

**UNIVERZITET SINGIDUNUM  
TEHNIČKI FAKULTET U NOVOM SADU**

**RAZVOJ INFORMACIONOG SISTEMA ZA  
PODRŠKU UPRAVLJANJU SPORTSKOM  
DVORANOM**

- diplomski rad -

**Mentor:**

**prof. dr *Đorđe Obradović***

**Kandidat:**

***Pavle Perić (2017270304)***

**Novi Sad, 2025**

# Sadržaj

<b>Uvod .....</b>	<b>- 1 -</b>
<b>1 Karakteristične tehnologije i alati.....</b>	<b>- 2 -</b>
1.1 Backend .....	- 2 -
1.2 Frontend.....	- 3 -
1.3 Baza podataka.....	- 3 -
1.4 Stripe.....	- 3 -
1.5 Alati .....	- 4 -
<b>2 Specifikacija modela.....</b>	<b>- 5 -</b>
2.1 Dijagram slučajeva upotrebe .....	- 5 -
2.2 Dijagram klasa .....	- 10 -
<b>3 Implementacija .....</b>	<b>- 14 -</b>
3.1 Serverska strana aplikacije .....	- 14 -
3.1.1 Stripe integracija za online plaćanje rezervacija.....	- 23 -
3.2 Klijentska strana aplikacije.....	- 25 -
3.2.1 Tok rada korisnika kroz aplikaciju .....	- 25 -
3.2.2 Administratorska strana aplikacije .....	- 31 -
<b>4 Zaključna razmatranja .....</b>	<b>- 33 -</b>
<b>Literatura .....</b>	<b>- 34 -</b>

## Spisak slika

Slika 1 Grafički prikaz arihtekture aplikacije.....	- 2 -
Slika 2 Swagger UI prikaz inicijalnog interfejsa.....	- 21 -
Slika 3 Swagger UI testiranje POST metode api/reservation.....	- 22 -
Slika 4 Stranica za prijavljivanje u sistem.....	- 25 -
Slika 5 Stranica za registraciju novog korisnika .....	- 26 -
Slika 6 Početna stranica aplikacije sa navigacionim menijem .....	- 26 -
Slika 7 Lista sala sa filterom i pretragom .....	- 27 -
Slika 8 Detaljan prikaz sale sa informacijama i formularom za rezervaciju .....	- 27 -
Slika 9 Popunjen formular za kreiranje rezervacije.....	- 28 -
Slika 10 Poruka o uspešno kreiranoj rezervaciji .....	- 28 -
Slika 11 Stranica „Moje rezervacije“ sa listom svih rezervacija.....	- 29 -
Slika 12 Popup prozor za potvrdu otkazivanja rezervacije .....	- 29 -
Slika 13 Stripe Checkout stranica za unos podataka o plaćanju.....	- 30 -
Slika 14 Stranica sa potvrdom da je plaćanje uspešno izvršeno.....	- 30 -
Slika 15 Administratorski panel sa pregledom tabela .....	- 31 -
Slika 16 Modalni prozor za izmenu zapisa.....	- 32 -
Slika 17 Potvrdni popup prozor za brisanje zapisa.....	- 32 -

## Uvod

U savremenom dobu digitalizacije, sve veći broj poslovnih procesa prenosi se na internet, sa ciljem da se unapredi efikasnost i omogući kvalitetnije korisničko iskustvo. Jedno od područja u kome digitalizacija još uvek nije dovoljno zastupljena jeste upravljanje sportskim objektima – dvoranama i salama koje se koriste za rekreativne i profesionalne sportske aktivnosti.

Tradicionalan način rezervacije termina u sportskim centrima obično se zasniva na telefonskim pozivima, ličnom dolasku ili ručnom vođenju evidencije. Takav pristup je spor, neprecizan i podložan greškama, posebno kada je u pitanju veliki broj sala i veliki obim korisnika. Primer ovakve situacije jeste i **Spens centar u Novom Sadu**, koji sadrži veliki broj sala, ali još uvek ne poseduje objedinjeni online sistem za rezervacije i upravljanje terminima. Upravo ovakvi problemi bili su inspiracija za razvoj sopstvenog informacionog sistema za rezervaciju sportskih dvorana.

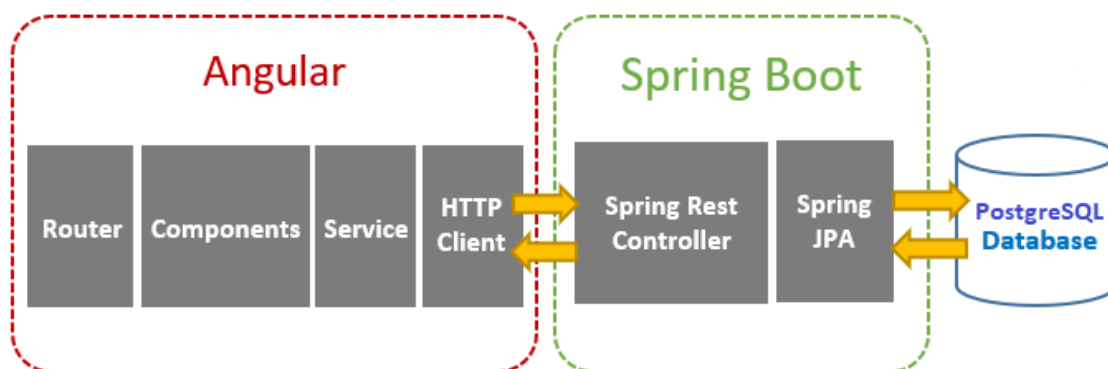
Na tržištu postoje i određena rešenja koja omogućavaju online rezervaciju sportskih terena. Na primer, platforma **BookaSport** (BookaSport, 2024) omogućava rezervaciju terena širom Srbije i nudi pregled dostupnih termina, dok aplikacija **Playo** (Playo, 2024) pruža međunarodno orijentisane funkcionalnosti i povezuje sportiste i objekte različitih profila. Međutim, ovakva rešenja često nisu dovoljno prilagođena specifičnim potrebama lokalnih sportskih centara, niti omogućavaju integraciju svih sala i sportova u jedinstvenom sistemu.

Prednosti sopstvenog sistema ogledaće se u tome što je u potpunosti prilagođen stvarnim potrebama korisnika i administratora objekata. Mane postojećih rešenja – nedovoljna fleksibilnost, opšta priroda ili visoka cena – predstavljaju dodatnu motivaciju za razvoj specifičnog rešenja. Na taj način, ovaj rad ima za cilj da ponudi model sistema koji objedinjeno rešava problem rezervacija sportskih sala i pruža mogućnost jednostavne primene u praksi.

# 1 Karakteristične tehnologije i alati

Razvoj informacionog sistema za upravljanje sportskom dvoranom zahteva primenu pouzdanih i savremenih tehnologija kako bi se obezbedila skalabilnost, sigurnost i efikasnost rada. Odeljak koji sledi pruža kratak pregled tehnologija korišćenih na svim nivoima sistema – backend, frontend, baza podataka i sistemi za integraciju plaćanja.

Na slici 1. predstavljena je arhitektura aplikacije sa sledećim tehnologijama. Spring Boot eksportuje REST API-je koristeći Spring Web MVC i komunicira sa PostgreSQL bazom podataka koristeći Spring Data JPA. Angular klijent šalje HTTP zahteve i preuzima HTTP odgovore koristeći HttpClient modul, prikazuje podatke o komponentama. Takođe koristimo Angular ruter za navigaciju do stranica. O arhitekturi projekta detaljnije u potpoglavljima.



Slika 1 Grafički prikaz arihtekture aplikacije

## 1.1 Backend

**Spring Boot** predstavlja proširenje okvira Spring Framework, kreiran u cilju pojednostavljenja razvoja enterprise aplikacija u programskom jeziku Java. Razvijen od strane VMware tima, Spring Boot omogućava brzo pokretanje i konfiguraciju aplikacija uz minimalnu količinu ručne konfiguracije. Ključna karakteristika ovog okvira jeste tzv. “auto-configuration” mehanizam, koji automatski detektuje zavisnosti i postavlja inicijalne vrednosti konfiguracije, čime se značajno skraćuje vreme razvoja i smanjuje prostor za greške (Spring IO, n.d.).

Spring Boot omogućava jednostavnu integraciju sa bazama podataka, autentifikaciju i autorizaciju putem Spring **Security** modula, kao i **ORM** pristup kroz Spring **Data JPA**, što programeru omogućava rad sa bazom koristeći objektnu reprezentaciju. Aplikacije izrađene pomoću Spring Boot-a poseduju ugrađeni web server, čime se eliminiše potreba za spoljnim kontejnerima, što je naročito korisno u kontekstu mikrouslužne arhitekture i kontejnerizacije (npr. Docker).

## 1.2 Frontend

**Angular** je open-source razvojni okvir za izradu jednostranih veb-aplikacija (eng. *Single Page Applications* – SPA) razvijen od strane kompanije Google. Baziran na programskom jeziku **TypeScript** (nadskup JavaScript-a), Angular omogućava deklarativni pristup razvoju korisničkog interfejsa koristeći komponente, šablone i dvosmerno vezivanje podataka (eng. *two-way data binding*) (Angular, n.d.).

Angular koristi tzv. *reactive programming paradigm* uz pomoć RxJS biblioteke, što omogućava efikasnu obradu asinhronih događaja i upravljanje podacima u realnom vremenu. Okvir podržava modularnu arhitekturu, testabilnost i hijerarhiju komponenti, što ga čini pogodnim za izgradnju skalabilnih web rešenja. U okviru ovog projekta korišćen je i dodatak Angular **Material**, koji implementira dizajnerske smernice.

## 1.3 Baza podataka

**PostgreSQL** je moćan, open-source sistem za upravljanje relacionim bazama podataka (RDBMS) koji se aktivno razvija više od tri decenije. Poznat je po svojoj stabilnosti, bezbednosti i podršci za napredne funkcionalnosti kao što su transakcije, indeksiranje, replikacija, kao i rad sa nestrukturiranim podacima putem JSONB formata. Za razliku od jednostavnijih rešenja kao što su MySQL ili SQLite, PostgreSQL pruža napredne mogućnosti prilagođene zahtevnijim sistemima, kao što su složeni upiti, procedure, trigere i pogleda (PostgreSQL Global Development Group, n.d.).

U kontekstu ove aplikacije, PostgreSQL je korišćen za čuvanje korisničkih podataka, rezervacija termina i istorije plaćanja. Integracija sa Spring Boot okvirom realizovana je putem JPA (eng. *Java Persistence API*) sloja, omogućavajući rad sa objektno-relacionim mapiranjem (ORM).

## 1.4 Stripe

**Stripe** je napredna platforma za procesuiranje online plaćanja, namenjena programerima i integratorima sistema koji žele da implementiraju sigurne i brze transakcije putem interneta. Stripe omogućava povezivanje sa različitim vrstama platnih kartica, kao i kreiranje sesija plaćanja, tokenizaciju kartičnih podataka i rad sa webhook mehanizmima za praćenje statusa transakcije (Stripe, n.d.).

Stripe funkcioniše po principu RESTful API integracije, gde backend aplikacija kreira zahtev za uplatu, dok korisnik završava plaćanje na frontend strani, nakon čega se rezultat automatski vraća aplikaciji. Jedna od ključnih karakteristika Stripe-a je **PCI-DSS** kompatibilnost, što znači da aplikacija nikada ne rukuje direktno podacima o karticama, čime se znatno povećava bezbednost sistema.

Stripe podržava i napredne funkcionalnosti kao što su plaćanje u više valuta, automatsko fakturisanje, integracija sa mobilnim aplikacijama, kao i mehanizmi za povrat novca i otkazivanje transakcija.

## 1.5 Alati

Pod ostalim alatima koje uključuju ceo proces izrade aplikaje jesu:

- **Visual Studio Code** – Moćan i široko korišćen integrisani razvojni alat (IDE) koji se koristi za razvoj frontend aplikacija. Zbog svoje fleksibilnosti, podrške za mnoge programske jezike i velikog broja dostupnih dodataka, predstavlja jedan od najpopularnijih editora za rad sa Angular, React i drugim frontend tehnologijama.
- **Git** – Sistem za verzionu kontrolu koda koji omogućava praćenje promena, upravljanje verzijama i timski rad na softverskim projektima. U ovom radu koristi se za upravljanje backend kodom, olakšavajući kolaboraciju i praćenje izmjena tokom razvoja.
- **Draw.io** – Alat za kreiranje dijagrama i vizuelnih prikaza sistema. Korišćen je za planiranje arhitekture aplikacije, modelovanje procesa i grafičko predstavljanje ključnih komponenti softverskog rešenja. To ujedno jeste i prvi korak u planiranju i kreiranju strukture projekta, pa i ujedno i njegove baze podataka. O nacrtanim dijagramima sa svim objašnjenjima šta oni predstavljaju više u sledećem podglavlju.

