Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektronsko poslovanje

Tema seminarskog rada

|  |  |  |  |
| --- | --- | --- | --- |
| Rbr | Ime | Prezime | Broj indeksa |
| 1. |  |  |  |
| 2. |  |  |  |
| Mentor | |  | |
| GitHub link | | https://github.com/elab-development/internet-tehnologije-2024-projekat-rezervacijasala\_2021\_0247 | |

Sadržaj

[1. UVOD 5](#_Toc211265036)

[2. METODE I TEHNOLOGIJE ZA RAZVOJ VEB APLIKACIJE 6](#_Toc211265037)

[2.1. Uprošćena Larmanova metoda 6](#_Toc211265038)

[2.2. Tehnologije za razvoj *frontend* dela aplikacije 6](#_Toc211265039)

[2.2.1. JavaScript 6](#_Toc211265040)

[2.2.2. React 7](#_Toc211265041)

[2.3. Tehnologije za razvoj *backend* dela aplikacije 9](#_Toc211265042)

[2.3.1. PHP 9](#_Toc211265043)

[2.3.2. Laravel 10](#_Toc211265044)

[3. IMPLEMENTACIJA VEB APLIKACIJE 11](#_Toc211265045)

[3.1. Faza prikupljanja korisničkih zahteva 11](#_Toc211265046)

[3.1.1. Verbalni opis 11](#_Toc211265047)

[3.1.2. Slučajevi korišćenja 12](#_Toc211265048)

[SK1: Slučaj korišćenja – Registracija korisnika 12](#_Toc211265049)

[SK2: Slučaj korišćenja – Prijava korisnika 13](#_Toc211265050)

[SK3: Slučaj korišćenja – Odjava korisnika 13](#_Toc211265051)

[SK4: Slučaj korišćenja – Reset lozinke 13](#_Toc211265052)

[SK5: Slučaj korišćenja – Pregled kataloga sala 14](#_Toc211265053)

[SK6: Slučaj korišćenja – Pretraga i filtriranje sala 14](#_Toc211265054)

[SK7: Slučaj korišćenja – Rezervacija sale 14](#_Toc211265055)

[SK8: Slučaj korišćenja – Uređivanje rasporeda sala na mapi (admin) 15](#_Toc211265056)

[SK9: Slučaj korišćenja – Upravljanje salama (kreiranje, izmena, deaktivacija) 15](#_Toc211265057)

[SK10: Slučaj korišćenja – Pregled svih rezervacija i odobravanje/odbijanje 16](#_Toc211265058)

[3.2. Faza analize 16](#_Toc211265059)

[3.2.1. Sistemski dijagrami sekvence 17](#_Toc211265060)

[DS1: Dijagram sekvenci – Registracija korisnika 17](#_Toc211265061)

[DS2: Dijagram sekvenci – Prijava korisnika 18](#_Toc211265062)

[DS3: Dijagram sekvenci – Odjava korisnika 18](#_Toc211265063)

[DS4: Dijagram sekvenci – Reset lozinke 19](#_Toc211265064)

[DS5: Dijagram sekvenci – Pregled kataloga sala 20](#_Toc211265065)

[DS6: Dijagram sekvenci – Pretraga i filtriranje sala 21](#_Toc211265066)

[DS7: Dijagram sekvenci – Rezervacija sale 22](#_Toc211265067)

[DS8: Dijagram sekvenci – Uređivanje rasporeda sala na mapi (admin) 23](#_Toc211265068)

[DS9: Dijagram sekvenci – Upravljanje salama (kreiranje, izmena, deaktivacija) 24](#_Toc211265069)

[DS10: Dijagram sekvenci – Pregled svih rezervacija i odobravanje/odbijanje 27](#_Toc211265070)

[3.2.2. Struktura softverskog sistema 29](#_Toc211265071)

[3.3. Faza projektovanja 30](#_Toc211265072)

[3.3.1. Projektovanje korisničkog interfejsa 30](#_Toc211265073)

[3.3.2. Projektovanje aplikacione logike 38](#_Toc211265074)

[Laravel Kontroleri 38](#_Toc211265075)

[Laravel Autentifikacija 52](#_Toc211265076)

[Laravel Resursi 59](#_Toc211265077)

[Laravel Rutiranje 60](#_Toc211265078)

[Laravel Database Seeder 61](#_Toc211265079)

[React Hooks 62](#_Toc211265080)

[React Komponente 64](#_Toc211265081)

[React Rutiranje 142](#_Toc211265082)

[Laravel i React 145](#_Toc211265083)

[3.3.3. Projektovanje strukture softverskog sistema 147](#_Toc211265084)

[Migracije 147](#_Toc211265085)

[Modeli 148](#_Toc211265086)

[3.3.4. Projektovanje skladišta podataka 154](#_Toc211265087)

[3.4. Faza implementacije 155](#_Toc211265088)

[3.5. Faza testiranja 157](#_Toc211265089)

[4. ZAKLJUČAK 158](#_Toc211265090)

[5. LITERATURA 159](#_Toc211265091)

# UVOD

Ovaj seminarski rad obrađuje razvoj veb aplikacije za rezervaciju sala zasnovane na arhitekturi klijent–server, sa Laravel-om na serverskoj strani i React-om na klijentskoj. U uvodnim poglavljima predstavljamo motivaciju i cilj sistema: brzo pronalaženje i zakazivanje adekvatnog prostora, kao i efikasnu administraciju sala, rezervacija i preporuka. Kroz uprošćenu Larmanovu iterativnu metodu prolazimo faze od zahteva i analize (use-case scenariji, dijagrami sekvenci, model domena), preko projektovanja (UI, aplikaciona logika, REST API i šema baze) do implementacije i testiranja. Poseban fokus je na jasnom razdvajanju odgovornosti između frontenda i backenda, standardizovanom JSON API-ju, autentikaciji zasnovanoj na tokenu i kvalitetnoj organizaciji koda, što omogućava skalabilnost, lakše održavanje i nadogradnju sistema

# METODE I TEHNOLOGIJE ZA RAZVOJ VEB APLIKACIJE

## Uprošćena Larmanova metoda

Larmanov pristup razvoju softvera obuhvata sve glavne etape životnog ciklusa sistema — od prikupljanja zahteva, preko analize i dizajna, do implementacije i testiranja — ali ih organizuje u kratke, ponavljajuće cikluse. Umesto jednosmernog „vodopada“, rad se deli na niz iteracija u kojima se svaki put obrađuje manji deo funkcionalnosti, pa se sistem postepeno dograđuje i prilagođava promenama. Na taj način smanjuje se rizik promašaja i omogućava brže reagovanje na nova saznanja do kojih se dolazi tokom rada.

U početku se beleže potrebe i očekivanja korisnika, često kroz razgovore i use-case scenarije. Potom sledi razrada tih zahteva u okviru analize, gde se oblikuje logički pogled na sistem, prepoznaju ključni pojmovi i njihove veze i definišu pravila koja treba podržati. Na temeljima analize gradi se dizajn rešenja: određuje se arhitektura, razdvajaju slojevi, planiraju komponente, baze podataka, interfejsi i algoritmi. UML dijagrami klasa, komponenti i sekvenci često služe za jasno prikazivanje strukture i ponašanja.

Nakon dizajna dolazi razvoj koda i različiti nivoi provere ispravnosti. Jedinice koda se testiraju izolovano, zatim se proverava njihovo zajedničko funkcionisanje, a na kraju se ispituje ceo sistem u okruženju što sličnijem stvarnom. Rezultati svake iteracije procenjuju se i koriste za doradu zahteva ili prepravku dizajna u narednim ciklusima, tako da rešenje korak po korak sazreva do stabilne verzije koja ispunjava očekivanja korisnika.

## Tehnologije za razvoj *frontend* dela aplikacije

Frontend ili korisnički deo veb aplikacije razvijen je korišćenjem savremenih tehnologija, pre svega programskog jezika **JavaScript** i biblioteke **React**. Ove tehnologije zajedno omogućavaju izradu interaktivnog korisničkog interfejsa koji se izvršava u veb pregledaču.

### JavaScript

JavaScript je ključni skript jezik za izradu interaktivnog korisničkog dela veb aplikacija. Dinamičan je, slabo tipiziran i interpretiran, a standardizovan kroz ECMAScript. Nastao je sredinom devedesetih u Netscape-u, autor je Brendan Ajk, prvo se zvao LiveScript, pa je preimenovan u JavaScript, što je često zbunjivalo zbog sličnosti u nazivu sa Javom. Danas, zajedno sa HTML-om i CSS-om, čini osnovu modernog veba i podržan je u svim savremenim pregledačima.

Izvršava se na strani klijenta, unutar pregledača, i omogućava da stranice budu žive i responzivne: validacija formi u hodu, menjanje sadržaja bez osvežavanja (AJAX), animacije, interaktivne mape i slične funkcionalnosti. Kod se može ugraditi direktno u HTML preko script taga ili izdvojiti u .js fajlove. Jezik kombinuje prototipski objektni model i funkcionalne ideje; funkcije su objekti prvog reda i tipičan rad je zasnovan na događajima kao što su klikovi, tastatura ili učitavanje stranice.

Sintaksa podseća na C/Java stil sa vitičastim zagradama i uobičajenim kontrolnim strukturama, ali bez statičke tipizacije. Tip promenljive određuje se u toku izvođenja i može da se menja, a prisutni su brojčani, tekstualni, logički, undefined i null tipovi, kao i objekti koji obuhvataju i funkcije; nizovi su posebna vrsta objekata. Savremeni standardi uvode sintaksu za klase koja se iznutra i dalje oslanja na prototipe. Pošto je slabo tipiziran, jezik često obavlja implicitne konverzije (na primer spajanje brojeva i stringova), što olakšava rad, ali zahteva pažnju da bi se izbegla iznenađenja u ponašanju koda.

