -d --build

Provjera kontejnera:

docker ps

docker compose logs -f --tail=200

5. Nginx reverse proxy (produkcija)

5.1. Instalacija Nginx-a

```
sudo apt install -y nginx
```

5.2. Važno: povećanje limita za upload slika

U Nginx konfiguraciju (u http { ... } bloku) dodati:

```
client_max_body_size 50M;
```

Najčešće lokacije:

- `/etc/nginx/nginx.conf` (preporučeno)
- ili unutar site konfiguracije (ali najbolje globalno u http bloku)

5.3. Proxy konfiguracija (sites-available)

Datoteka `/etc/nginx/sites-available/quantumhotel`:

```
server {
    server_name quantumhotel.duckdns.org;
```

```
location / {
    proxy_pass http://localhost:3000;
}
```

```
location /api {
    proxy_pass http://localhost:8080;
}
```

```
location /uploads/profile_pics/ {
    proxy_pass http://localhost:8080/uploads/profile_pics/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

```
location /uploads/categories/ {
    proxy_pass http://localhost:8080/uploads/categories/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

```
location /oauth2/ {
    proxy_pass http://localhost:8080/oauth2/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

```
location /login/oauth2/ {
```

```

proxy_pass http://localhost:8080/login/oauth2/;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}

location /logout {
proxy_pass http://localhost:8080/logout;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}

listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/quantumhotel.duckdns.org/fullchain.pem; # managed by Certbot
ssl_certificate_key /etc/letsencrypt/live/quantumhotel.duckdns.org/privkey.pem; # managed by Certbot
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
if ($host = quantumhotel.duckdns.org) {
return 301 https://$host$request_uri;
} # managed by Certbot

listen 80;
server_name quantumhotel.duckdns.org;
return 404; # managed by Certbot
}

```

Enable site + restart:

```
sudo ln -s /etc/nginx/sites-available/quantumhotel /etc/nginx/sites-enabled/quantumhotel
```

```
sudo nginx -t
```

```
sudo systemctl reload nginx
```

6. HTTPS (Certbot / Let's Encrypt)

6.1. Instalacija Certbot-a

```
sudo apt install -y certbot python3-certbot-nginx
```

6.2. Dobivanje SSL certifikata

Preduvjet: DNS za quantumhotel.duckdns.org mora pokazivati na javni IP AWS instance.

```
sudo certbot --nginx -d quantumhotel.duckdns.org
```

Automatska obnova certifikata:

```
sudo systemctl status certbot.timer
```

```
sudo certbot renew --dry-run
```

Nakon uspješnog certifikata, Certbot će automatski dopuniti Nginx konfiguraciju (listen 443 + redirect 80 → 443)

7. Pristup aplikaciji (javni poslužitelj)

7.1. Korisnički pristup

Aplikaciji se pristupa putem preglednika:

<https://quantumhotel.duckdns.org>

7.2. Administratorsko sučelje

Aplikacija ima admin rute/sučelje:

<https://quantumhotel.duckdns.org/admin>

Default admin pristup (inicijalno):

-> korisnik: admin

-> lozinka: admin123

Sigurnosna preporuka (obavezno nakon prvog ulaska):

-> promijeniti email na admin accountu na stvarni / važeći email (da radi reset lozinke)

-> zatim resetirati lozinku preko opcije „Forgot password“

8. Administracija i održavanje

8.1. Logovi

```
docker compose logs -f --tail=200
```

```
docker logs -f quantumhotel-backend --tail=200
```

```
docker logs -f quantumhotel-frontend --tail=200
```

8.2. Update aplikacije

```
cd QuantumHotel
```

```
git pull origin master
```

```
docker compose up -d --build
```

8.3. Backup baze (PostgreSQL)

```
docker exec -t quantumhotel-db pg_dump -U quantum_user quantumhotel >
backup_quantumhotel.sql
```

Restore (primjer):