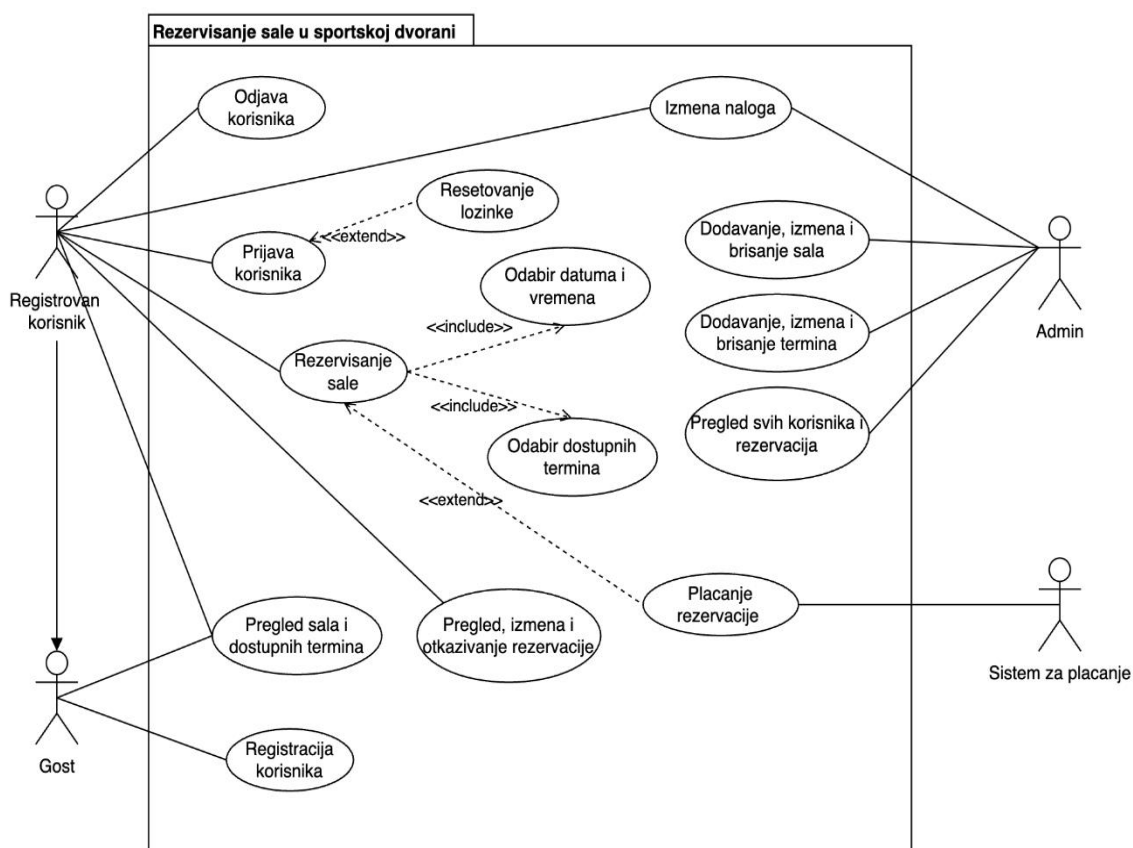
## 2 Specifikacija modela

U ovom poglavlju biće predstavljeni korisnički zahtevi koji bi trebalo da budu implementirani u sistemu i oni će biti predstavljeni pomoću dijagrama slučajeva upotrebe i dijagram klasa, koji prikazuje klase koje obuhvataju ceo sistem.

### 2.1 Dijagram slučajeva upotrebe

Tokom analize zahteva, dijagrami slučaja upotrebe pomažu da se identifikuju akteri i da se definiše ponašanje sistema. Modeliranje slučaja upotrebe je prihvaćeno i široko se koristi u industriji. Pored aktera i slučajeva upotrebe UML definiše mali skup odnosa za strukturu aktera i slučajevne upotrebe. Slučajevi korišćenja mogu biti povezani sa drugim slučajevima korišćenja sledećim odnosima:

1. **<<extend>>** – odnos **proširenja** podrazumeva da slučaj upotrebe može da proširi ponašanje opisano u drugom slučaju upotrebe, kojim upravlja uslov
2. **<<include>>** – odnos **uključivanja** znači da slučaj upotrebe uključuje ponašanje opisano u drugom slučaju upotrebe
3. **Generalization** – odnos generalizacije znači da je dete specifičniji oblik roditeljskog slučaja upotrebe. Dete nasleđuje sve karakteristike i asocijacije roditelja i može da dodaje nove karakteristike i udruženja



Dijagram 1: Dijagram slučajeva upotrebe



## ***1. Slučaj upotrebe: Logovanje***

**Kratak opis:** Prijava korisnika na sistem.

**Učesnici:** Registrovani korisnik.

**Preduslovi:** Korisnik mora imati kreiran nalog i uneti tačno korisničko ime i lozinku.

**Osnovni tok događaja:**

1. Korisnik otvara stranicu za prijavu.
2. Korisnik unosi korisničko ime i lozinku.
3. Sistem proverava kredencijale.
4. Ukoliko su podaci tačni, korisnik se uspešno prijavljuje na sistem.
5. Ako korisnik nema nalog, može se preusmeriti na stranicu za registraciju.
6. Ako je zaboravljena lozinka, korisnik može zatražiti resetovanje putem mejla.

**Alternativni tokovi:**

- **Resetovanje lozinke:**
  - Korisnik klikne na link za resetovanje lozinke.
  - Unosi svoju mejl adresu.
  - Ako mejl postoji u sistemu, dobija uputstva za reset lozinke.

**Postuslovi:** Korisnik je uspešno prijavljen na sistem. Uspešno se odjavljuje sa svog naloga i prekida sesiju u sistemu.

## ***2. Slučaj upotrebe: Odjavljivanje***

**Kratak opis:** Korisnik se odjavljuje sa svog naloga i prekida sesiju u sistemu.

**Učesnici:** Registrovani korisnik.

**Preduslov:** Korisnik je uspešno prijavljen na sistem.

**Osnovni tok:**

1. Klik na dugme za odjavu.
2. Sistem prekida sesiju.
3. Korisnik se vraća na početnu stranicu.

**Alternativni tokovi:** Nema.

**Postuslov:** Korisnik je odjavljen.

### ***3. Slučaj upotrebe: Registracija korisnika***

**Kratak opis:** Gost kreira korisnički nalog.

**Učesnici:** Gost.

**Preduslovi:** Gost nije prethodno registrovan.

**Osnovni tok:**

1. Gost otvara stranicu za registraciju.
2. Popunjava formular sa traženim podacima.
3. Sistem validira podatke.
4. Po uspešnoj validaciji, kreira se novi korisnički nalog.

**Alternativni tokovi:** Nema.

**Postuslov:** Korisnik je uspešno registrovan.

### ***4. Slučaj upotrebe: Pregled sala i dostupnih termina***

**Kratak opis:** Gost ili registrovani korisnik vidi sve sale i dostupne termine.

**Učesnici:** Gost i registrovani korisnik.

**Preduslovi:** Nema.

**Osnovni tok:**

1. Korisnik se usmerava na stranicu za pregled svih sala i dostupnih termina
2. Sistem prikazuje listu sala sa slobodnim terminima.

**Alternativni tokovi:** Nema.

**Postuslovi:** Korisnik ima uvid u mogućnosti rezervacije.

### ***5. Slučaj upotrebe: Rezervisanje sale***

**Kratak opis:** Registrovani korisnik rezerviše salu u odabranom terminu.

**Učesnici:** Registrovani korisnik i Stripe sistem za plaćanje.

**Preduslovi:** Korisnik mora biti prijavljen.

**Osnovni tok:**

1. Korisnik bira salu.
2. Bira datum i vreme za rezervaciju.
3. Pregleda dostupne termine.

4. Potvrđuje i kreira rezervaciju.

**Alternativni tokovi:**

- Korisnik može samo da rezerviše i plati uživo svoj termina, a to može i da uradi odmah, online putem.
- Ako je potvrdio rezervaciju može odmah i da plati putem stripe sistema.

**Postuslovi:** Rezervacija je uspešno zabeležena.

**6. Slučaj upotrebe: Pregled, izmena i otkazivanje rezervacije**

**Kratak opis:** Korisnik menja ili otkazuje postojeću rezervaciju.

**Učesnici:** Registrovani korisnik.

**Preduslovi:** Korisnik mora imati prethodno kreiranu rezervaciju.

**Osnovni tok:**

1. Korisnik pregleda svoje rezervacije.
2. Po izboru, može da ih izmeni ili otkaže.
3. Ako želi da izmeni, može samo da izbare u tom momentu dostupne termine.

**Postuslovi:** Rezervacija je izmenjena ili otkazana.

**7. Slučaj upotrebe: Plaćanje rezervacije**

**Kratak opis:** Korisnik vrši plaćanje rezervacije.

**Učesnici:** Registrovani korisnik i Stripe sistem za plaćanje

**Preduslovi:** Postoji validno kreirana rezervacija i validan korisnik u sistemu.

**Osnovni tok:**

1. Korisnik pristupa opciji za plaćanje.
2. Unosi potrebne podatke.
3. Sistem za plaćanje obrađuje transakciju.
4. Dobija potvrdu o uspešnoj uplati.

**Alternativni tokovi:** Nema.

**Postuslovi:** Rezervacija je uspešno nasplaćena.

## ***8. Slučaj upotrebe: Izmena naloga***

**Kratak opis:** Korisnik menja svoje podatke (npr. lozinku, ime, e-mail).

**Učesnici:** Registrovani korisnik i admin

**Preduslovi:** Korisnik ima nalog i prijavljen je u sistem.

**Osnovni tok:**

1. Korisnik ili admin pristupa stranici za izmenu naloga.
2. Menja željene podatke.
3. Kada završi klikom na dugme sačuvaće svoje izmenjene podatke.
4. Sistem potvrđuje i pamti izmenu.

**Alternativni tokovi:** Nema.

**Postuslovi:** Podaci o nalogu korisnika ili admina su ažurirani.

## ***9. Slučaj upotrebe: Upravljanje sportskim salama i terminima za rezervaciju***

**Kratak opis:** Administrator ima mogućnost upravljanja salama kao i terminima za korišćenje sportskih sala, što podrazumeva dodavanje novih termina ili sala, izmenu postojećih kao i njihovo brisanje ukoliko više nisu dostupni ili važeći.

**Učesnici:** Admin.

**Preduslovi:** Admin mora biti prijavljen na sistem sa odgovarajućim ovlašćenjima.

**Osnovni tok događaja:**

1. Administrator pristupa interfejsu za upravljanje dvoranom.
2. Ulazi na karticu sa listom sala ili termina.
3. Ima mogućnost da:
  - a. Dodaje nove termine dostupne za rezervaciju ili nove sale,
  - b. Ažurira vreme postojećih termina ili bilo koje druge podatke vezane za termine ili sale,
  - c. Obriše termine koji više nisu dostupni ili su zastareli ili sale.

**Alternativni tokovi:** Nema.

**Postuslovi:** Termin i za odabranu salu su uspešno ažurirani i dostupni korisnicima u realnom vremenu. Isti princip i za salu, izvršen uspešno i baza je ažurirana.

## ***10. Slučaj upotrebe: Pregled korisnika i njihovih rezervacija***

**Kratak opis:** Administrator ima uvid u sve registrovane korisnike sistema, kao i pregled svih njihovih izvršenih i zakazanih rezervacija sala.