### React

React (često nazivan i React.js) je biblioteka otvorenog koda za izgradnju korisničkih interfejsa u JavaScript-u. Nastao je u Facebooku početkom 2010-ih, najpre je korišćen za News Feed 2011, a kao open-source objavljen 2013. Od tada je postao standard u frontendu i našao se u produkciji velikog broja kompanija i projekata.

Suština React-a je da olakša pravljenje dinamičkih, interaktivnih ekrana kroz sastavljanje malih, višekratno upotrebljivih delova — komponenti. Umesto direktnog rada nad DOM-om, React koristi virtuelni DOM: održava u memoriji laganu reprezentaciju prikaza, upoređuje staro i novo stanje i u pravi DOM primenjuje samo minimalne izmene. Ovakav pristup smanjuje trošak ponovnog renderovanja i čini ažuriranja bržim i predvidljivijim.

Komponente su osnovna građevna jedinica: enkapsuliraju izgled i ponašanje (logiku). Najčešće se pišu kao funkcije koje vraćaju JSX — proširenje sintakse koje dozvoljava da se u JavaScript-u piše markup nalik HTML-u. JSX nije obavezan, ali čini strukturu interfejsa jasnijom; u pozadini se prevodi u standardne JavaScript pozive.

Svaka komponenta može imati sopstveno stanje (state), koje se menja tokom rada, i prima svojstva (props) koja dolaze od roditelja. React forsira jednosmeran tok podataka: props idu nadole ka potkomponentama, a događaji i promene se signalizuju nagore preko callback funkcija. To pojednostavljuje razumevanje toka informacija i održavanje koda. Kada je potrebno centralizovano ili deljeno stanje kroz veći deo aplikacije, često se koriste React Context ili biblioteke poput Redux-a, dok sam React ostaje fokusiran pre svega na „View“ sloj — prikaz i interakciju.

## Tehnologije za razvoj *backend* dela aplikacije

Serverski (backend) deo aplikacije izrađen je korišćenjem tehnologija **PHP** i **Laravel**. PHP predstavlja programski jezik u kojem je napisan glavni deo poslovne logike na serveru, dok je Laravel moderni razvojni okvir (framework) koji pruža strukturu i mnoge gotove funkcionalnosti za brži i organizovaniji razvoj u PHP-u.

### PHP

PHP je široko korišćen skript jezik opšte namene namenjen izradi dinamičkih veb aplikacija, koji se izvršava na strani servera. Prvobitno je skraćenica značila “Personal Home Page”, jer je nastao kao skup skripti koje je sredinom devedesetih Rasmus Lerdorf koristio za svoju ličnu prezentaciju. Kasnije je naziv dobio današnje značenje “PHP: Hypertext Preprocessor”, čime je naglašena uloga u generisanju HTML-a i drugih formata na serveru.

Prve verzije objavljene 1994. vrlo brzo su postale open-source, a dodatni zamah jezik dobija kada su Zeev Suraski i Andi Gutmans tokom 1997–1998. preoblikovali osnovu i predstavili PHP 3, sa boljom sintaksom i snažnijom podrškom za baze. PHP 4 iz 2000. donosi znatno brže izvršavanje zahvaljujući Zend engine-u, a PHP 5 iz 2004. uvodi zrelu podršku za objektno-orijentisano programiranje. Savremene grane, PHP 7 i PHP 8, dodatno ubrzavaju rad i donose nove jezičke mogućnosti, pri čemu zajednica otvorenog koda i dalje aktivno razvija ekosistem.

Tipična upotreba podrazumeva .php fajlove u kojima se meša HTML i PHP kod. Serverov interpreter obrađuje skriptu, generiše rezultat i šalje ga pregledaču, pa se sadržaj može menjati u zavisnosti od korisničkog unosa, podataka iz baze, kolačića ili stanja sesije. Jezik ima bogat standardni skup funkcija i ekstenzija za rad sa fajlovima, nizovima, mrežom, sesijama i bezbednošću, pa je pogodan za brzu izradu funkcionalnih aplikacija.

Tokom vremena PHP je prerastao iz pretežno proceduralnog pristupa u jezik sa potpunim OOP mogućnostima: klase, nasleđivanje, interfejse, apstraktne klase i izuzetke. Programer može birati stil u zavisnosti od kompleksnosti zadatka, od malih skripti do složenih arhitektura. U praksi se često koristi sa relacionim bazama, najčešće MySQL ili MariaDB, u okviru LAMP okruženja, a pristup podacima se ostvaruje preko ekstenzija ili ORM alata, čime backend jednostavno komunicira sa bazom i isporučuje podatke ka frontendu.

### Laravel

Laravel je PHP okvir za izgradnju veb aplikacija koji donosi jasnu strukturu i obilje gotovih delova kako bi razvoj kompleksnih sistema bio brži i uredniji. Nastao je 2011. godine kao projekat Tejlora Otvela i zamišljen je kao napredniji naslednik CodeIgniter-a, sa funkcijama koje tada nedostajale — pre svega ugrađenom autentikacijom i boljom modularnošću. Laravel prati MVC pristup: Model upravlja podacima i bazom, View predstavlja sloj prikaza (HTML šabloni), a Controller sadrži poslovnu logiku i povezuje sve u celinu. Iako „ispod haube“ koristi mnoge Symfony komponente, Laravel nudi jednostavniju, čitljiviju sintaksu.

Jedna od glavnih prednosti je bogat set funkcionalnosti koje pokrivaju tipične potrebe aplikacije. Autentikacija i autorizacija su uključene „iz kutije“, pa registracija, prijava i pravila pristupa ne moraju da se pišu od nule. Za rad sa bazom koristi se Eloquent ORM: umesto ručnih SQL upita, tabele predstavljaju PHP klase i nad njima se rade operacije (kreiranje, čitanje, ažuriranje) na prirodan, objekat-orijentisan način.

Strukturu baze je moguće držati pod kontrolom kroz migracije — verzionisane skripte koje opisuju promene šeme i olakšavaju timski rad i razmeštanje na različita okruženja. Za prikaz koristi se Blade, Laravelov šablonski jezik koji omogućava upotrebu uslova, petlji i uključivanja delova šablona uz veoma čitljivu sintaksu; pri izvršavanju, Blade se prevodi u čisti PHP, pa nema suvišnog overheada. Dodatno, Artisan komandna linija ubrzava svakodnevne zadatke: generisanje kontrolera i modela, pokretanje migracija, sejanje podataka i start lokalnog servera.

Aplikacije su organizovane predvidivo: app/ sadrži modele i kontrolere, resources/views Blade šablone, routes/ definiciju ruta, a config/ podešavanja. Definisanje rute izgleda čitljivo (npr. GET zahtev ka /korisnici mapira se na UserController@index), a u samom kontroleru često se direktno koriste Eloquent modeli ili se oslanja na dependency injection za servise. U Blade-u se vrednosti ispisuju pomoću {{ ... }}, a direktive tipa @if i @foreach pojednostavljuju prikaz dinamičkog sadržaja. Sve zajedno, Laravel naglašava izražajnu sintaksu i dobre prakse, što u praksi znači manje „bojnog“ koda i brže isporučivanje funkcionalnosti.

# IMPLEMENTACIJA VEB APLIKACIJE

## Faza prikupljanja korisničkih zahteva

### Verbalni opis

Aplikacija služi zaposlenima da brzo nađu i zakažu odgovarajuću salu, a menadžerima i administratorima da upravljaju prostorima, rasporedom i rezervacijama. Ulazna tačka je katalog sala: korisnik ukuca svoje preferencije (tip prostorije, očekivan broj ljudi, željeni datum i termin, pa i „tip događaja“ poput sastanka ili radionice). Na osnovu toga front odmah rangira sale i predlaže najbolje kandidate. Rangiranje je poenima: bolje se kotiraju sale čiji tip odgovara želji, čiji kapacitet je najbliži potrebama i koje su aktivne; možeš i da postaviš kapacitet kao „tvrdu ogradu“ pa se prikazuju samo sale u tom rasponu. Po potrebi, korisnik može potpuno ignorisati postojeće rezervacije – ideja je da preporuka odražava njegove ulaze, ne istoriju sistema.

Iznad „pametnih“ predloga, katalog nudi i klasičan rad: pretraga po nazivu/opisu, filtriranje po tipu i kapacitetu, sortiranje, paginacija i instant prikaz dostupnosti za dati raspon vremena (kada se koristi mod sa proverom zauzeća). Iz svakog rezultata moguće je otvoriti formu za rezervaciju, koja automatski popunjava izabrani termin i salu. Predlog koji ti se dopada možeš i da „sačuvaš kao preporuku“ u bazi — backend pamti ko je predlog napravio, za koju salu i koji termin.

Za prostornu orijentaciju postoji vizuelna „Mapa sprata“. Prikazuje sobe kao kartice na mreži, označava dostupnost u zadatom terminu i omogućava da klikom kreneš u rezervaciju. Administrator tu ima specijalnu mogućnost: uključuje „edit mode“ i prevlačenjem menja raspored (koordinate/veličine) soba, a izmena se snima preko API-ja. Obični korisnici mapu samo gledaju i rezervišu – ne mogu da uređuju raspored.

Administratorski deo obuhvata nadzor nad salama, rezervacijama i preporukama. Admin vidi sve preporuke iz sistema, može da ih pretražuje, sortira, menja osnovne podatke (sala, datum, vreme, tip događaja) ili briše. U listama se, pored ID-jeva, pojavljuju i „lepa imena“: naziv sale i ime korisnika koji je preporuku kreirao, zahvaljujući obogaćenim API resursima.

### Slučajevi korišćenja

Slučajevi korišćenja (eng. use cases) predstavljaju način za opisivanje funkcionalnosti koje softverski sistem treba da pruži krajnjim korisnicima. Oni opisuju kako korisnik (ili neki drugi sistem) ostvaruje interakciju sa aplikacijom u cilju postizanja određenog cilja. Svaki slučaj korišćenja jasno definiše ko su učesnici, koji su preduslovi, osnovni i alternativni scenariji, kao i očekivani ishod. Služe kao osnova za komunikaciju između klijenata, korisnika i razvojnog tima, jer omogućavaju precizno razumevanje zahteva i ponašanja sistema. Pored toga, koriste se za planiranje testiranja i razvoj funkcionalnosti jer oslikavaju realne potrebe korisnika.