```
cat backup_quantumhotel.sql | docker exec -i quantumhotel-db psql -U quantum_user
-d quantumhotel
```

9. Ograničenja i napomene

- Upload slika: Nginx mora imati client_max_body_size 50M; inače se slike neće uploadati (413 Request Entity Too Large).
- Portovi: u produkciji se korisnicima izlaže samo 80/443 preko Nginx-a; 3000 i 8080 su "iza" reverse proxy-a.
- OAuth2: Google OAuth redirect URI mora odgovarati točnoj HTTPS domeni.
- Tajne (mail password, OAuth secret, DB password): ne držati u repozitoriju; preporuka je prebaciti u environment varijable (Compose / AWS Secrets / SSM).

9. Zaključak i budući rad

Projekt Quantum Hotel uspješno je priveden kraju, čime je ostvaren cilj kreiranja cjelovitog sustava za digitalizaciju hotelskog poslovanja. Razvojni tim od sedam ljudi bio je strukturiran kroz jasnu podjelu na frontend i backend razvoj te ulogu voditelja projekta koji je koordinirao sve aktivnosti. Za izradu korisničkog sučelja korišten je React.js uz framework Next.js, dok je serverska strana aplikacije izgrađena u Javi koristeći Spring Boot framework. Cjelokupni sustav, uključujući frontend i backend, povezan je u kontejnere pomoću Dockera i alata docker-compose, što je omogućilo stabilnu i izoliranu okolinu. Aplikacija je postavljena u produkciju na Linux virtualnu mašinu unutar Azure cloud platforme, čime je osigurana njezina dostupnost na internetu.

Tijekom rada na projektu, tim je održavao redovite sastanke, a sve podjele poslova i ključne odluke zapisivane su u word dokumente radi lakšeg praćenja napretka. Komunikacija među članovima tima odvijala se svakodnevno putem Discorda i WhatsAppa, što je osiguralo brzu razmjenu informacija i rješavanje problema. Ovaj projekt je bio vrlo vrijedno iskustvo koje ćemo moći dalje primijeniti u budućim projektima. Iskusili smo rad u grupi gdje su komunikacija, organizacija i suradnja ključni elementi uspjeha te smo stekli bitna znanja za nastavak fakultetskog i poslovnog života.

Rad u timu od sedam ljudi donio je brojne izazove, ali i prilike za učenje o važnosti usklađivanja različitih dijelova sustava. Naučili smo kako efikasno rješavati konflikte kod integracije koda te kako održati motivaciju i produktivnost unutar grupe. Postigli smo značajni napredak u razumijevanju izrade i implementacije web aplikacija te iskustvu funkcioniranja u timu za razvoj aplikacija. Zadovoljni smo i ponosni našim stečenim znanjima i finalnom verzijom naše aplikacije koja predstavlja spoj moderne tehnologije i praktične primjenjivosti.

Iako je projekt završen u planiranom okviru, prostora za usavršavanje uvijek ima, a u budućnosti se sustav može dodatno nadograditi uvođenjem naprednijih funkcionalnosti. To uključuje integraciju sustava za online plaćanje karticama, razvoj mobilne aplikacije za hotelsko osoblje te implementaciju sustava lojalnosti za stalne goste. Također, postoji potencijal za uvođenje napredne analitike temeljene na umjetnoj inteligenciji koja bi pomogla u predviđanju popunjenoosti hotelskih kapaciteta, čime bi Quantum Hotel postao još konkurentnije rješenje na tržištu.