**Učesnici:** Admin.

**Preduslovi:** Admin je uspešno prijavljen na sistem i ima odgovarajuće administratorske privilegije.

**Osnovni tok događaja:**

1. Administrator pristupa administrativnoj kontrolnoj tabli.
2. Pokreće prikaz liste korisnika sistema.
3. Za svakog korisnika može da pregleda:
  - a. Lične podatke (npr. ime, e-mail),
  - b. Istoriju i status rezervacija (potvrđene, otkazane, zakazane)

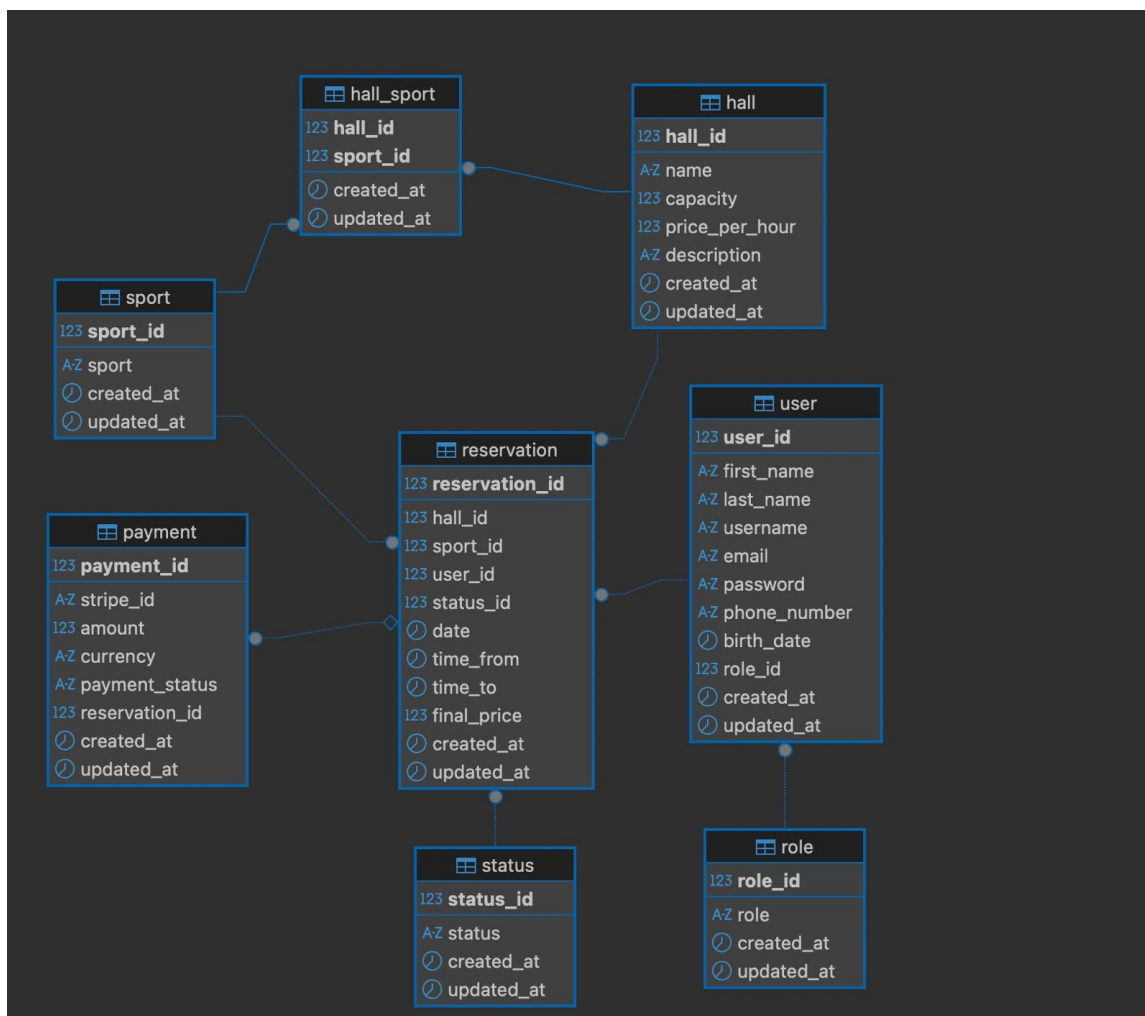
**Alternativni tokovi:** Nema.

**Postuslovi:** Administrator je dobio kompletan i ažuriran uvid u korisničke naloge i pripadajuće rezervacije, što omogućava efikasno upravljanje i nadzor nad radom sistema.

## **2.2 Dijagram klasa**

Dijagram klasa je glavni gradivni blok objektno-orijentisanog modelovanja. Koristi se za opšte konceptualno modelovanje strukture aplikacije, kao i za detaljno modelovanje prevođenja modela u programski kod. Dijagrami klasa mogu se koristiti i za modelovanje podataka. Klase u dijagramu klasa predstavljaju glavne elemente, interakcije u aplikaciji i klase koje treba da budu isprogramiranje.

U slučaju informacionog sistema za upravljanje sportskom dvoranom, dijagram klasa se sastoji iz sledećih klasa: User, Role, Hall, Slot, Reservation, Status, Payment. U nastavku sledi analiza svake od klasa pojedinačno.



Dijagram 2: Dijagram klasa

## Opis tabela:

**1. User** – Ova klasa predstavlja korisnike sistema, koji mogu imati različite uloge u zavisnosti od povezanosti sa klasom *Role*. Svaki korisnik može kreirati više rezervacija, a može biti i administrator u zavisnosti od dodeljene uloge.

- *user\_id*: Jedinstveni identifikator korisnika (primarni ključ).
- *first\_name*: Ime korisnika.
- *last\_name*: Prezime korisnika.
- *username*: Korisničko ime, obeležje korisničkog naloga.
- *email*: Imejl adresa korisnika, koristi se za prijavu.
- *password*: Lozinka korisnika, koja se čuva u heširanom obliku (BCrypt).
- *phone\_number*: Broj telefona korisnika.
- *birth\_date*: Datum rođenja korisnika.
- *role\_id*: Strani ključ koji povezuje korisnika sa njegovom ulogom u sistemu.

**2. Role** – Klasa *Role* definiše uloge koje korisnici mogu imati u sistemu. Ovo omogućava diferencijaciju prava pristupa i funkcionalnosti između administratora i registrovanog člana.

- *role\_id*: Jedinstveni identifikator uloge (primarni ključ).
- *role*: Naziv uloge, npr. “ADMIN”, “USER”.

**3. Hall** – Klasa *Hall* predstavlja jednu salu u okviru sportske dvorane koja se može rezervirati. Sala može biti povezana sa više sportova.

- *hall\_id*: Jedinstveni identifikator sale (primarni ključ).
- *name*: Naziv sale.
- *capacity*: Maksimalan broj korisnika koje sala može da primi.
- *price\_per\_hour*: Cena korišćenja sale po satu.
- *description*: Tekstualni opis sale (npr. veličina, oprema, tip sportova).

**4. Sport** – Klasa *Sport* predstavlja tip sporta koji može da se igra u određenim salama.

- *sport\_id*: Jedinstveni identifikator sporta (primarni ključ).
- *sport*: Naziv sporta (npr. “Fudbal”, “Košarka”, “Odbojka”).

**5. Status** – Klasa *Status* određuje stanje rezervacije, što omogućava filtriranje i upravljanje raspoloživošću.

- *status\_id*: Jedinstveni identifikator statusa (primarni ključ).
- *status*: Naziv statusa (“Kreirana”, “Placena”, “Otkazana”, “Završena”).

**6. Reservation** – Klasa *Reservation* beleži informacije o rezervacijama. Spaja više tabela i daje konačnu sliku jedne rezervacije. Za razliku od ranijeg plana, termin (datum i vreme) je sada deo same rezervacije, pa posebna tabela *Slot* ne postoji.

- *reservation\_id*: Jedinstveni identifikator rezervacije (primarni ključ).
- *hall\_id*: Strani ključ koji povezuje rezervaciju sa salom.
- *sport\_id*: Strani ključ koji povezuje rezervaciju sa sportom.
- *user\_id*: Strani ključ koji pokazuje koji korisnik je izvršio rezervaciju.
- *status\_id*: Strani ključ koji određuje status rezervacije.
- *date*: Datum rezervacije.
- *time\_from*: Vreme početka rezervacije.
- *time\_to*: Vreme završetka rezervacije.
- *final\_price*: Konačna cena rezervacije.

**7. Payment** – Klasa *Payment* evidentira podatke o transakcijama korisnika. Omogućava uvid u naplatu rezervacije.

- *payment\_id*: Jedinstveni identifikator plaćanja (primarni ključ).
- *stripe\_id*: Identifikator plaćanja u Stripe sistemu.
- *amount*: Iznos plaćen za rezervaciju.
- *currency*: Valuta plaćanja.
- *payment\_status*: Status plaćanja (npr. “Paid”, “Failed”).
- *reservation\_id*: Strani ključ – za koju rezervaciju je izvršeno plaćanje.

### **Medusobne veze između tabela:**

- **User / Role** – Svaki korisnik ima tačno jednu ulogu, dok jedna uloga može biti dodeljena većem broju korisnika (relacija 1:N).
- **Hall / Sport** – Jedna sala može podržavati više sportova, a jedan sport može biti vezan za više sala. Ova veza je realizovana kao relacija N:M preko pomoćne tabele *hall\_sport*.
- **Reservation / User** – Svaka rezervacija pripada tačno jednom korisniku, dok jedan korisnik može da kreira više rezervacija (relacija 1:N).
- **Reservation / Hall** – Svaka rezervacija je vezana za tačno jednu salu, dok jedna sala može imati više rezervacija (relacija 1:N).
- **Reservation / Sport** – Svaka rezervacija je vezana za tačno jedan sport, dok jedan sport može biti deo više različitih rezervacija (relacija 1:N).
- **Reservation / Status** – Svaka rezervacija ima tačno jedan status, dok jedan status može biti dodeljen većem broju rezervacija (relacija 1:N).
- **Reservation / Payment** – Svaka rezervacija može imati najviše jedno plaćanje (1:0..1), dok svako plaćanje mora biti vezano za tačno jednu rezervaciju.



### 3 Implementacija

U ovom poglavlju detaljno je opisan predlog tehničkog rešenja informacionog sistema za rezervacije u sportskoj dvorani. Nakon što su prethodno predstavljene osnovne strukture baze podataka i dijagrami procesa, ovde se razmatraju ključne komponente aplikacije koje omogućavaju realizaciju sistema. Fokus je na serverskom delu razvijenom u okviru **Spring Boot** okruženja, klijentskom delu izrađenom u **Angular** framework-u, kao i načinu na koji ove komponente komuniciraju sa **PostgreSQL** bazom podataka. Poseban akcenat stavljen je na obezbeđivanje konzistentnosti podataka, sigurnost sistema i jednostavnost korišćenja. Na taj način se osigurava da aplikacija pouzdano podržava proces rezervacije termina u sali, pregled dostupnosti i administraciju podataka.

#### 3.1 Serverska strana aplikacije

Serverska strana aplikacije razvijena je korišćenjem **Spring Boot** okvira u programskom jeziku Java. Arhitektura je zasnovana na REST principima, gde svaki resurs sistema (korisnik, sala, termin, rezervacija, plaćanje) ima definisane **endpoint-e** preko kojih klijentska aplikacija komunicira sa serverom putem HTTP zahteva. Aplikacija je organizovana po slojevima koji jasno razdvajaju poslovnu logiku, komunikaciju sa bazom podataka i kontrolu nad API zahtevima.

Struktura projekta prikazana je kroz pakete, gde svaki ima svoju ulogu:

- **config** – sadrži konfiguracione klase neophodne za rad aplikacije, poput sigurnosti (*SecurityConfig*, *JwtAuthFilter*, *JwtProperties*), podešavanja za CORS, integraciju sa Stripe servisom i generisanje OpenAPI dokumentacije.
- **controller** – implementira REST kontrolere (npr. *UserController*, *ReservationController*, *PaymentController*), koji primaju HTTP zahteve, prosleđuju ih servisnom sloju i vraćaju odgovore u obliku DTO objekata.
- **dto** – definiše Data Transfer Object klase koje služe za razmenu podataka između slojeva i komunikaciju sa frontend aplikacijom.
- **exception** – centralizovano rukovanje greškama kroz *GlobalExceptionHandler*, čime se obezbeđuje konzistentan format odgovora pri različitim izuzecima.
- **model** – obuhvata entitetske klase (*User*, *Hall*, *Slot*, *Reservation*, *Payment*...) koje su mapirane na tabele u PostgreSQL bazi.
- **repository** – sloj za pristup bazi podataka, realizovan kroz Spring Data JPA interfejs (*UserRepository*, *ReservationRepository*).
- **service** – implementira poslovnu logiku aplikacije (*ReservationService*, *HallService*, *UserService*...) i predstavlja vezu između kontrolera i repozitorijuma.
- **SportskaDvoranaApplication.java** – glavna klasa za pokretanje aplikacije

Jedan od ključnih primera rada backend-a jeste proces kreiranja nove rezervacije. Klijentska aplikacija šalje **HTTP POST zahtev** ka odgovarajućem endpoint-u u okviru klase *ReservationController*.

```

@PostMapping
public ResponseEntity<?> createReservation(@Valid @RequestBody ReservationDTO reservationDTO) {
    try {
        Reservation created = reservationService.createReservation(reservationDTO);
        ReservationDTO dto = reservationService.toDTO(created);

        Map<String, Object> response = new HashMap<>();
        response.put(key:"message", value:"Reservation created successfully");
        response.put(key:"reservation", dto);

        return new ResponseEntity<>(response, HttpStatus.CREATED);
    } catch (IllegalArgumentException ex) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
            .body(new ApiResponse(ex.getMessage()));
    }
}