Primeri slučajeva korišćenja u našoj aplikaciji:

1. Registracija korisnika
2. Prijava korisnika
3. Odjava korisnika
4. Reset lozinke
5. Pregled kataloga sala
6. Pretraga i filtriranje sala po tipu, kapacitetu i statusu
7. Rezervacija sale
8. Uređivanje rasporeda sala na mapi (admin)
9. Upravljanje salama (kreiranje, izmena, deaktivacija)
10. Pregled svih rezervacija i odobravanje/odbijanje
11. Pregled svih preporuka

#### SK1: Slučaj korišćenja – Registracija korisnika

**Naziv SK:** Registracija korisnika  
**Aktori SK:** Gost (neulogovani korisnik)  
**Učesnici SK:** Gost i sistem  
**Preduslov:** Gost je otvorio stranicu za registraciju.  
**Osnovni scenario SK:**

1. Gost unosi ime, email i lozinku (sa potvrdom). (APUSO)
2. Gost potvrđuje registraciju. (APSO)
3. Sistem validira podatke i kreira nalog. (SO)
4. Sistem vraća poruku o uspehu i opciono automatski prijavljuje korisnika. (IA)  
   **Alternativni scenariji:**  
   3.1 Ako email već postoji ili je forma neispravna, sistem prikazuje grešku. (IA)

#### SK2: Slučaj korišćenja – Prijava korisnika

**Naziv SK:** Prijava korisnika  
**Aktori SK:** Gost (neulogovani korisnik)  
**Učesnici SK:** Gost i sistem  
**Preduslov:** Gost je otvorio stranicu za prijavu.  
**Osnovni scenario SK:**

1. Gost unosi email i lozinku. (APUSO)
2. Gost pokreće prijavu. (APSO)
3. Sistem proverava akreditive. (SO)
4. Sistem čuva token i ulogu u localStorage. (SO)
5. Sistem preusmerava na početnu/katalog. (IA)  
   **Alternativni scenariji:**  
   3.1 Ako su podaci neispravni, sistem prikazuje poruku o grešci. (IA)

#### SK3: Slučaj korišćenja – Odjava korisnika

**Naziv SK:** Odjava korisnika  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik je prijavljen.  
**Osnovni scenario SK:**

1. Korisnik klikne na „Odjavi se“. (APSO)
2. Sistem poništava sesiju/poziva API za odjavu. (SO)
3. Sistem briše token i korisnika iz localStorage i preusmeri na login/početnu. (IA)

#### SK4: Slučaj korišćenja – Reset lozinke

**Naziv SK:** Reset lozinke  
**Aktori SK:** Gost ili registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Otvorena stranica „Zaboravljena lozinka“.  
**Osnovni scenario SK:**

1. Korisnik unosi email i šalje zahtev za reset. (APUSO/APSO)
2. Sistem validira email i generiše token. (SO)
3. Sistem šalje email sa linkom ka front-end formi za promenu lozinke. (SO)
4. Korisnik otvara link, unosi novu lozinku i potvrdu. (APUSO/APSO)
5. Sistem menja lozinku i prijavljuje/obaveštava korisnika. (SO/IA)  
   **Alternativni scenariji:**  
   2.1 Ako je previše pokušaja, sistem javlja throttling. (IA)  
   3.1 Ako slanje emaila ne uspe, sistem vraća poruku o grešci. (IA)

#### SK5: Slučaj korišćenja – Pregled kataloga sala

**Naziv SK:** Pregled kataloga sala  
**Aktori SK:** Registrovani korisnik, Admin/Menadžer  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik je na stranici kataloga.  
**Osnovni scenario SK:**

1. Sistem učitava listu sala sa osnovnim podacima. (SO)
2. Sistem prikazuje tabelu/listu sa salama. (IA)

#### SK6: Slučaj korišćenja – Pretraga i filtriranje sala

**Naziv SK:** Pretraga/filtriranje sala  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Otvoren katalog sala.  
**Osnovni scenario SK:**

1. Korisnik unosi kriterijume (tip, kapacitet od/do, status, tekstualna pretraga). (APUSO)
2. Korisnik pokreće filtriranje ili se primenjuje „na unos“. (APSO)
3. Sistem primenjuje filtere i prikazuje rezultat. (SO/IA)

#### SK7: Slučaj korišćenja – Rezervacija sale

**Naziv SK:** Rezervacija sale  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik gleda katalog ili mapu; ima termin i salu.  
**Osnovni scenario SK:**

1. Korisnik otvara formu za rezervaciju sa izabranom salom/terminom. (APUSO)
2. Korisnik unosi/confirm-uje datum, vreme i tip događaja. (APUSO)
3. Korisnik potvrđuje rezervaciju. (APSO)
4. Sistem validira i kreira rezervaciju sa statusom (npr. pending). (SO)
5. Sistem prikazuje poruku o uspehu i osvežava listu. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako je termin zauzet, sistem prikazuje grešku/predlaže alternativu. (IA)

#### SK8: Slučaj korišćenja – Uređivanje rasporeda sala na mapi (admin)

**Naziv SK:** Uređivanje rasporeda na mapi  
**Aktori SK:** Admin  
**Učesnici SK:** Admin i sistem  
**Preduslov:** Admin je prijavljen i otvorio „Mapa sprata“.  
**Osnovni scenario SK:**

1. Admin uključuje „Uredi raspored“. (APSO)
2. Admin prevlači/menja poziciju i dimenzije prostorija. (APUSO)
3. Admin pušta element; sistem optimistički ažurira prikaz. (SO/IA)
4. Sistem čuva nove koordinate/dimenzije (PATCH /sale/{id}/layout). (SO)
5. Sistem potvrđuje snimljene izmene. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako čuvanje ne uspe, sistem vrši rollback ili osvežava podatke. (IA)

#### SK9: Slučaj korišćenja – Upravljanje salama (kreiranje, izmena, deaktivacija)

**Naziv SK:** Upravljanje salama  
**Aktori SK:** Admin ili Menadžer  
**Učesnici SK:** Admin/Menadžer i sistem  
**Preduslov:** Admin/Menadžer je na admin stranici za sale.  
**Osnovni scenario SK:**

1. Aktor otvara formu „Nova sala“ ili „Izmena“. (APSO)
2. Unosi/menja naziv, tip, kapacitet, status, opis i raspored meta-podatke. (APUSO)
3. Potvrđuje snimanje. (APSO)
4. Sistem validira i čuva izmene. (SO)
5. Sistem osvežava tabelu sala. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako validacija padne, sistem prikazuje poruke o greškama. (IA)

#### SK10: Slučaj korišćenja – Pregled svih rezervacija i odobravanje/odbijanje

**Naziv SK:** Administracija rezervacija  
**Aktori SK:** Admin/Menadžer  
**Učesnici SK:** Admin/Menadžer i sistem  
**Preduslov:** Aktor je na admin stranici „Rezervacije“.  
**Osnovni scenario SK:**

1. Sistem učitava sve rezervacije sa statusima. (SO)
2. Sistem prikazuje tabelu sa filtrima/sortom. (IA)
3. Aktor odabira rezervaciju i menja status u „odobreno“ ili „odbijeno“. (APSO)
4. Sistem ažurira status i vrati potvrdu. (SO/IA)  
   **Alternativni scenariji:**  
   4.1 Ako ažuriranje ne uspe, sistem prikazuje poruku o grešci. (IA)

## Faza analize

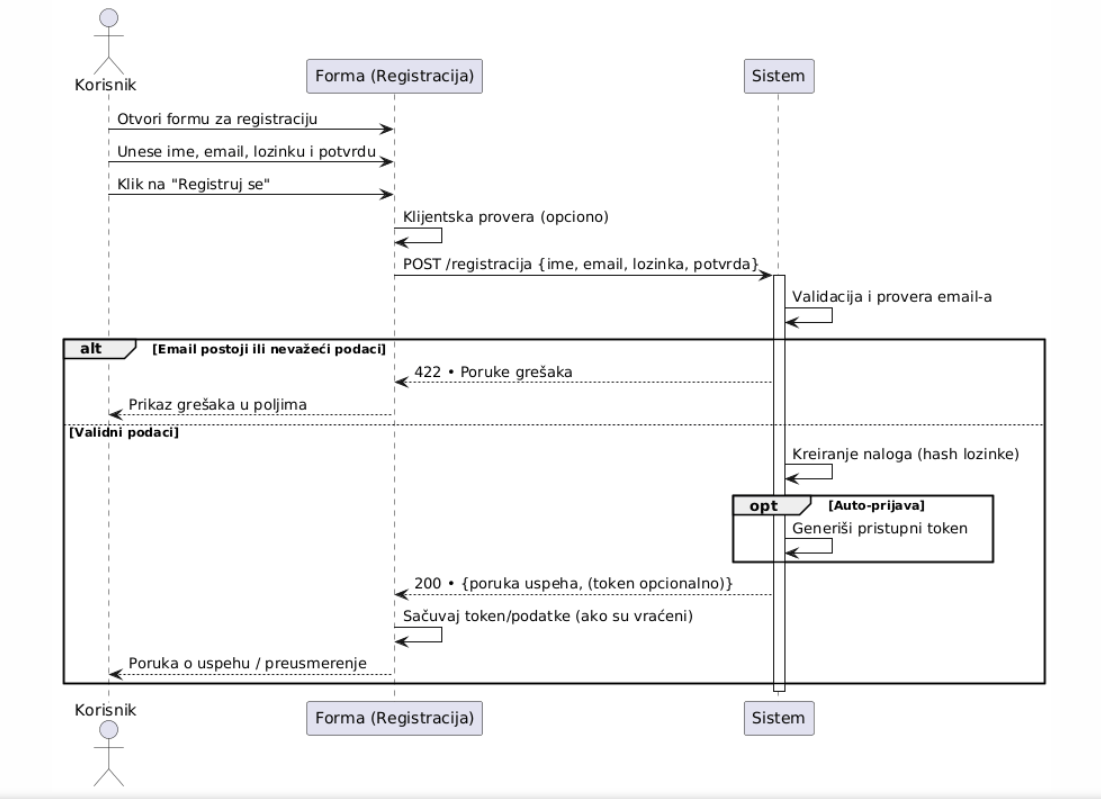
Dijagrami sekvenci prikazuju kako objekti/sistemske komponente međusobno komuniciraju tokom vremena za jedan konkretan scenario. Na vertikalnim “linijama života” stoje učesnici (npr. korisnik, UI, kontroler, servis, baza), a horizontalne strelice predstavljaju poruke/pozive metoda koje teku hronološki od vrha ka dnu. Time se jasno vidi redosled koraka, ko koga poziva, koje podatke šalje i kada se vraćaju odgovori ili nastaju izuzeci. Koriste se da razjasne ponašanje sistema, usaglase očekivanja među timovima (analiza, backend, frontend), otkriju zavisnosti i potencijalne race uslove, i kao osnova za implementaciju i test scenarije.