Kao zaključak, ovaj projekt nije samo rezultat tehničkog znanja, već i dokaz snage timskog rada. Ponosni smo na činjenicu da smo uspjeli proći cijeli put od ideje i planiranja do funkcionalne aplikacije u produkciji. Stečena iskustva u rješavanju stvarnih problema, svladavanju novih tehnologija i timskome radu smatramo najvažnijim ishodom ovog projekta te s velikim optimizmom gledamo na buduće inženjerske izazove koji su pred nama.

A. Popis literature

- Programsko inženjerstvo, FER ZEMRIS, <http://www.fer.hr/predmet/proinz>
- ChatGPT, <https://chat.openai.com>
- Gemini, <https://gemini.google.com>
- IntelliJ IDEA, <https://www.jetbrains.com/idea>
- React Documentation, <https://react.dev>
- Next.js Documentation, <https://nextjs.org/docs>
- Spring Boot Reference Documentation, <https://docs.spring.io/spring-boot/index.html>
- Docker Documentation, <https://docs.docker.com>
- Microsoft Azure Documentation, <https://learn.microsoft.com/en-us/azure>
- Google Maps Platform, <https://developers.google.com/maps>
- OAuth 2.0 Authorization Framework, <https://oauth.net/2>
- Visual Studio Code, <https://code.visualstudio.com/docs>
- Astah Community, <https://astah.net/products/astah-uml>
- GitHub, <https://github.com/VladoSruk/Programsko-inzenjerstvo>

B. Dnevnik promjena dokumentacije

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Napravljen predložak i napisan početni opis projektnog zadatka	Fran Bistrović	18.10.2025
0.2	Razrađena analiza zahtjeva	Matija Tušek, Marko Majstorović, Lukas Kraljić	18.10.2025
0.3	Raspisani obrasci uporabe, dodani pripadni dijagrami i raspisana tablica uključenosti ključnih funkcionalnosti u obrasce uporabe	Marija Špoljarić, Nina Jurić	19.10.2025
0.4	Ažurirani dijagrami obrazaca uporabe, analiza zahtjeva, specifikacija zahtjeva sustava te dodani sekvensijski dijagrami	Marija Špoljarić, Nina Jurić, Matija Tušek, Marko Majstorović, Dina Janđel	3.11.2025
0.5	Razrađena prva verzija opisa arhitekture sustava	Matija Tušek	4.11.2025
0.6	Dizajnirana i unesena prva verzija baze podataka	Marko Majstorović	5.11.2025
0.7	Nadopunjjen opis projektnog zadatka	Fran Bistrović	8.11.2025
0.8	Dodani dijagrami razreda	Lukas Kraljić	10.11.2025
0.9	Dodani dijagrami komponenata i razmještaja	Matija Tušek	21.12.2025
0.10	Razrađena prva verzija ispitivanja programskog rješenja	Matija Tušek	11.1.2026
0.11	Dodani dijagrami stanja i aktivnosti	Fran Bistrović, Dina Janđel	13.1.2026.
0.12	Unesena druga verzija baze podataka	Marko Majstorović	20.1.2026.
0.13	Unesene korištene tehnologije i alati	Marko Majstorović	20.1.2026.