```

Listing 1 POST metoda za kreiranje rezervacije

Ovaj kontroler preuzima podatke u obliku **DTO objekta** (ReservationDTO), čime se obezbeđuje da aplikacija prima samo relevantne i validne informacije koje dolaze iz spoljnog sveta.

```

public class ReservationDTO {

    @Schema(accessMode = Schema.AccessMode.READ_ONLY)
    private Long reservationId;

    @Schema(example = "1")
    private Long hallId;

    @Schema(example = "1")
    private Long sportId;

    @Schema(example = "1")
    private Long userId;

    @Schema(example = "1")
    private Long statusId;

    @Schema(example = "2025-08-04")
    private LocalDate date;

    @Schema(example = "10:00")
    private LocalTime timeFrom;

    @Schema(example = "12:00")
    private LocalTime timeTo;

    @Schema(example = "2000")
    private BigDecimal finalPrice;
}

```

Listing 2 DTO klasa za prenos podataka o rezervaciji

Nakon preuzimanja zahteva, kontroler prosleđuje podatke servisnom sloju, odnosno klasi *ReservationService*. U metodi *createReservation* vrši se validacija poslovnih pravila – najpre se proverava da li postoji sala sa zadatim identifikatorom, da li je izabrani sport podržan u toj sali, kao i da li su korisnik i status validni. Ukoliko neki od ovih uslova nije ispunjen, metoda generiše odgovarajući izuzetak, čime se korisniku vraća jasna poruka o grešci.

```

public Reservation createReservation(ReservationDTO dto) {
    Hall hall = hallRepository.findById(dto.getHallId())
        .orElseThrow(() -> new IllegalArgumentException("Hall not found with ID: " + dto.getHallId()));

    Sport sport = sportRepository.findById(dto.getSportId())
        .orElseThrow(() -> new IllegalArgumentException("Sport not found with ID: " + dto.getSportId()));

    if (!hall.getSports().contains(sport)) {
        throw new IllegalArgumentException(s:"The hall does not support the selected sport");
    }

    User user = userRepository.findById(dto.getUserId())
        .orElseThrow(() -> new IllegalArgumentException("User not found with ID: " + dto.getUserId()));

    Status status = statusRepository.findById(dto.getStatusId())
        .orElseThrow(() -> new IllegalArgumentException("Status not found with ID: " + dto.getStatusId()));

    Reservation reservation = new Reservation();
    reservation.setHall(hall);
    reservation.setSport(sport);
    reservation.setUser(user);
    reservation.setStatus(status);
    reservation.setDate(dto.getDate());
    reservation.setTimeFrom(dto.getTimeFrom());
    reservation.setTimeTo(dto.getTimeTo());
    reservation.setFinalPrice(dto.getFinalPrice());

    return reservationRepository.save(reservation);
}

```

Listing 3: Metoda *createReservation* u *ReservationService* sa validacijom i čuvanjem rezervacije

```

public Reservation fromDTO(ReservationDTO dto, Hall hall, Sport sport,
    User user, Payment payment, Status status) {
    if (dto == null) return null;

    Reservation reservation = new Reservation();
    reservation.setReservationId(dto.getReservationId());
    reservation.setHall(hall);
    reservation.setSport(sport);
    reservation.setUser(user);
    reservation.setStatus(status);
    reservation.setDate(dto.getDate());
    reservation.setTimeFrom(dto.getTimeFrom());
    reservation.setTimeTo(dto.getTimeTo());
    reservation.setFinalPrice(dto.getFinalPrice());

    return reservation;
}

```

Listing 4: Mapiranje iz DTO objekta u entitet u *ReservationService*

Kada su svi podaci validni, kreira se nova instanca klase *Reservation*, koja predstavlja entitet mapiran na tabelu *reservation* u bazi podataka. Ovaj objekat povezuje se sa ostalim entitetima (sala, sport, korisnik i status) putem anotacija *@ManyToOne*, čime se jasno definišu relacije između tabela u PostgreSQL bazi.

```

@Entity
@Table(name = "reservation")
public class Reservation extends BaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "reservation_id")
    private Long reservationId;

    @ManyToOne
    @JoinColumn(name = "hall_id")
    private Hall hall;

    @ManyToOne
    @JoinColumn(name = "sport_id")
    private Sport sport;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "status_id")
    private Status status;

    private LocalDate date;
    private LocalTime timeFrom;
    private LocalTime timeTo;
    private BigDecimal finalPrice;

    // Getteri i setteri

```

*Listing 5: Entitetska klasa Reservation mapirana na tabelu u bazi podataka*

Kao rezultat uspešne operacije, servis vraća DTO objekat koji sadrži osnovne informacije o kreiranoj rezervaciji – identifikator, izabranu salu, termin i cenu. Kontroler zatim formira HTTP odgovor u formatu JSON, koji sadrži poruku o uspešnom kreiranju i same podatke o rezervaciji, i prosleđuje ga klijentskoj aplikaciji.

Ovim primerom jasno se vidi struktura i podela odgovornosti unutar serverske strane aplikacije:

- **kontroler** prihvata zahtev i vraća odgovor,
- **servis** implementira poslovnu logiku i validaciju,
- **entitet** predstavlja mapu ka tabeli u bazi,
- **repozitorijum** omogućava trajno čuvanje podataka.

Na taj način se postiže čista arhitektura, lakše održavanje sistema i jasna podela slojeva, što čini aplikaciju skalabilnom i pouzdanom.

Serverska strana aplikacije zasniva se na *Spring Security* i *JWT (JSON Web Token)* mehanizmima kako bi obezbedila autentifikaciju i autorizaciju bez održavanja serverskih sesija (*stateless* pristup). Nakon uspešne prijave ili registracije, klijentu se izdaje JWT token koji se u svakom narednom zahtevu prosleđuje preko HTTP hедера **Authorization: Bearer <token>**. Na osnovu tog tokena, backend pouzdano identifikuje korisnika i primenjuje pravila pristupa prema dodeljenoj ulozi.

U pogledu obrade zahteva, ključnu ulogu ima namenski filter *JwtAuthFilter* (izveden iz *OncePerRequestFilter*). Za svaki dolazni zahtev filter proverava postojanje i format hедера *Authorization*, izdvaja JWT token, te preko servisa za rad sa tokenima (*JwtService*) sprovodi ekstrakciju subject-a (email) i validaciju tokena. Na osnovu dobijenog email-a učitava se korisnik iz skladišta (*UserRepository*). Ako je token ispravan i važeći za tog korisnika (*validateToken(...)*), kreira se *Authentication* objekat (tipa *UsernamePasswordAuthenticationToken*) sa pripadajućim autoritetima (npr. *ROLE\_ADMIN*, *ROLE\_USER*) i upisuje u *SecurityContext*. Na ovaj način ostatak aplikacije (kontroleri/servisi) dobija pouzdanu informaciju o identitetu i privilegijama korisnika bez dodatnih koraka.

Pravila pristupa centralno se konfigurišu u klasi *SecurityConfig* unutar *SecurityFilterChain* konfiguracije. Ovde se:

- uključuje **CORS** zbog komunikacije sa Angular klijentom (definiše se *allowed origins*, *methods*, *headers*, *eventualno exposed headers*),
- isključuje **CSRF** (jer se koristi *JWT* i *SessionCreationPolicy.STATELESS*),
- precizno navode javno dostupni endpoint-i (npr. */api/auth/login*, */api/auth/register*, kao i *Stripe webhook/checkout*),
- opcionalno dozvoljava *read-only* pristup za pojedine *GET* rute (npr. pregled *sala/sportova*),
- i štite zaštićeni resursi koji zahtevaju prijavu (npr. *\_/api/reservation/\_* za kreiranje/menjanje rezervacija, *\_/api/payment/*, *\*\*/api/user/*). Za strožu kontrolu administrativnih operacija koristi se *role-based* ograničenje (npr. *hasRole('ADMIN')* za kreiranje/izmenu/brisanje entiteta), a zahvaljujući *\*\*@EnableMethodSecurity\_*, pravila se mogu dodatno precizirati na nivou metoda anotacijama poput *@PreAuthorize*. U slučaju pokušaja pristupa bez validnog tokena, prilagođeni *authenticationEntryPoint* vraća *401 Unauthorized*, čime je komunikacija jasna i dosledna.

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .cors(Customizer.withDefaults())
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers(
                "/swagger-ui/**",
                "/v3/api-docs/**",
                "/swagger-resources/**",
                "/webjars/**"
            ).permitAll()
            .requestMatchers("/api/auth/login", "/api/auth/register").permitAll()
            .requestMatchers("/api/stripe/webhook", "/api/stripe/checkout").permitAll()
            .requestMatchers(HttpMethod.GET, "/api/hall/*", "/api/sport/*").permitAll()
            .requestMatchers(HttpMethod.POST, "/api/hall/*", "/api/status/", "/api/role/*").hasRole("ADMIN")
            .requestMatchers(HttpMethod.PUT, "/api/hall/*", "/api/status/", "/api/role/*").hasRole("ADMIN")
            .requestMatchers(HttpMethod.DELETE, "/api/hall/*", "/api/status/", "/api/role/*").hasRole("ADMIN")
            .requestMatchers("/api/reservation/*", "/api/payment/", "/api/user/*").authenticated()
            .anyRequest().denyAll()
        )
        .exceptionHandling(ex -> ex
            .authenticationEntryPoint((req, res, e) ->
                res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized")
            )
        )
        .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

```

Listing 6: Konfiguracija bezbednosnog lanca kroz metodu `securityFilterChain` u klasi `SecurityConfig`

Pri **registraciji** sistem najpre proverava jedinstvenost username i email vrednosti (sprečavanje duplikata), zatim učitava odgovarajuću ulogu iz *RoleRepository* i kreira korisnika uz obavezno heširanje lozinke algoritmom **BCrypt** (kroz `PasswordEncoder`). Nakon upisa korisnika u bazu, servis generiše JWT token (`jwtService.generateToken(user)`) i vraća ga klijentu. Time se obezbeđuje da novoregistrovani korisnik može odmah pristupiti zaštićenim resursima u okviru svoje uloge, a da se poverljivi podaci (lozinke) nikada ne čuvaju u čistom tekstu.



```

public String register(UserCreateDTO newUser) {
    if (userRepo.existsByUsername(newUser.getUsername())) {
        throw new RuntimeException(message:"Username already taken.");
    }

    if (userRepo.existsByEmail(newUser.getEmail())) {
        throw new RuntimeException(message:"Email already taken.");
    }

    Role userRole = roleRepo.findById(newUser.getRoleId())
        .orElseThrow(() -> new RuntimeException(message:"Role not found"));

    User user = new User();
    user.setFirstName(newUser.getFirstName());
    user.setLastName(newUser.getLastName());
    user.setUsername(newUser.getUsername());
    user.setEmail(newUser.getEmail());
    user.setPassword(passwordEncoder.encode(newUser.getPassword()));
    user.setPhoneNumber(newUser.getPhoneNumber());
    user.setBirthDate(newUser.getBirthDate());
    user.setRole(userRole);

    userRepo.save(user);

    return jwtService.generateToken(user);
}

```

Listing 7: Metoda register u AuthService za registraciju korisnika i izdavanje JWT tokena

Pri **prijavi (login)**, servis pronalazi korisnika na osnovu email-a, a potom proverava lozinku metodom `passwordEncoder.matches(...)`. U slučaju neusaglašenosti kredencijala vraća se jasna poruka o grešci; u suprotnom, izdaje se novi JWT token. Na strani klijenta token se čuva u skladu sa bezbednosnim smernicama i šalje se uz svaki sledeći zahtev kroz Authorization heder.