### Sistemski dijagrami sekvence

#### DS1: Dijagram sekvenci – Registracija korisnika

**Naziv SK:** Registracija korisnika  
**Aktori SK:** Gost (neulogovani korisnik)  
**Učesnici SK:** Gost i sistem  
**Preduslov:** Gost je otvorio stranicu za registraciju.  
**Osnovni scenario SK:**

1. Gost unosi ime, email i lozinku (sa potvrdom). (APUSO)
2. Gost potvrđuje registraciju. (APSO)
3. Sistem validira podatke i kreira nalog. (SO)
4. Sistem vraća poruku o uspehu i opciono automatski prijavljuje korisnika. (IA)  
   **Alternativni scenariji:**  
   3.1 Ako email već postoji ili je forma neispravna, sistem prikazuje grešku. (IA)

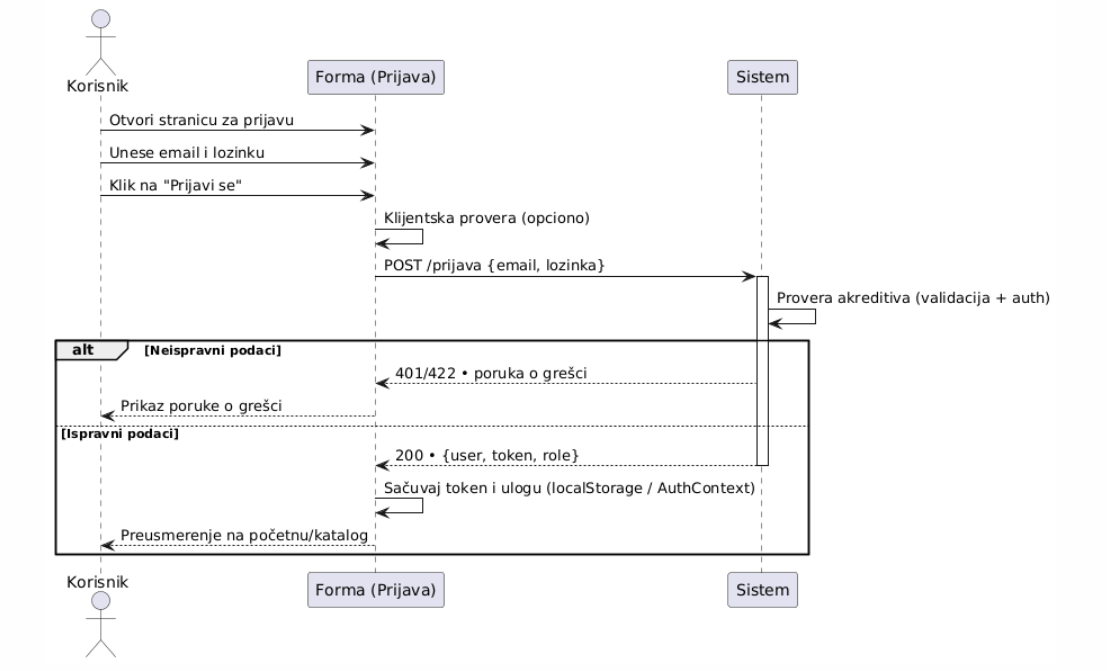


Slika 1 ds registracija

#### DS2: Dijagram sekvenci – Prijava korisnika

**Naziv SK:** Prijava korisnika  
**Aktori SK:** Gost (neulogovani korisnik)  
**Učesnici SK:** Gost i sistem  
**Preduslov:** Gost je otvorio stranicu za prijavu.  
**Osnovni scenario SK:**

1. Gost unosi email i lozinku. (APUSO)
2. Gost pokreće prijavu. (APSO)
3. Sistem proverava akreditive. (SO)
4. Sistem čuva token i ulogu u localStorage. (SO)
5. Sistem preusmerava na početnu/katalog. (IA)  
   **Alternativni scenariji:**  
   3.1 Ako su podaci neispravni, sistem prikazuje poruku o grešci. (IA)

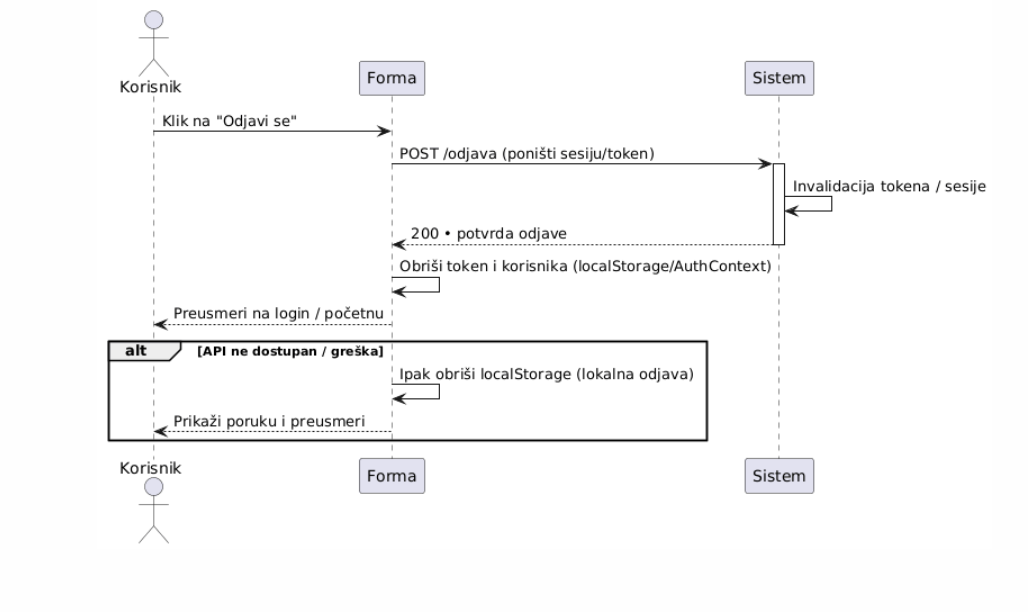


Slika 2 ds prijava

#### DS3: Dijagram sekvenci – Odjava korisnika

**Naziv SK:** Odjava korisnika  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik je prijavljen.  
**Osnovni scenario SK:**

1. Korisnik klikne na „Odjavi se“. (APSO)
2. Sistem poništava sesiju/poziva API za odjavu. (SO)
3. Sistem briše token i korisnika iz localStorage i preusmeri na login/početnu. (IA)