0.14	Ažurirana analiza zahtjeva i specifikacija zahtjeva sustava	Marija Špoljarić	20.1.2026.
0.15	Ažuriran opis projektnog zadatka	Fran Bistrović	21.1.2026.
0.16	Ažuriran README.md	Fran Bistrović	21.1.2026.
0.17	Ažuriran popis literature	Fran Bistrović	21.1.2026.
0.18	Raspisan zaključak projekta	Fran Bistrović	21.1.2026.
0.19.	Ažurirani dijagrami razreda	Lukas Kraljić	21.1.2026.
0.20.	Dodano ispitivanje komponenti za UserService	Lukas Kraljić	21.1.2026.
0.21.	Dodane upute za puštanje u pogon	Lukas Kraljić	21.1.2026.
0.22.	Uređene korištene tehnologije i alati	Lukas Kraljić	21.1.2026.
0.23.	Ažuriran README.md - Tehnologije	Fran Bistrović	22.1.2026.
0.24.	Dodano ispitivanje sustava	Lukas Kraljić	22.1.2026.

C. Prikaz aktivnosti grupe

1. sastanak

- Datum: 15. listopada 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - upoznavanje svih članova tima
 - proučavanje zadatka
 - iznesene želje svih članova tima oko uloga u timu
 - raspodjela zadataka - uspostava GitHuba, analiza zahtjeva, specifikacije zahtjeva sustava i prezentacija
 - uspostava WhatsApp i Discord grupa svih članova

2. sastanak

- Datum: 2. studenoga 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - dogovorena vrsta arhitekture (klijent - poslužitelj)
 - proučavanje Next.js
 - dogovor o korištenju PostgreSQL
 - raspodjela dalnjih poslova (frontend, baze, login i registracija bez Google računa)
 - dogovorene revizije dokumentacije

3. sastanak

- Datum: 7. studenoga 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - pregled napravljenoga (baza podataka, arhitektura sustava, sekvencijski dijagrami)
 - dogovoren dizajn
 - raspodjela dalnjih poslova (Class dijagrami, Frontend, ažuriranje dijagrama)
 - dogovoreno testiranje prije 1. predaje

4. sastanak

- Datum: 11. studenoga 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - pregled i provjera funkcionalnosti prije 1. predaje
 - pregled dokumentacije prije 1. predaje

5. sastanak

- Datum: 18. prosinca 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - pregled i diskutiranje o rezultatima nakon 1. predaje
 - podjela zadataka za SPRINT 2

6. sastanak

- Datum: 18. prosinca 2025.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - pregled i diskutiranje o rezultatima nakon 1. predaje
 - podjela zadataka za SPRINT 2

7. sastanak

- Datum: 10. siječnja 2026.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - komentiranje napravljenoga u SPRINT-u 2 do sada
 - posljednji zadaci i privođenje SPRINT-a 2 kraju

8. sastanak

- Datum: 13. siječnja 2026.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - detaljna analiza i testiranje Alpha verzije aplikacije
 - postavljanje Alpha verzije aplikacije na server

9. sastanak

- Datum: 19. siječnja 2026.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - dodijeljeni finalni poslovi oko otklanjanja bugova
 - dodijeljeni finalni poslovi oko dokumentacije