```

public String login(LoginDTO loginUser) {
    User user = userRepo.findByEmail(loginUser.getEmail())
        .orElseThrow(() -> new RuntimeException(message:"Invalid email or password"));

    if (!passwordEncoder.matches(loginUser.getPassword(), user.getPassword())) {
        throw new RuntimeException(message:"Invalid email or password");
    }

    return jwtService.generateToken(user);
}

```

Listing 8: Metoda login u AuthService za autentifikaciju korisnika i izdavanje JWT tokena

```

public String generateToken(User user) {
    return Jwts.builder()
        .setSubject(user.getEmail())
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + jwtProperties.getExpiration()))
        .signWith(key, SignatureAlgorithm.HS512)
        .compact();
}

```

Listing 9: Metoda generateToken u JwtService za kreiranje JWT tokena

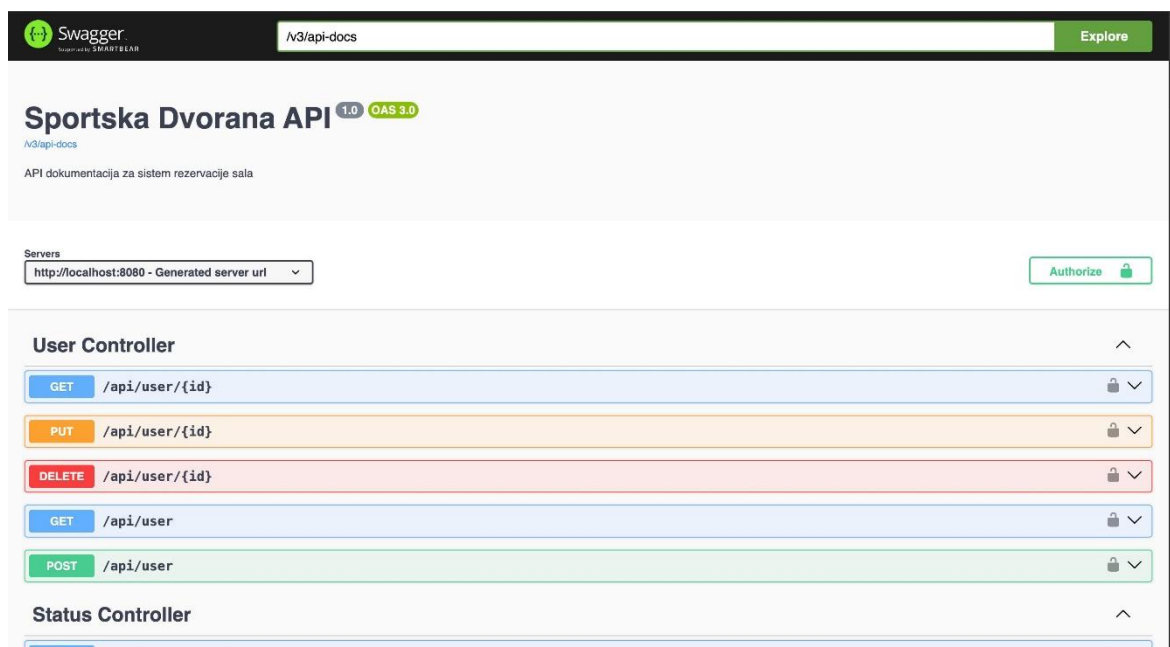
Ovakav model obezbeđenja spaja jednostavnost (token putuje sa zahtevom), efikasnost (nema server-side sesije), i fleksibilnost (centralizovana pravila pristupa i/ili

anotacije na metodima). Uz BCrypt heširanje lozinki, jasno razdvajanje javnih i zaštićenih ruta u SecurityConfig, te dosledan rad JwtAuthFilter i SecurityContext, sistem dobija stabilan i skalabilan bezbednosni sloj. Preporučuje se i da se rok važenja tokena postavi razumno (ograničen) i da se eliminiše unošenje osetljivih podataka u payload tokena (osim minimalnih tvrdnji kao što je subject = email i role), čime se dodatno smanjuje površina napada.

Napomena o integraciji sa klijentom. Angular aplikacija pre svake komunikacije ka zaštićenim endpoint-ima dodaje Authorization heder; zaštita ruta na klijentu (npr. route guards) može sprečiti pristup UI stranama bez tokena, ali se stvarna zaštita sprovodi na serveru kroz SecurityConfig i JwtAuthFilter.

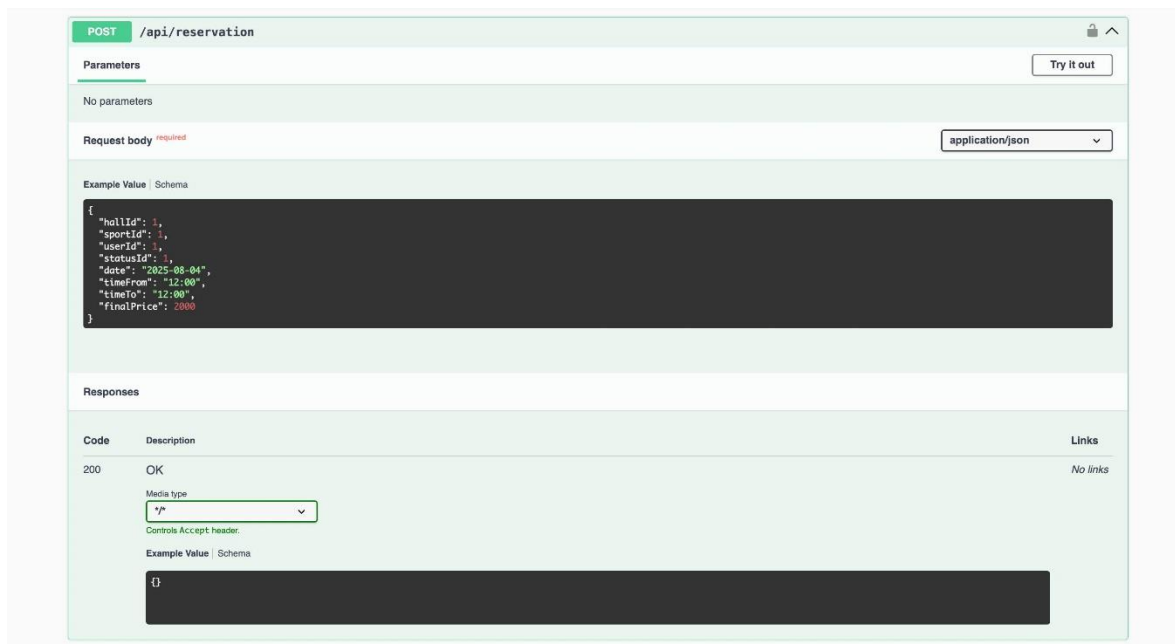
Za potrebe testiranja API-ja i obezbeđivanja jasne dokumentacije, u aplikaciju je integrisan **Swagger UI** na osnovu **OpenAPI** specifikacije. Ova integracija omogućava da se svi dostupni endpoint-i pregledaju kroz pregledan web interfejs, uz precizan opis metoda (**GET**, **POST**, **PUT**, **DELETE**), parametara i očekivanih odgovora. Na taj način, sistem rezervacije sportskih sala dobija interaktivnu dokumentaciju koja istovremeno služi kao alat za razvoj i kao verifikacija ispravnosti implementiranih funkcionalnosti.

Prikaz na slici 1. ilustruje inicijalni izgled Swagger UI-a sa listom dostupnih ruta i primerom testiranja POST metode /api/reservation koji se detaljno vidi na slici 2., kojom se kreira nova rezervacija. Na osnovu dokumentovanog endpoint-a, moguće je uneti potrebne parametre (identifikator sale, sport, korisnik, termin, status i cenu) i izvršiti zahtev, nakon čega Swagger prikazuje strukturu odgovora. Time se jasno može ispratiti kompletan proces: od slanja JSON objekta prema kontroleru, preko validacije u servisnom sloju, do kreiranja rezervacije i vraćanja odgovora u formatu JSON.



Slika 2 Swagger UI prikaz inicijalnog interfejsa





Slika 3 Swagger UI testiranje POST metode `/api/reservation`

Pored vizuelnog prikaza, konfiguracija Swagger-a implementirana je u posebnoj klasi **OpenApiConfig**, čiji je deo prikazan u Listingu 10. Ovom konfiguracijom definišu se osnovne informacije o API-ju (naziv aplikacije, verzija i opis), ali i bezbednosne šeme. Naime, u komponentama je dodata *bearerAuth* šema koja definiše da API koristi JWT token kao mehanizam autentifikacije, sa prefiksom ***Bearer***. Na ovaj način Swagger omogućava i testiranje endpoint-a koji su zaštićeni autentifikacijom, jer korisnik može uneti validan token i pristupiti resursima koji zahtevaju prijavu.

```
@Configuration
public class OpenApiConfig {

    private static final String SECURITY_SCHEME_NAME = "bearerAuth";

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("Sportska Dvorana API")
                .version("1.0")
                .description("API dokumentacija za sistem rezervacije sala"))
            .addSecurityItem(new SecurityRequirement().addList(SECURITY_SCHEME_NAME))
            .components(new Components()
                .addSecuritySchemes(SECURITY_SCHEME_NAME,
                    new SecurityScheme()
                        .name(SECURITY_SCHEME_NAME)
                        .type(SecurityScheme.Type.HTTP)
                        .scheme("bearer")
                        .bearerFormat("JWT"))));
    }
}
```

Listing 10: Klasa `OpenApiConfig` za definisanje OpenAPI dokumentacije i JWT autentifikacije

Kombinacijom **interaktivne dokumentacije** (Swagger UI) i **programske konfiguracije** (OpenApiConfig), sistem je dobio pouzdan i pregledan način testiranja svih funkcionalnosti, što je značajno olakšalo razvoj, otkrivanje grešaka i komunikaciju između backend i frontend timova.

### 3.1.1 Stripe integracija za online plaćanje rezervacija

Za potrebe **online naplate rezervacija** implementirana je integracija sa **Stripe Checkout** servisom. Rešenje obuhvata dva osnovna toka: (1) kreiranje Checkout Session objekta na strani servera i preusmeravanje korisnika na Stripe interfejs za unos kartičnih podataka, i (2) webhook obradu uspešno završenih plaćanja, kojom se trajno evidentira transakcija i ažurira status rezervacije u bazi.

#### 1. Kreiranje plaćanja (Checkout Session)

Kada korisnik potvrdi plaćanje, frontend šalje zahtev ka endpoint-u `/api/stripe/checkout` sa parametrima `reservationId` i `amount`. U metodi **createCheckoutSession** najpre se učitava ciljana rezervacija. Iznos se konvertuje u najmanju jedinicu valute ( $amount \times 100$ ), a zatim se formira opis stavke (sala, sport, datum, termin, korisnik). Parametri sesije obuhvataju `mode=PAYMENT`, `successUrl` i `cancelUrl`, jednu line item stavku sa `priceData`, kao i metadata (`reservationId`, `amount`, `userId`). Kreiranjem sesije backend dobija URL koji se vraća klijentu i na koji se korisnik preusmerava radi bezbednog unosa podataka. Na ovaj način aplikacija nikada ne obrađuje kartične podatke, već to u potpunosti obavlja Stripe.

```
public String createCheckoutSession(Long reservationId, Double amount) throws StripeException {
    Reservation reservation = reservationRepository.findById(reservationId)
        .orElseThrow(() -> new IllegalArgumentException("Reservation not found: " + reservationId));
    long stripeAmount = Math.round(amount * 100);
    String description = String.format(
        format:"RSD %.2f | Sport: %s | Datum: %s | Vreme: %s - %s | Korisnik: %s %s",
        amount, reservation.getSport().getSport(), reservation.getDate(), reservation.getTimeFrom(),
        reservation.getTimeTo(), reservation.getUser().getFirstName(), reservation.getUser().getLastName());
    SessionCreateParams params = SessionCreateParams.builder()
        .setMode(SessionCreateParams.Mode.PAYMENT)
        .setSuccessUrl("http://localhost:4200/payment-success?reservationId=" + reservationId)
        .setCancelUrl("http://localhost:4200/reservations?canceled=true")
        .addLineItem(
            SessionCreateParams.LineItem.builder()
                .setQuantity(quantity:1L)
                .setPriceData(
                    SessionCreateParams.LineItem.PriceData.builder()
                        .setCurrency(currency:"RSD")
                        .setUnitAmount(stripeAmount)
                        .setProductData(
                            SessionCreateParams.LineItem.PriceData.ProductData.builder()
                                .setName(description)
                                .build()
                        ).build()
                ).build()
        ).build()
        .putMetadata(key:"reservationId", reservationId.toString())
        .putMetadata(key:"amount", amount.toString())
        .putMetadata(key:"userId", reservation.getUser().getUserId().toString())
        .build();
    Session session = Session.create(params);
    return session.getUrl();
}
```

Listing 11: Metoda `createCheckoutSession` u klasi `StripeService` za kreiranje Stripe Checkout sesije

## 2. Obrada uspešnog plaćanja (Webhook).

Stripe asinhrono šalje obaveštenja o statusu plaćanja preko webhook mehanizma. Kontroler `StripeController` izlaže endpoint `/api/stripe/webhook`, koji verifikuje dolazni zahtev pozivom `Webhook.constructEvent` uz korišćenje tajnog webhook ključa. Nakon verifikacije, prema tipu događaja (npr. `checkout.session.completed`) izvlači se objekat `Session`. Ako je status `complete` i `paid`, poziva se metoda `handleSuccessfulCheckout`, gde se:

- validira i parsira `reservationId` iz metadata,
- učitava rezervaciju i proverava da li već postoji `Payment` za dati `stripeId`,
- upisuje novi `Payment` (`stripeId`, `amount`, `currency`, `payment_status`),
- ažurira `Status` rezervacije na „plaćena“.

Ova metoda je označena anotacijom `@Transactional`, čime se obezbeđuje da se i upis plaćanja i promena statusa izvrše atomarno. U slučaju izuzetka, vraća se jasna poruka, a greška se loguje.