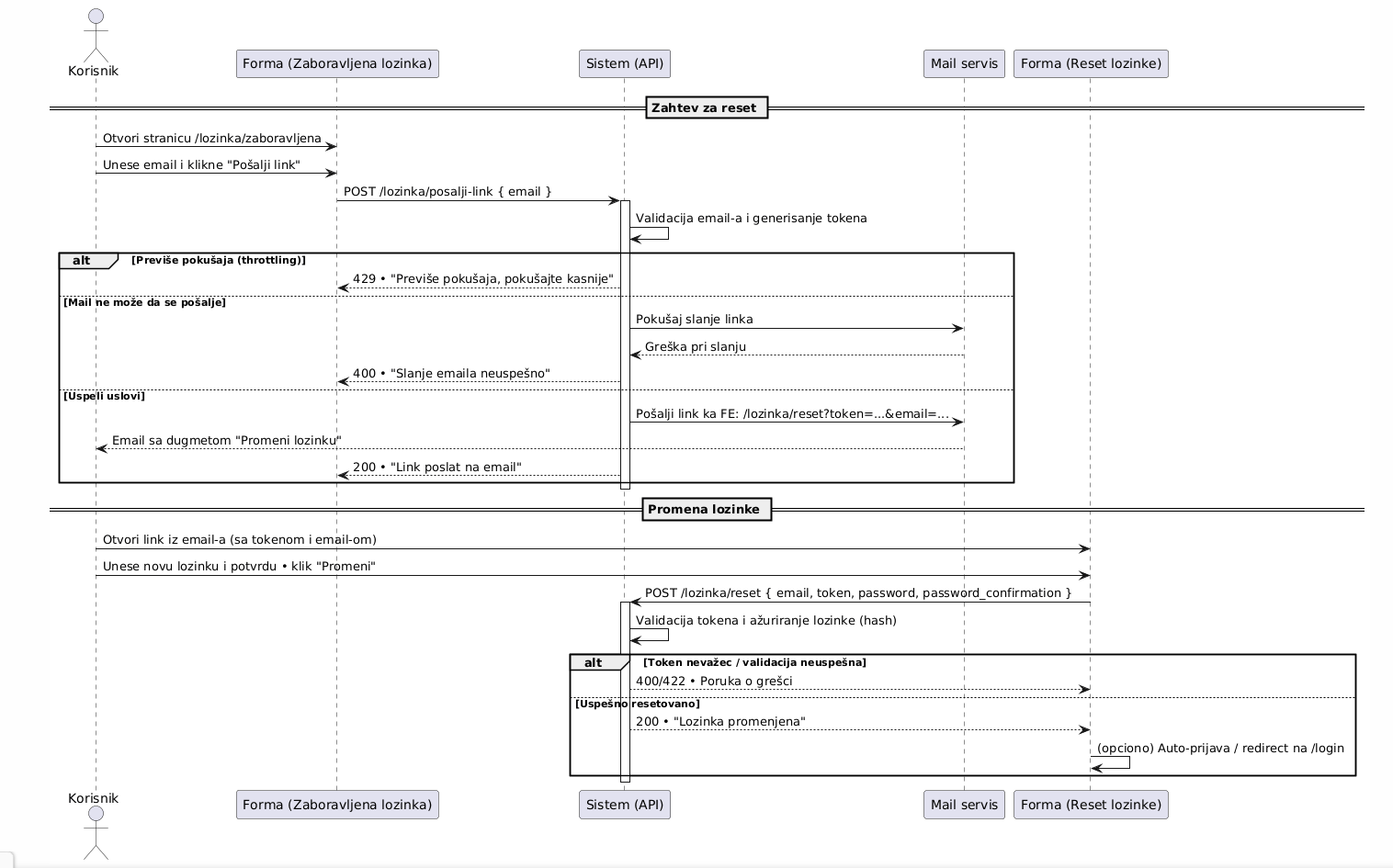


Slika 3 ds odjava korisnika

#### DS4: Dijagram sekvenci – Reset lozinke

**Naziv SK:** Reset lozinke  
**Aktori SK:** Gost ili registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Otvorena stranica „Zaboravljena lozinka“.  
**Osnovni scenario SK:**

1. Korisnik unosi email i šalje zahtev za reset. (APUSO/APSO)
2. Sistem validira email i generiše token. (SO)
3. Sistem šalje email sa linkom ka front-end formi za promenu lozinke. (SO)
4. Korisnik otvara link, unosi novu lozinku i potvrdu. (APUSO/APSO)
5. Sistem menja lozinku i prijavljuje/obaveštava korisnika. (SO/IA)  
   **Alternativni scenariji:**  
   2.1 Ako je previše pokušaja, sistem javlja throttling. (IA)  
   3.1 Ako slanje emaila ne uspe, sistem vraća poruku o grešci. (IA)

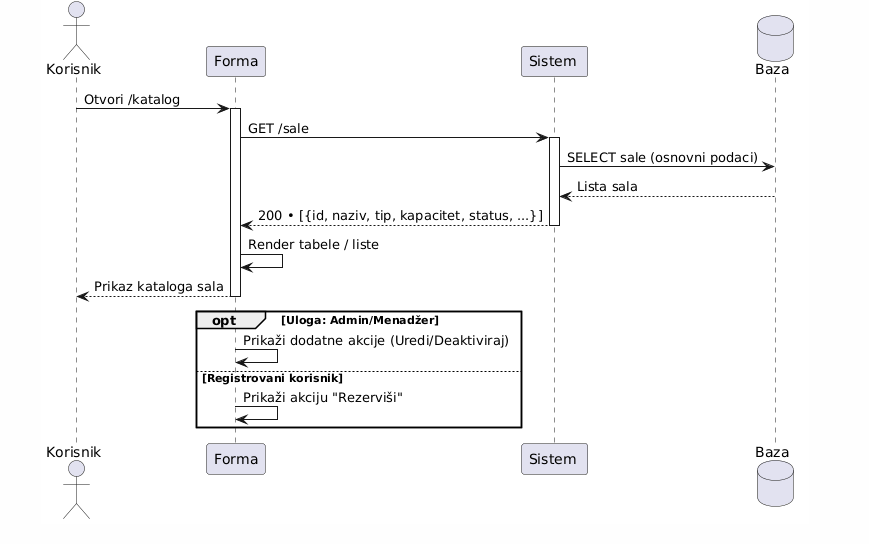


Slika 4 ds reset lozinke

#### DS5: Dijagram sekvenci – Pregled kataloga sala

**Naziv SK:** Pregled kataloga sala  
**Aktori SK:** Registrovani korisnik, Admin/Menadžer  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik je na stranici kataloga.  
**Osnovni scenario SK:**

1. Sistem učitava listu sala sa osnovnim podacima. (SO)
2. Sistem prikazuje tabelu/listu sa salama. (IA)

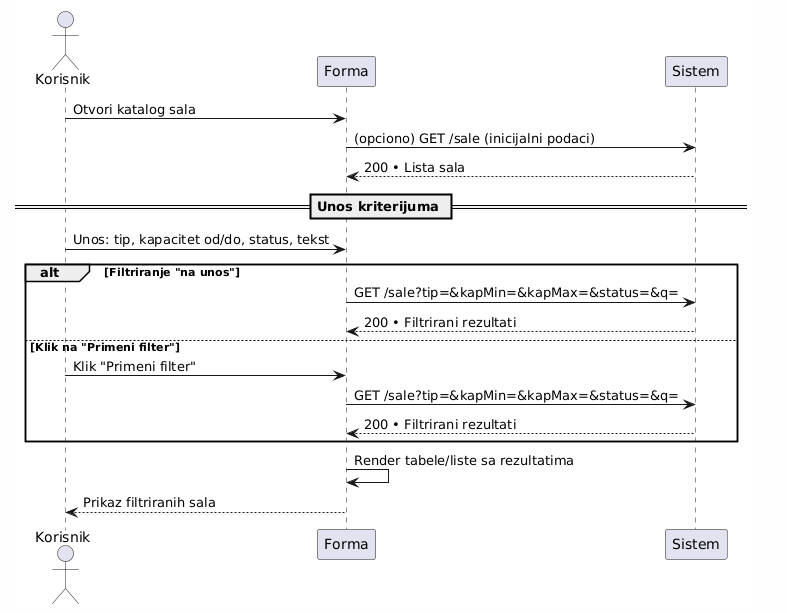


Slika 5 ds katalog

#### DS6: Dijagram sekvenci – Pretraga i filtriranje sala

**Naziv SK:** Pretraga/filtriranje sala  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Otvoren katalog sala.  
**Osnovni scenario SK:**

1. Korisnik unosi kriterijume (tip, kapacitet od/do, status, tekstualna pretraga). (APUSO)
2. Korisnik pokreće filtriranje ili se primenjuje „na unos“. (APSO)
3. Sistem primenjuje filtere i prikazuje rezultat. (SO/IA)



Slika 6 ds pretraga sala

#### DS7: Dijagram sekvenci – Rezervacija sale

**Naziv SK:** Rezervacija sale  
**Aktori SK:** Registrovani korisnik  
**Učesnici SK:** Korisnik i sistem  
**Preduslov:** Korisnik gleda katalog ili mapu; ima termin i salu.  
**Osnovni scenario SK:**

1. Korisnik otvara formu za rezervaciju sa izabranom salom/terminom. (APUSO)
2. Korisnik unosi/confirm-uje datum, vreme i tip događaja. (APUSO)
3. Korisnik potvrđuje rezervaciju. (APSO)
4. Sistem validira i kreira rezervaciju sa statusom (npr. pending). (SO)
5. Sistem prikazuje poruku o uspehu i osvežava listu. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako je termin zauzet, sistem prikazuje grešku/predlaže alternativu. (IA)

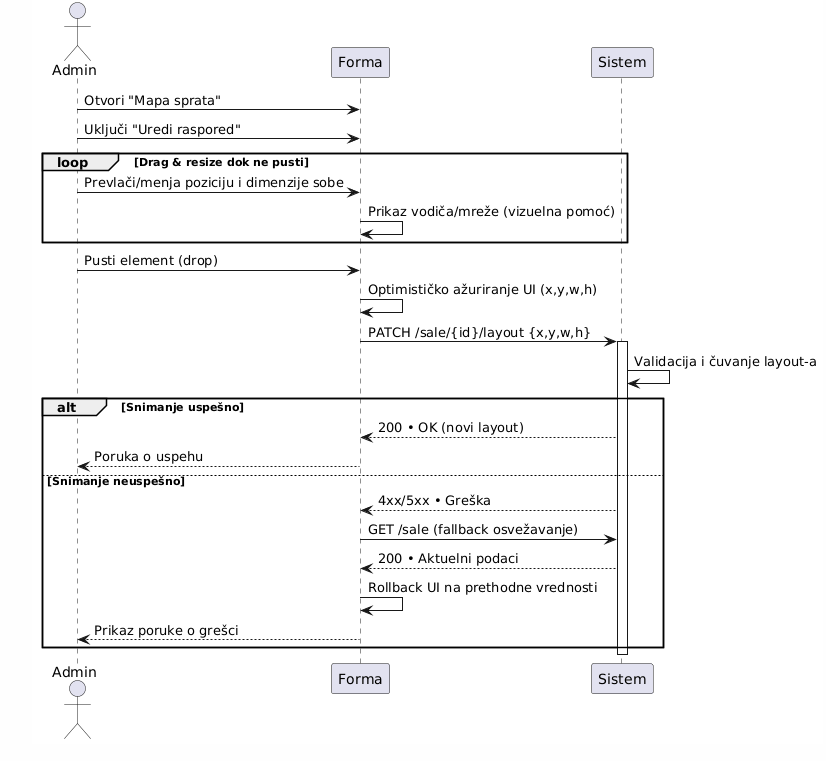


Slika 7 ds rezervacija sala

#### DS8: Dijagram sekvenci – Uređivanje rasporeda sala na mapi (admin)

**Naziv SK:** Uređivanje rasporeda na mapi  
**Aktori SK:** Admin  
**Učesnici SK:** Admin i sistem  
**Preduslov:** Admin je prijavljen i otvorio „Mapa sprata“.  
**Osnovni scenario SK:**