10. sastanak

- Datum: 23. siječnja 2026.
- Prisustvovali: F. Bistrović, D. Jandžel, N. Jurić, L. Kraljić, M. Majstorović, M. Špoljarić, M. Tušek
- Teme sastanka:
 - finalni sastanak prije predaje projekta

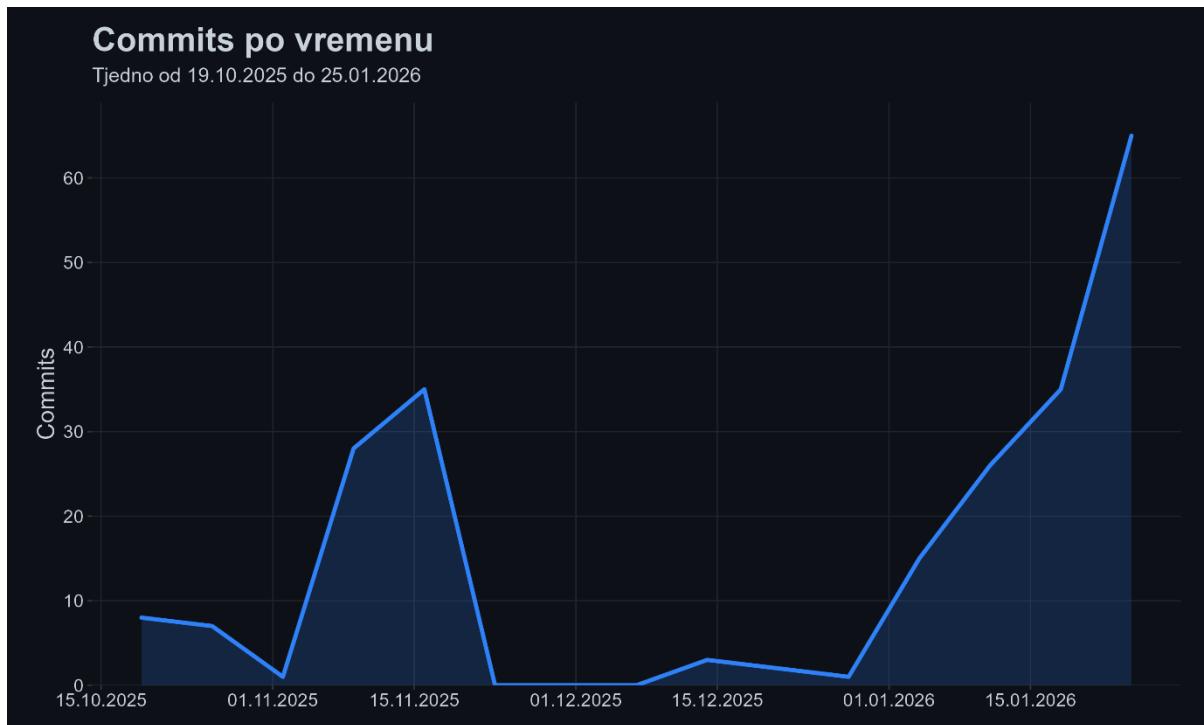
Napomena: plan rada je nakon svakog sastanka bio objavljen u online WORD dokumentu.

Tablica aktivnosti

Napomena: upravljanje projektom sadrži i vrijeme provedeno na sastancima

Aktivnost	F.B.	D.J.	N.J.	L.K.	M.M.	M.Š.	M.T.
Upravljanje projektom	12h						
Opis projektnog zadatka	2h						
Funkcionalni zahtjevi	2h			2h	6h	1h	5h
Opis pojedinih obrazaca			1h			1h	
Dijagrami obrazaca			9h			8h	
Sekvencijski dijagrami			4h			5h	
Opis ostalih zahtjeva					2h		3h
Arhitektura i dizajn sustava	1h	2h					5h
Baza podataka					7h		
Dijagram razreda				3h			
Dijagram stanja		2h					
Dijagram aktivnosti			2h				
Dijagram komponenti						3h	
Korištene tehnologije i alati				2h			
Ispitivanje programskog rješenja				5h	3h		5h
Dijagram razmještaja						2h	
Upute za puštanje u pogon				1h			
Dnevnik sastajanja	1h						
Zaključak i budući rad	2h						
Popis literature	1h						
Backend	12h			30h	20h		20h
Frontend		20h	28h	20h		23h	
Puštanje u pogon				3h			
Učenje programskih alata	6h	5h	5h	1h	4h	6h	5h
Izrada prezentacija I PDF-ova		10h					8h

Dijagram pregleda promjena



C.1. Dijagram pregleda promjena

Ključni izazovi i rješenja

Opis izazova

Na samom početku projekta najveći nam je izazov bio savladavanje novih alata i tehnologija koje smo odabrali za izradu aplikacije. Budući da se većina nas ranije nije susrela s tim sustavima, trebalo nam je vremena da pohvatamo osnove i uskladimo način rada. Osim tehničkog dijela, dosta nas je mučila organizacija vremena. Bilo je jako teško pronaći termine za sastanke koji bi svima odgovarali zbog brojnih fakultetskih obaveza svih sedam članova tima. Zbog toga smo se često morali nalaziti u prilično nezgodno vrijeme, poput vrlo ranih jutarnjih sati ili kasno navečer. To početno nesnalaženje dovelo je do malih kašnjenja, ali kako smo postajali sigurniji u ono što radimo, tako je i posao u drugoj fazi išao puno brže i jednostavnije.

Rješenja

Probleme s kojima smo se sretali rješavali smo zajedničkim trudom i stalnim učenjem u hodu. Kako bismo nadoknadili manjak zajedničkih termina uživo, maksimalno smo iskoristili aplikacije poput Discorda i WhatsAppa. Tamo smo svakodnevno komunicirali, rješavali probleme u kodu i dogovarali sljedeće korake, što nam je omogućilo da konstantno napredujemo. Ključ uspjeha bila je bolja podjela zadataka i međusobna podrška unutar tima. Na kraju smo uspjeli sve dijelove aplikacije povezati u jednu cjelinu i uspješno je pokrenuti na serveru. Ovo iskustvo rada u velikoj grupi naučilo nas je koliko je važna dobra komunikacija i prilagodba unutar tima kako bi se jedan ovakav projekt uspješno završio.