```
@Transactional
public void handleSuccessfulCheckout(Session session) {
    try {
        if (!"complete".equals(session.getStatus()) || !"paid".equals(session.getPaymentStatus())) {
            throw new IllegalStateException("Session is not completed or payment not done.");
        }
        String reservationIdStr = session.getMetadata().get(key:"reservationId");
        if (reservationIdStr == null || reservationIdStr.trim().isEmpty()) {
            throw new IllegalArgumentException("Session does not contain reservationId in metadata. Metadata: " + session.getMetadata());
        }
        Long reservationId;
        try {
            reservationId = Long.parseLong(reservationIdStr);
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Invalid reservationId format: " + reservationIdStr);
        }
        Reservation reservation = reservationRepository.findById(reservationId)
            .orElseThrow(() -> new IllegalArgumentException("Reservation not found: " + reservationId));
        Optional<Payment> existingPayment = paymentRepository.findByStripeId(session.getId());
        if (existingPayment.isPresent()) return;
        Payment payment = new Payment();
        payment.setStripeId(session.getId());
        payment.setAmount(reservation.getFinalPrice());
        payment.setCurrency(currency:"RSD");
        payment.setPaymentStatus(paymentStatus:"PAID");
        payment.setReservation(reservation);
        paymentRepository.save(payment);
        Status paidStatus = statusRepository.findByStatus(status:"placena")
            .orElseThrow(() -> new IllegalArgumentException("Status 'placena' not found"));
        reservation.setStatus(paidStatus);
        reservationRepository.save(reservation);
    } catch (Exception e) {
        throw new RuntimeException("Failed to process successful checkout for session: " + session.getId(), e);
    }
}
```

Listing 12DijagrD: Metoda `handleSuccessfulCheckout` u klasi `StripeService` za obradu uspešno završenog plaćanja i ažuriranje rezervacije

## Bezbednosne napomene

Webhook ruta je javna, ali se svaki zahtev verifikuje pomoću potpisa. Ključne vrednosti (Stripe secret key i webhook secret) čuvaju se u konfiguracionim fajlovima (`application.properties` ili `StripeProperties`), nikada u kodu. U servisu je implementirana idempotentnost obrade – sprečava se duplo kreiranje plaćanja ako je isti `stripeId` već procesiran. Iznosi se obrađuju u celobrojnim jedinicama valute da bi se izbegle greške sa decimalama.

## Veza sa modelom podataka

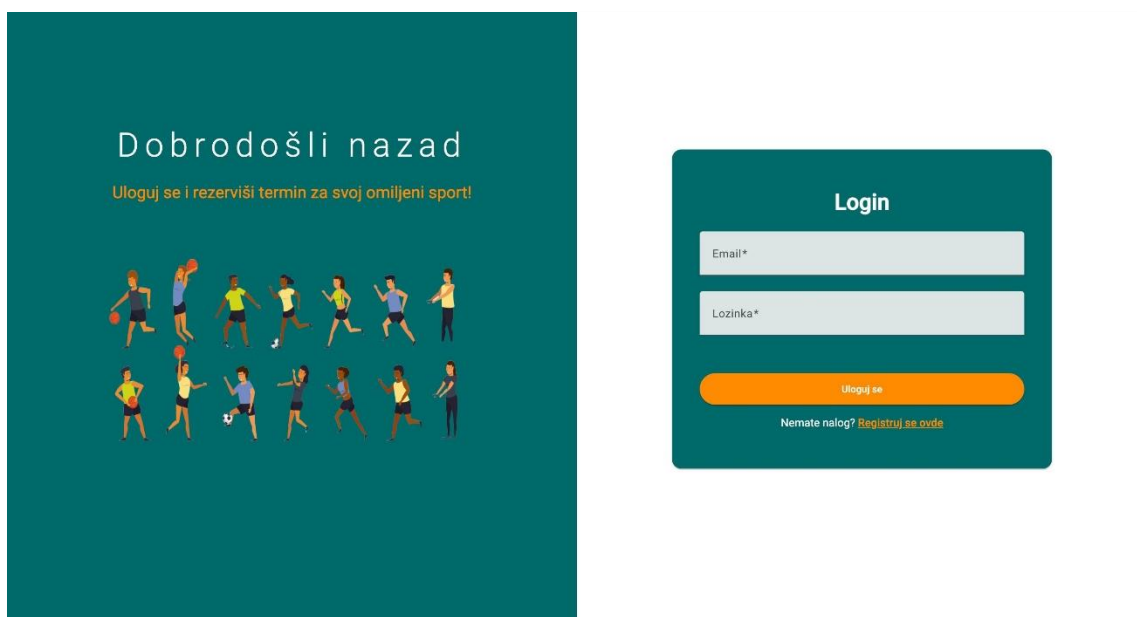
Entitet *Payment* referencira *Reservation* (relacija 1:1) i čuva podatke o transakciji (*stripe\_id*, *amount*, *currency*, *payment\_status*). Nakon uspešne uplate, status rezervacije se menja na „Plaćena“. Na ovaj način sistem obezbeđuje konzistentnost između poslovnog stanja rezervacije i finansijskog toka.

### 3.2 Klijentska strana aplikacije

U ovom odeljku se opisuje korisnički interfejs izrađen u Angular okruženju, sa fokusom na tok rada krajnjeg korisnika (pregled i rezervacija sala, plaćanje), kao i na administratorske funkcionalnosti (upravljanje salama, sportovima i statusima).

#### 3.2.1 Tok rada korisnika kroz aplikaciju

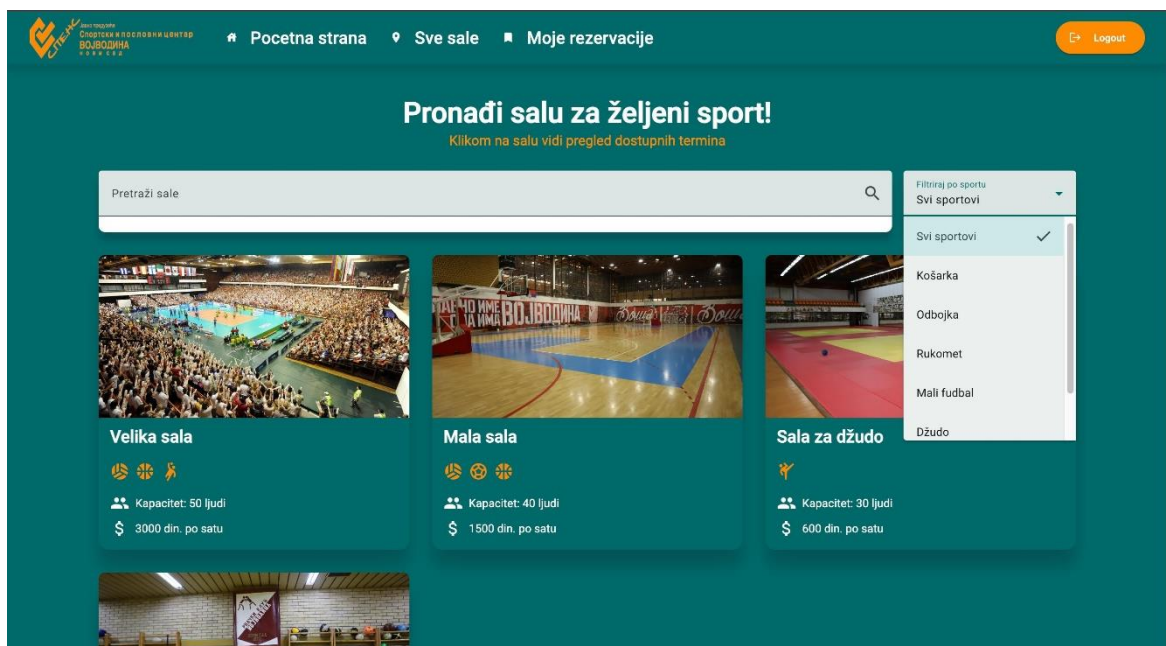
Prvi korak za korišćenje sistema jeste **prijava (login)** ili, ukoliko korisnik nema nalog, **registracija**. Na stranici za registraciju unose se osnovni podaci: ime, prezime, username, email, lozinka, kao i dodatni podaci poput broja telefona i datuma rođenja. Nakon uspešne registracije, korisniku se omogućava prijava na sistem korišćenjem e-mail adrese i lozinke.



Slika 4 Stranica za prijavljivanje u sistem



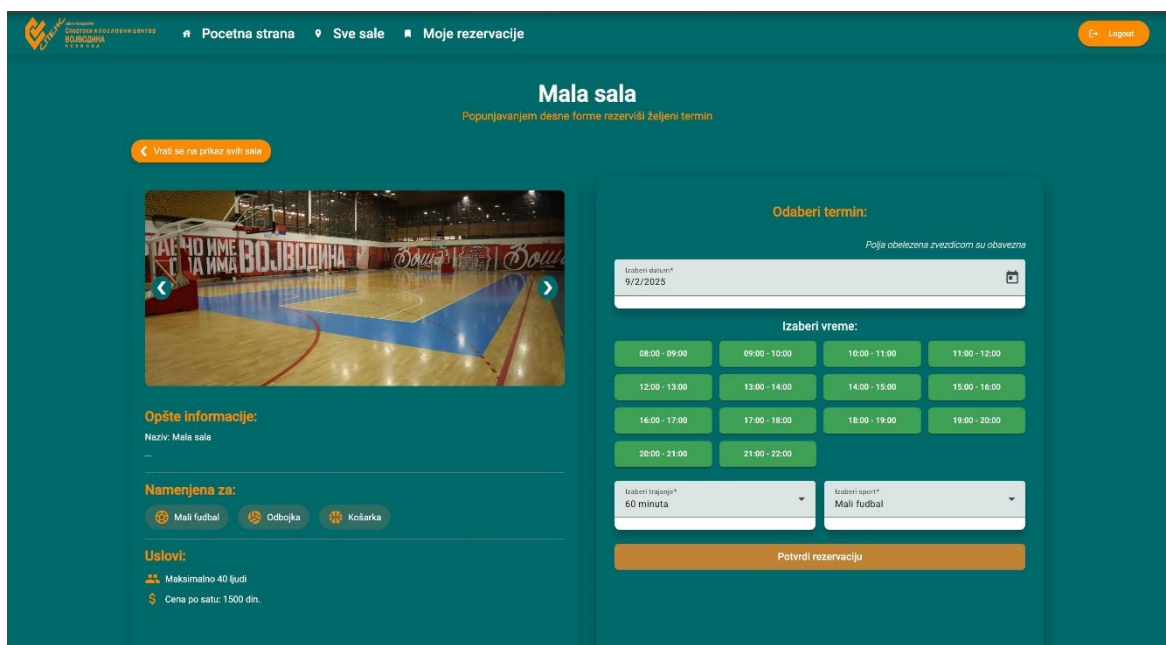




Slika 7 Lista sala sa filterom i pretragom

Klikom na određenu salu, korisnik prelazi na **detaljan prikaz sale**. Ova stranica je podeljena na dva dela:

- u **levom delu** nalaze se statičke informacije o sali, poput njenog naziva, kapaciteta, cene po satu i liste sportova koje je moguće igrati u tom prostoru;
- u **desnom delu** nalazi se dinamički deo interfejsa – formular za kreiranje rezervacije



Slika 8 Detaljan prikaz sale sa informacijama i formularom za rezervaciju

Kako bi rezervacija bila moguća, korisnik u formularu bira sve neophodne parametre: datum, vremenski interval (od – do), kao i željeni sport. Tek nakon popunjavanja svih traženih polja formular postaje validan i omogućava potvrdu rezervacije. Ovakav