1. Admin uključuje „Uredi raspored“. (APSO)
2. Admin prevlači/menja poziciju i dimenzije prostorija. (APUSO)
3. Admin pušta element; sistem optimistički ažurira prikaz. (SO/IA)
4. Sistem čuva nove koordinate/dimenzije (PATCH /sale/{id}/layout). (SO)
5. Sistem potvrđuje snimljene izmene. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako čuvanje ne uspe, sistem vrši rollback ili osvežava podatke. (IA)

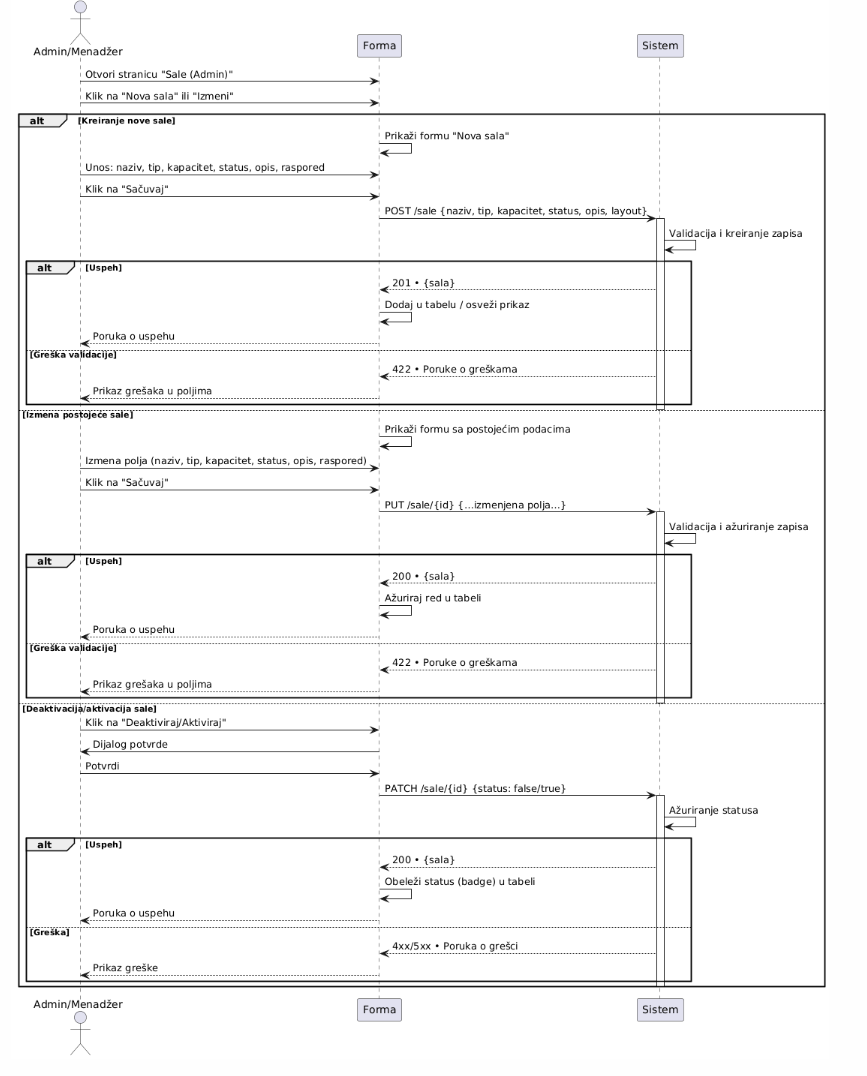


Slika 8 ds raspored sala

#### DS9: Dijagram sekvenci – Upravljanje salama (kreiranje, izmena, deaktivacija)

**Naziv SK:** Upravljanje salama  
**Aktori SK:** Admin ili Menadžer  
**Učesnici SK:** Admin/Menadžer i sistem  
**Preduslov:** Admin/Menadžer je na admin stranici za sale.  
**Osnovni scenario SK:**

1. Aktor otvara formu „Nova sala“ ili „Izmena“. (APSO)
2. Unosi/menja naziv, tip, kapacitet, status, opis i raspored meta-podatke. (APUSO)
3. Potvrđuje snimanje. (APSO)
4. Sistem validira i čuva izmene. (SO)
5. Sistem osvežava tabelu sala. (IA)  
   **Alternativni scenariji:**  
   4.1 Ako validacija padne, sistem prikazuje poruke o greškama. (IA)



Slika 9 ds upravljanje salama

#### DS10: Dijagram sekvenci – Pregled svih rezervacija i odobravanje/odbijanje

**Naziv SK:** Administracija rezervacija  
**Aktori SK:** Admin/Menadžer  
**Učesnici SK:** Admin/Menadžer i sistem  
**Preduslov:** Aktor je na admin stranici „Rezervacije“.  
**Osnovni scenario SK:**

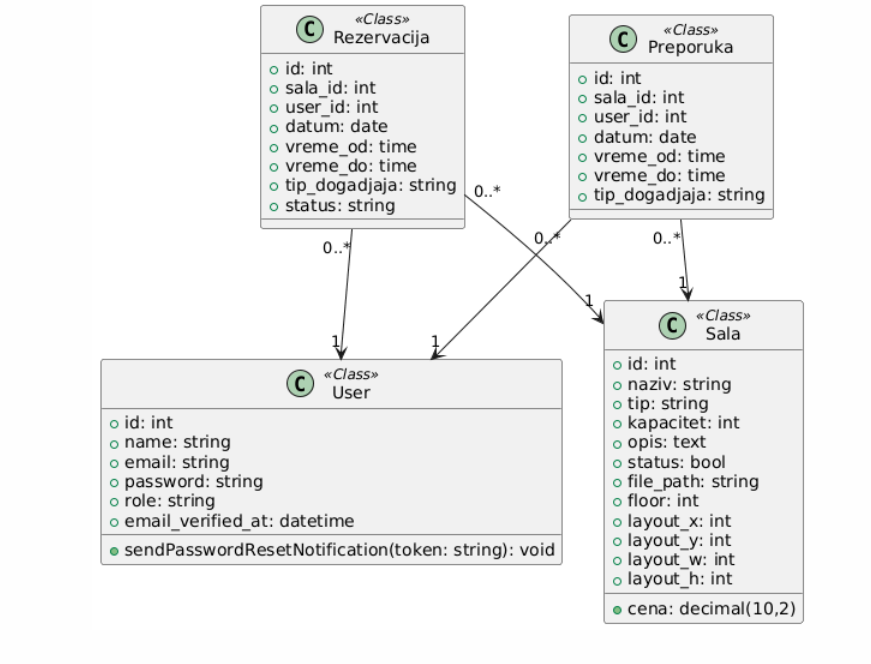
1. Sistem učitava sve rezervacije sa statusima. (SO)
2. Sistem prikazuje tabelu sa filtrima/sortom. (IA)
3. Aktor odabira rezervaciju i menja status u „odobreno“ ili „odbijeno“. (APSO)
4. Sistem ažurira status i vrati potvrdu. (SO/IA)  
   **Alternativni scenariji:**  
   4.1 Ako ažuriranje ne uspe, sistem prikazuje poruku o grešci. (IA)



Slika 10 pregled rezervacija

### Struktura softverskog sistema

U okviru ove faze analize izrađen je UML dijagram klasa koji prikazuje glavne entitete sistema i odnose između njih. Ovaj dijagram predstavlja statički prikaz strukture softverskog sistema i omogućava bolju vizualizaciju modela koji su korišćeni prilikom implementacije aplikacije.



Slika 11 UML dijagram klasa

## Faza projektovanja

Tokom projektovanja utvrđuju se arhitektura, izbor tehnologija i način organizacije sistema, kako bi razvoj tekao brzo, modularno i uz jednostavno održavanje. Naš sistem je postavljen kao klijent–server: korisnički deo razvijen je u React-u, dok serversku logiku obavlja Laravel. Dva sloja komuniciraju preko REST API-ja i razmenjuju podatke u JSON formatu, a trajno čuvanje informacija realizovano je u MySQL bazi.

Na frontendu interfejs je građen iz manjih, višekratno upotrebljivih React komponenti, dok backend koristi Laravel resursne kontrolere za obradu HTTP zahteva i generisanje odgovora. API rute su oblikovane po REST principima i dodatno osigurane middleware-ima, pre svega mehanizmima za proveru identiteta. Prijavljivanje i autorizacija korisnika rešeni su JSON Web Token (JWT) tokenima, što omogućava bezbedno upravljanje sesijama u pregledaču.