pristup obezbeđuje da sistem dobije kompletne i ispravne podatke pre nego što rezervacija bude sačuvana u bazi.

**Mala Sala**  
Popunjavanjem desne forme rezerviši željeni termin

[Vrati se na prikaz svih sala](#)

**Opšte informacije:**  
Naziv: Mala sala

**Namenjena za:**  
Mini fudbal Odbojka Košarka

**Uslovi:**  
Maksimalno 40 ljudi  
Cena po satu: 1500 din.

**Odaberi termin:**  
Polja obeležena zvezdicom su obavezna

Izaberi datum\*  
9/2/2025

**Izaberi vreme:**

08:00 - 10:00	09:00 - 11:00	10:00 - 12:00	11:00 - 13:00
12:00 - 14:00	13:00 - 15:00	14:00 - 16:00	15:00 - 17:00
16:00 - 18:00	17:00 - 19:00	18:00 - 20:00	19:00 - 21:00
20:00 - 22:00			

Izaberi trajanje\*  
120 minuta

Izaberi sport\*  
Odbojka

**Rezervacija:**  
Datum: 02.09.2025  
Vreme: 13:00 - 15:00  
Sport: Odbojka  
Cena: 3000 din.

Potvrdi rezervaciju

Slika 9 Popunjen formular za kreiranje rezervacije

Kada korisnik potvrdi rezervaciju, aplikacija prikazuje poruku o uspehu. Ova poruka služi kao vizuelna potvrda da je rezervacija uspešno zabeležena u sistemu i da je korisnik sada može pronaći u sekciji „Moje rezervacije“. Na ovaj način korisniku se odmah pruža povratna informacija o rezultatu akcije, čime se povećava pouzdanost i jasnoća korišćenja aplikacije.

**Mala Sala**  
Popunjavanjem desne forme rezerviši željeni termin

[Vrati se na prikaz svih sala](#)

**Opšte informacije:**  
Naziv: Mala sala

**Namenjena za:**  
Mini fudbal Odbojka Košarka

**Uslovi:**  
Maksimalno 40 ljudi  
Cena po satu: 1500 din.

**Odaberi termin:**  
Polja obeležena zvezdicom su obavezna

Izaberi datum\*  
9/2/2025

**Izaberi vreme:**

08:00 - 10:00	09:00 - 11:00	10:00 - 12:00	11:00 - 13:00
12:00 - 14:00	13:00 - 15:00	14:00 - 16:00	15:00 - 17:00
16:00 - 18:00	17:00 - 19:00	18:00 - 20:00	19:00 - 21:00
20:00 - 22:00			

Izaberi trajanje\*  
120 minuta

Izaberi sport\*  
Odbojka

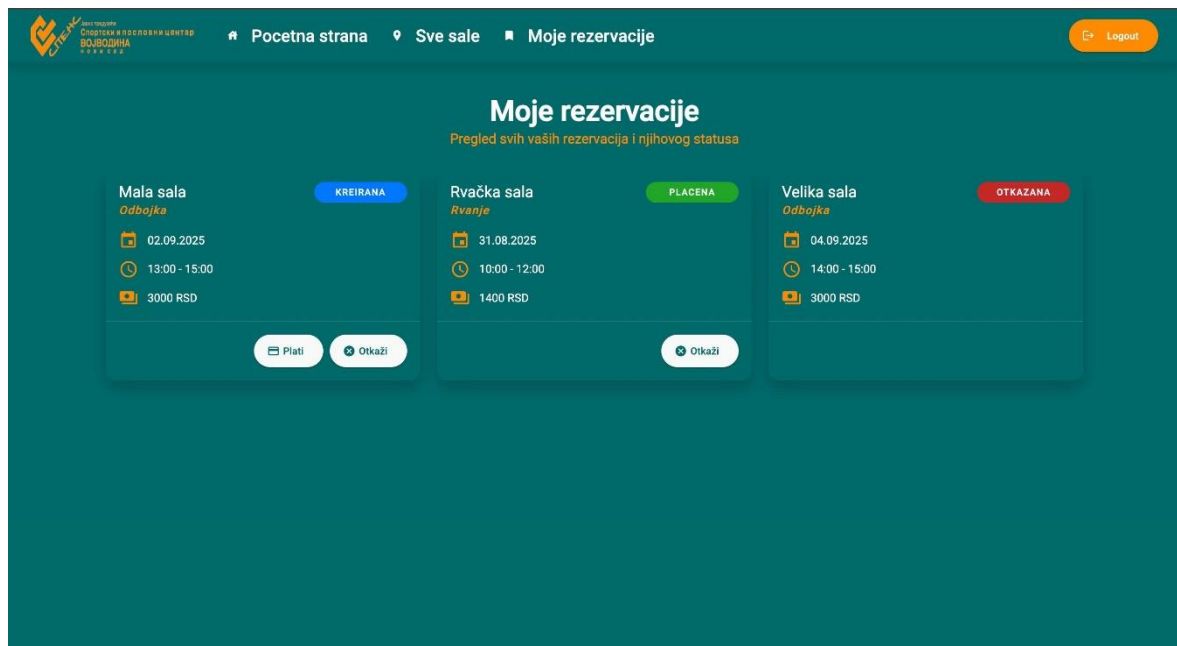
Potvrdi rezervaciju

Rezervacija uspešno kreirana!

Slika 10 Poruka o uspešno kreiranoj rezervaciji

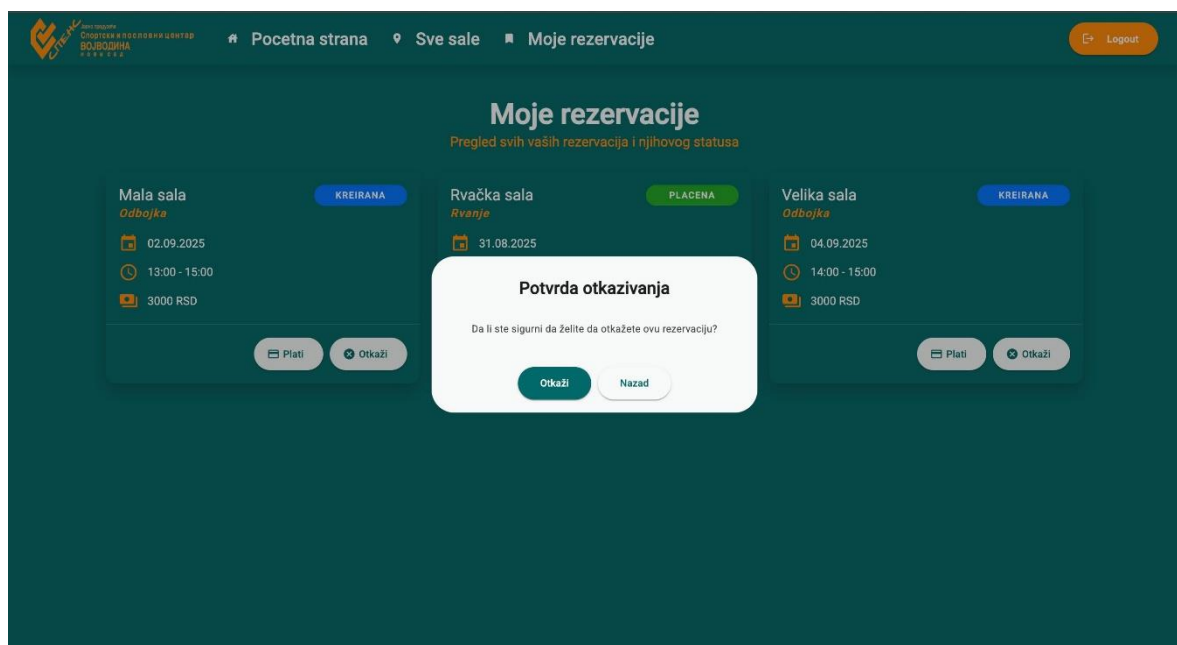
Nakon uspešnog kreiranja rezervacije, korisnik prelazi na stranicu „**Moje rezervacije**“. Ova stranica predstavlja centralno mesto gde su prikazane sve prethodno kreirane rezervacije, sa osnovnim informacijama o sali, sportu, datumu i terminu, kao i

statusom rezervacije. Korisniku su u ovom prikazu dostupne i akcije koje može preduzeti nad svakom rezervacijom.



Slika 11 Stranica „Moje rezervacije“ sa listom svih rezervacija

Jedna od ponuđenih akcija jeste **otkazivanje rezervacije**. Klikom na opciju „Otkazi“ otvara se potvrđujući prozor (popup), koji služi kao dodatna sigurnosna provera kako bi se izbeglo slučajno brisanje rezervacije. Tek nakon što korisnik potvrdi svoju odluku, rezervacija se briše i termin se oslobađa za dalju upotrebu.



Slika 12 Popup prozor za potvrdu otkazivanja rezervacije



Druga važna akcija jeste opcija **plaćanja rezervacije**. Kada korisnik pokrene opciju plaćanja, sistem ga automatski preusmerava na **Stripe Checkout** stranicu. Na ovom interfejsu korisnik unosi podatke sa svoje platne kartice, dok aplikacija sama vodi računa o sigurnosti i validaciji procesa.

TEST MODE

RSD 1400.00 | Sport: Rvanje | Datum: 2025-08-31 |  
Vreme: 10:00 - 12:00 | Korisnik: Bogdan Dangubic

**RSD 1,400.00**

Pay with link

Or

Email  
email@example.com

Payment method

☒ Card

Card information

1234 1234 1234 1234

MM / YY CVC

Cardholder name  
Full name on card

Country or region  
Serbia

☐ Google Pay

☐ Save my information for faster checkout  
Pay securely at Sofija Dangubic and everywhere is accepted.

Pay

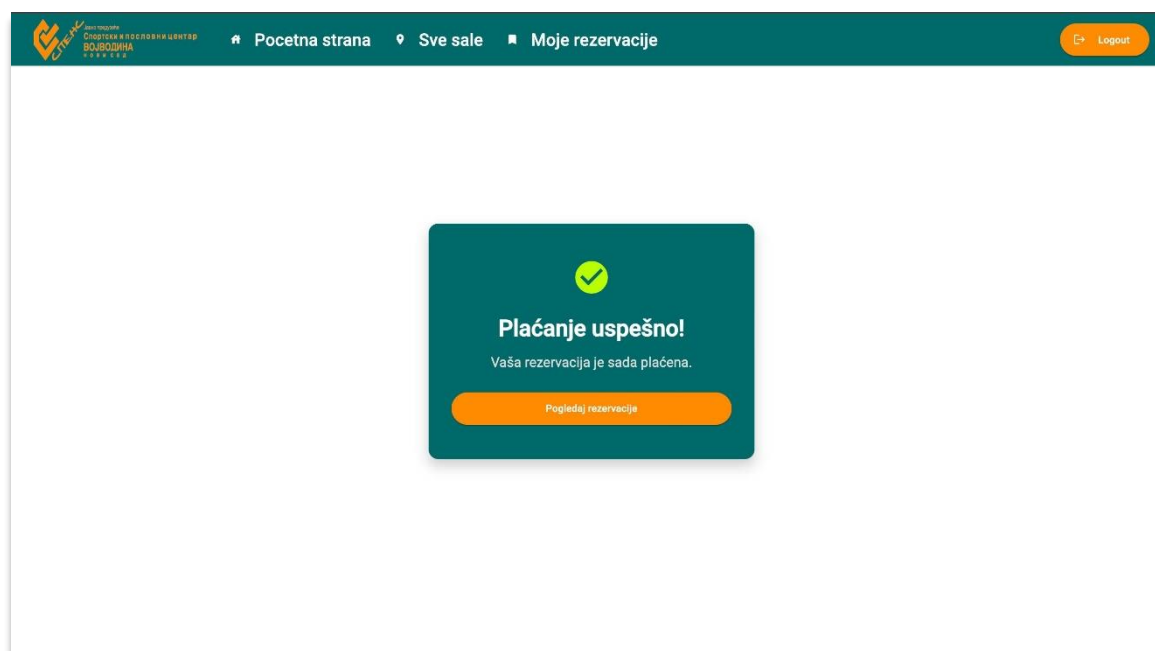
Sofija Dangubic

Powered by stripe | Terms | Privacy

localhost:4200/reservations?canceled=true

Slika 13 Stripe Checkout stranica za unos podataka o plaćanju

Nakon uspešne transakcije, korisnik se vraća u aplikaciju gde mu se prikazuje jasna poruka o uspešnom izvršenju plaćanja. Istovremeno, status rezervacije u bazi se menja u „plaćena“, što korisniku daje potpunu potvrdu da je rezervacija validna i spremna za korišćenje.

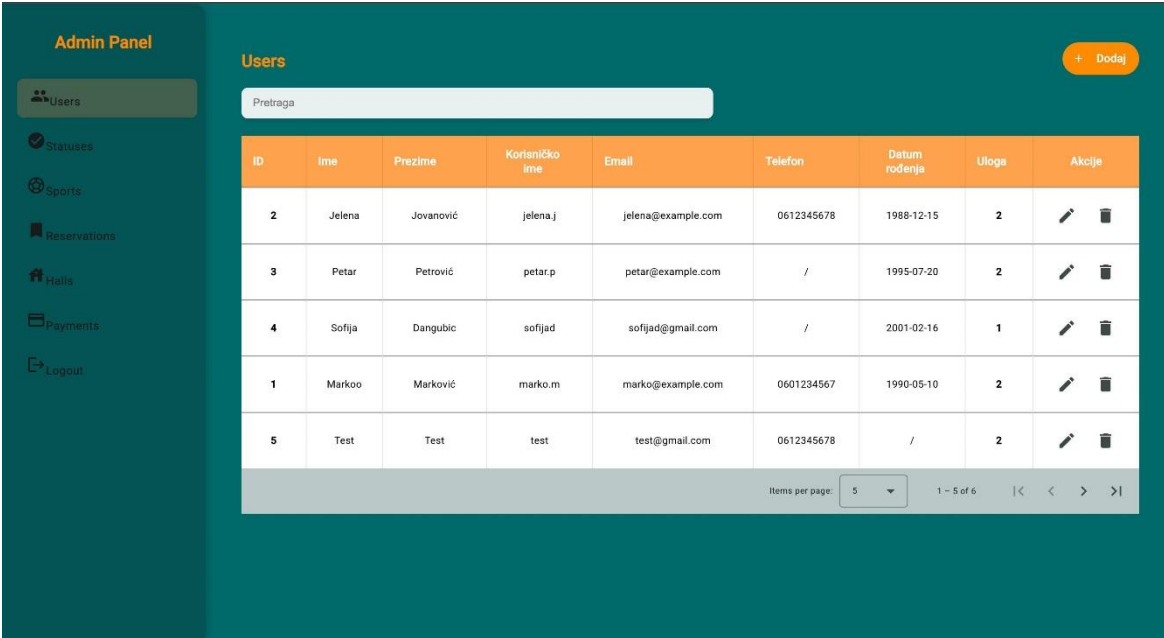












Slika 14 Stranica sa potvrdom da je plaćanje uspešno izvršeno

### 3.2.2 Administratorska strana aplikacije

Administratori se na sistem prijavljuju preko istog login interfejsa kao i obični korisnici, ali na osnovu svoje uloge dobijaju pristup dodatnim funkcionalnostima. Nakon što se administrator prijavi u sistem, dočekuje ga interfejs prilagođen njegovim ovlašćenjima. Na levoj strani ekrana nalazi se navigacioni panel, koji sadrži listu svih entiteta u sistemu – sale, sportove, statuse, korisnike, rezervacije i plaćanja. Klikom na bilo koju od ponuđenih opcija, u desnom delu ekrana otvara se odgovarajuća tabela sa podacima.

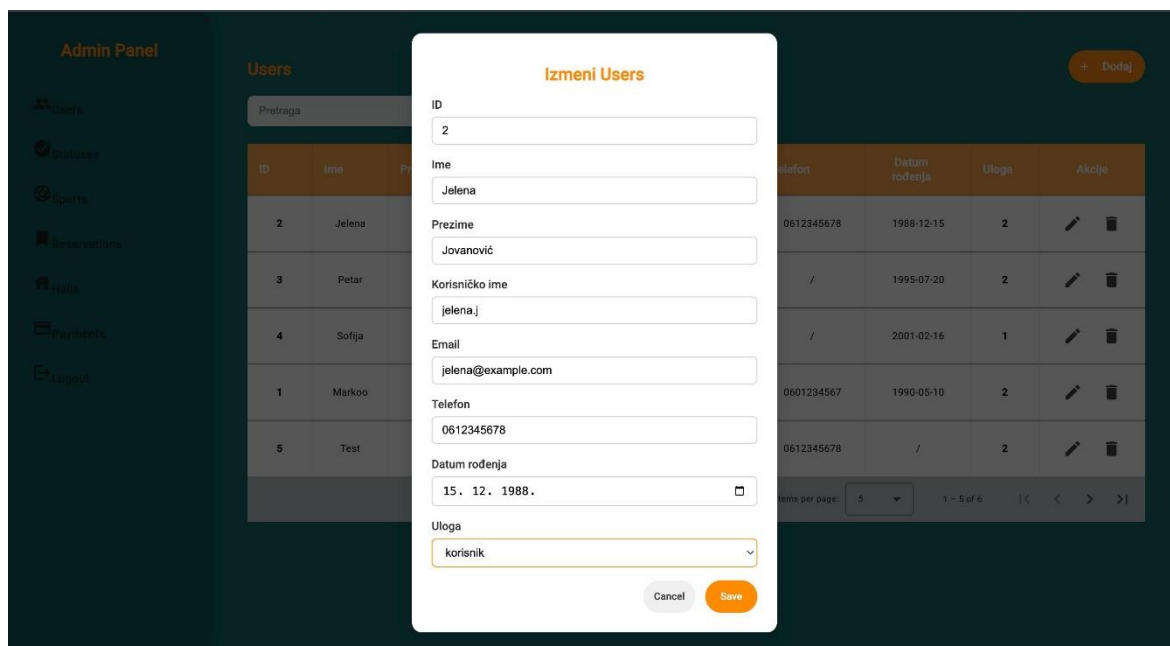
Tabela je organizovana tako da administrator može lako da pregleda i pretražuje podatke pomoću *search polja*, dok su dodatno omogućeni *sortiranje i paginacija*, čime se olakšava rad i nad većim skupovima podataka. Pored samog pregleda, svaka tabela nudi i akcione opcije u vidu dugmadi za dodavanje novog zapisa, izmenu postojećeg (olovka ikonica) i brisanje zapisa (kanta ikonica). Na ovaj način administrator ima potpunu kontrolu nad podacima koji se nalaze u sistemu.



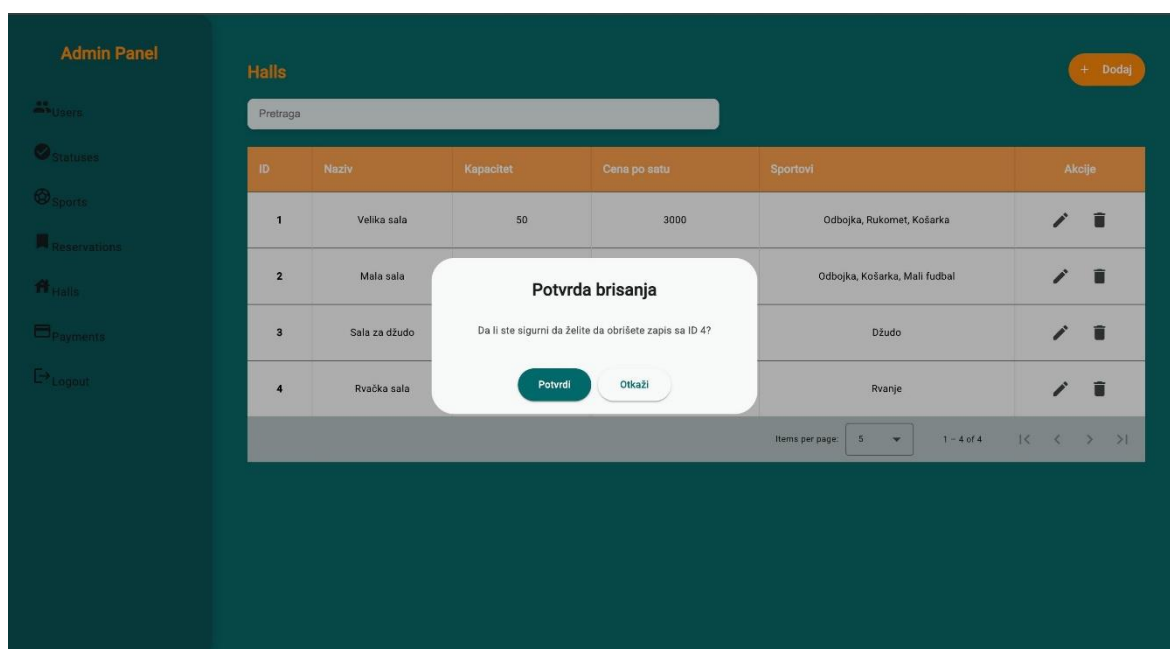
ID	Ime	Prezime	Korisničko ime	Email	Telefon	Datum rođenja	Uloga	Akcije
2	Jelena	Jovanović	jelena.j	jelena@example.com	0612345678	1988-12-15	2	 
3	Petar	Petrović	petar.p	petar@example.com	/	1995-07-20	2	 
4	Sofija	Dangubic	sofjad	sofjad@gmail.com	/	2001-02-16	1	 
1	Markoo	Marković	marko.m	marko@example.com	0601234567	1990-05-10	2	 
5	Test	Test	test	test@gmail.com	0612345678	/	2	 

Slika 15 Administratorski panel sa pregledom tabela

Akcije dodavanja i izmene realizovane su kroz **modalne prozore** (*popup forme*) u kojima administrator unosi ili menja podatke. Ovaj pristup obezbeđuje da se rad odvija u okviru istog ekrana, bez potrebe za dodatnim navigacijama, što doprinosi efikasnosti i preglednosti interfejsa. Brisanje zapisa, s druge strane, uvek je praćeno potvrdnim prozorčićem, čime se smanjuje mogućnost greške i slučajnog gubitka podataka.



Slika 16 Modalni prozor za izmenu zapisa



Slika 17 Potvrdni popup prozor za brisanje zapisa

Kao što se vidi iz opisanog toka, administrator ima potpun uvid u sve podatke u sistemu i ovlašćenja da ih menja, dodaje ili briše. Time se obezbeđuje da baza podataka ostane ažurna i u skladu sa realnim stanjem, dok se kroz jasno strukturisan interfejs omogućava brzo reagovanje i jednostavno upravljanje celokupnim sistemom rezervacija u sportskoj dvorani.

## 4 Zaključna razmatranja

Rad prikazuje kompletan proces razvoja informacionog sistema za rezervaciju sportskih sala, sa jasno razdvojenom serverskom i klijentskom stranom. Kroz primenu opasnih savremenih tehnologija, ostvareno je rešenje koje omogućava korisnicima jednostavan i pregledan način rezervisanja termina, dok administratorima pruža sve potrebne alate za nadzor i održavanje sistema.

Najznačajniji doprinos ovog rada ogleda se u demonstraciji kako kombinacija standardnih tehnologija i pažljivo osmišljene arhitekture može da rezultuje stabilnim, sigurnim i lako proširivim rešenjem. Poseban akcenat stavljen je na sigurnost (JWT autentifikacija, kontrola pristupa), kao i na integraciju sa eksternim servisom za plaćanje, čime se sistem približava realnim zahtevima modernih web aplikacija.

Dalja istraživanja i unapređenja mogu se kretati u više pravaca. Jedan od njih je uvođenje naprednijih algoritama za **optimizaciju termina** i automatsku detekciju preklapanja rezervacija. Takođe, sistem se može proširiti integracijom sa mobilnim aplikacijama ili uvođenjem notifikacionih servisa (e-mail, SMS, push obaveštenja) radi bolje komunikacije sa korisnicima. Dodatna vrednost mogla bi se postići i implementacijom analitičkih modula za praćenje iskorišćenosti sala, što bi pomoglo menadžmentu u donošenju odluka.

Na ovaj način, prezentovani sistem ne predstavlja samo tehnički rad već i osnovu za dalje unapređenje i istraživanja u oblasti informacionih sistema za sportske i srodne objekte. Time se potvrđuje značaj ovakvih rešenja u praksi, ali i njihova prilagodljivost budućim zahtevima i trendovima u razvoju softverskih aplikacija.

## Literatura

- BookaSport. (2024). *BookaSport – Online rezervacija sportskih terena*. Dostupno na: <https://www.bookasport.com/>
- Playo. (2024). *Playo – Meet. Play. Follow*. Dostupno na: <https://playo.co/>
- Spring IO. (n.d.) *Spring Boot – Official Documentation* [online]. Dostupno na: <https://spring.io/projects/spring-boot>
- Angular. (n.d.) *Angular – Official Website* [online]. Dostupno na: <https://angular.io>
- Stripe. (n.d.) *Online Payment Platform – Stripe Documentation* [online]. Dostupno na: <https://stripe.com/docs>
- PostgreSQL Global Development Group. (n.d.) *PostgreSQL Documentation* [online]. Dostupno na: <https://www.postgresql.org/docs/>