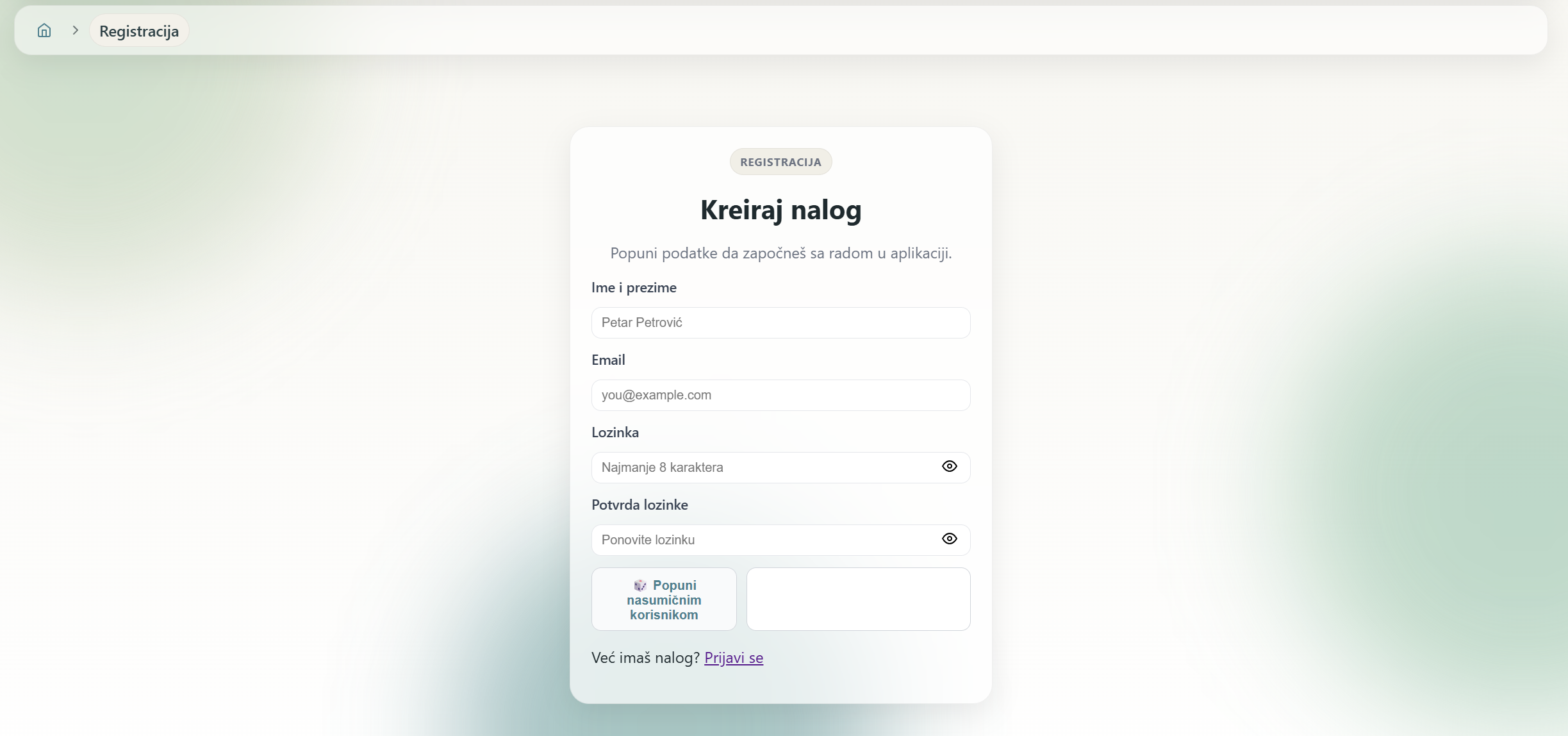
### Projektovanje korisničkog interfejsa

Prilikom pokretanja aplikacije otvara se početna stranica. Ona izgleda kao na slici ispod.



Slika 12 početna stranica

Kako bi korisnici mogli da koriste ovu aplikaciju, potrebno je da se ulogouju, a ukoliko nemaju nalog moraju ga prvo kreirati.

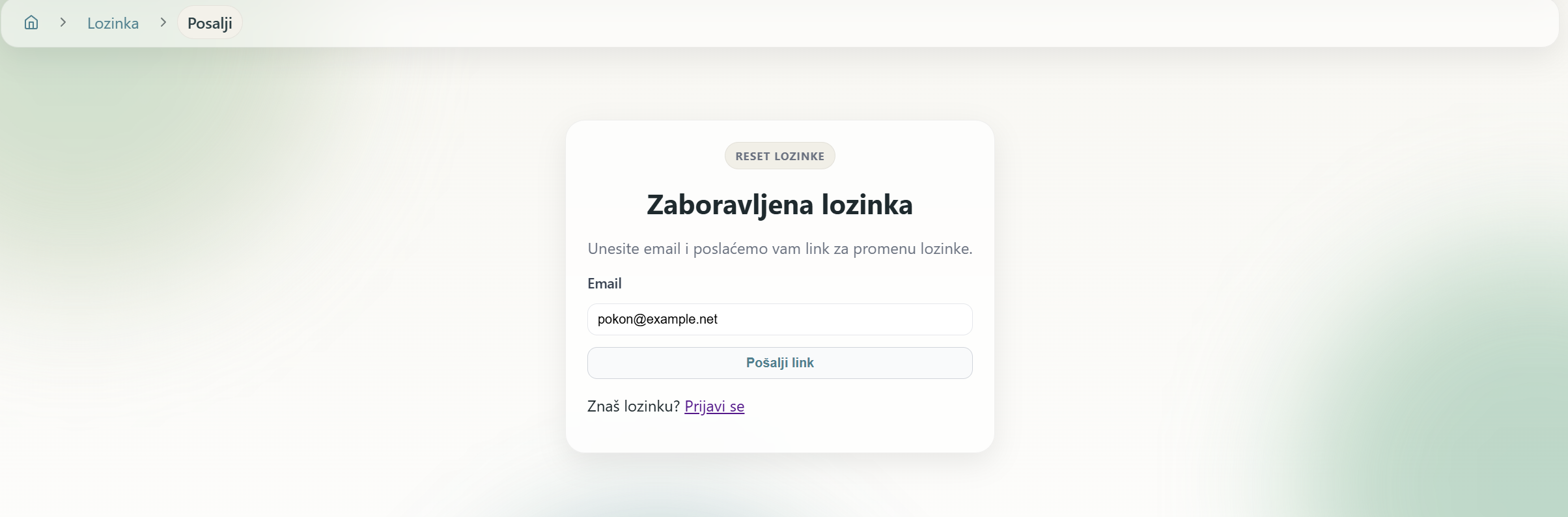


Slika 13 registracija korisnika



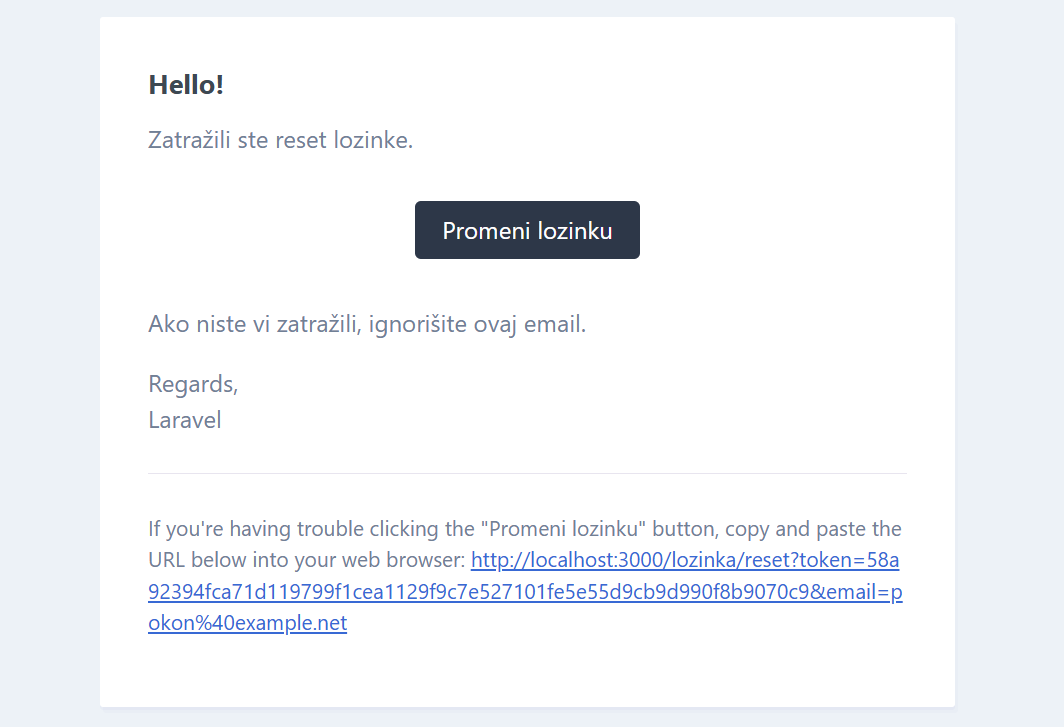
Slika 14 prijava korisnika

Ukoliko su korisnici zaboravili lozinku, mogu je resetovati odabirom odgovarajuće opcije sa login stranice.



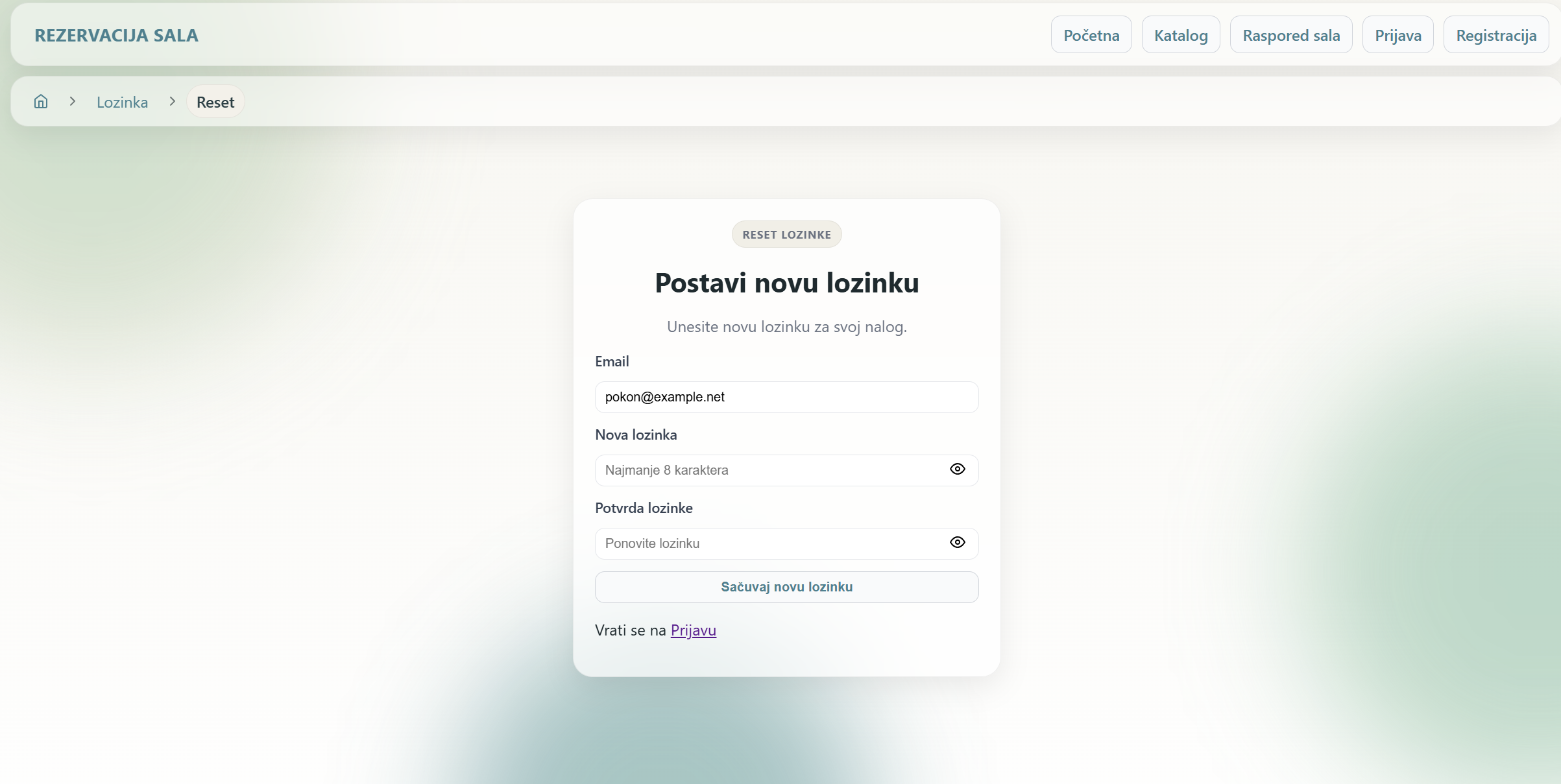
Slika 15 reset lozinke

Nakon unosa mejla, stiže link za formu za reset lozinke.



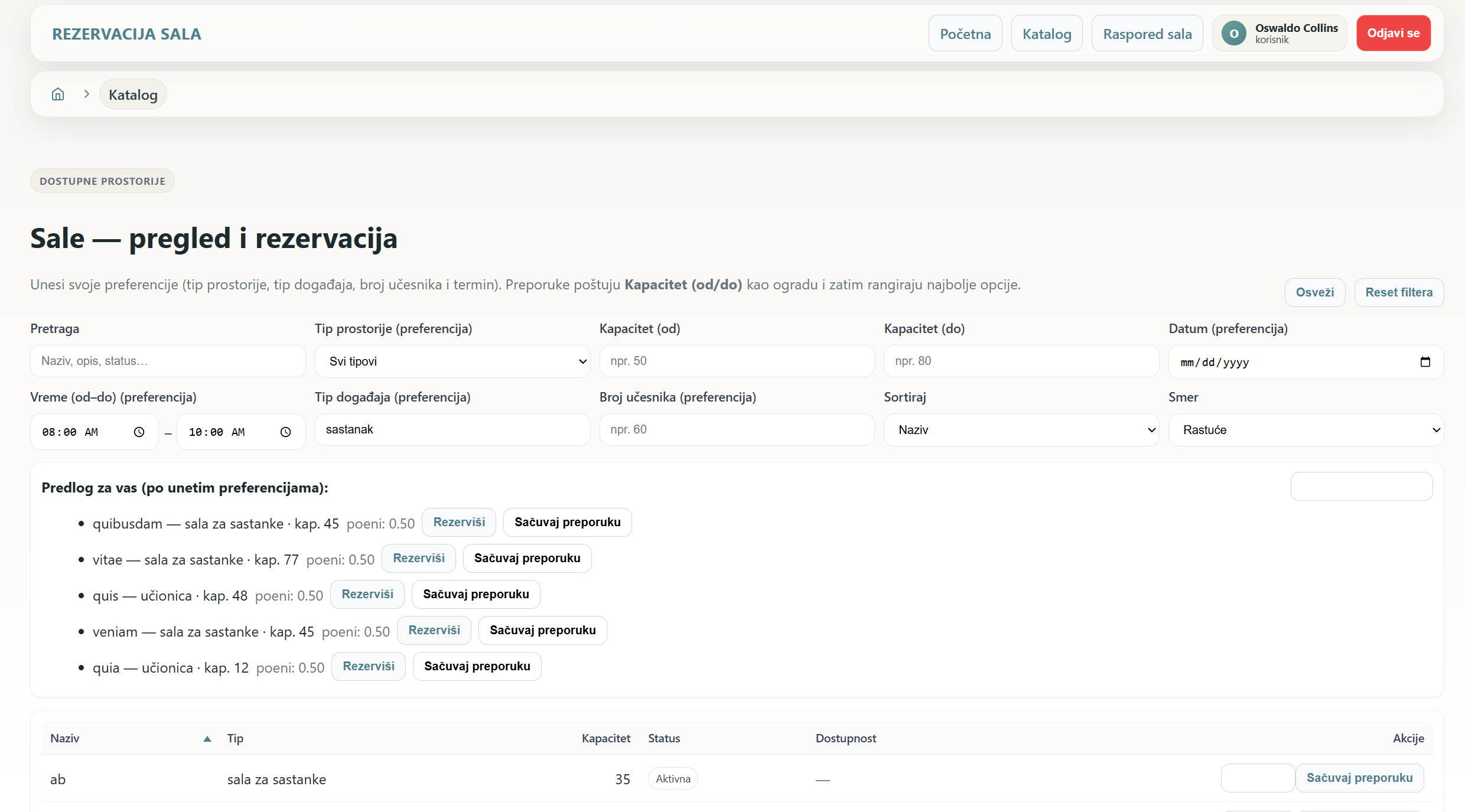
Slika 16 mejl za promenu lozinke

Klikom na dugme iz mejla otvara se forma za unos nove lozinke



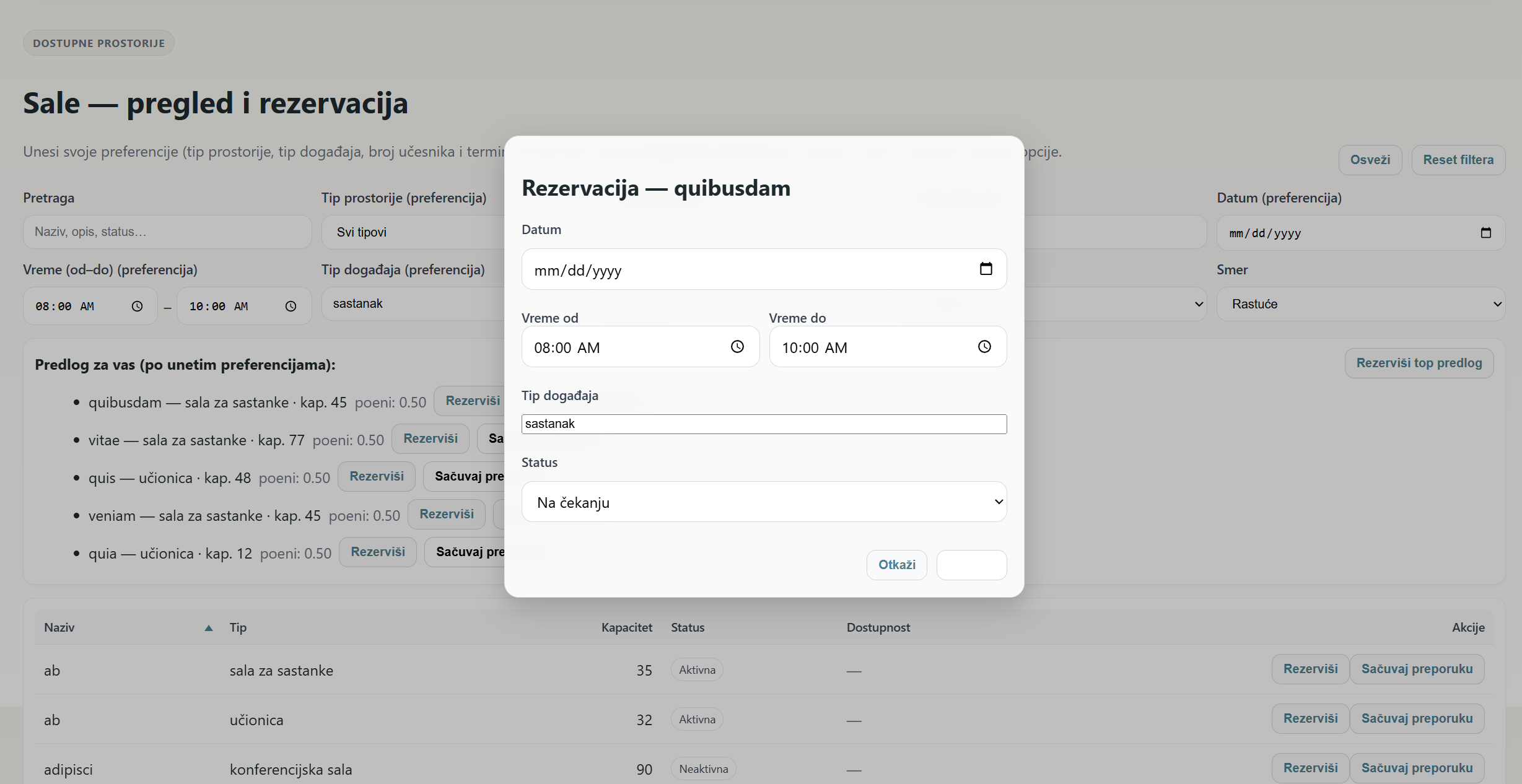
Slika 17 reset lozinke

Kada se u aplikaciju uloguje korisnik otvara mu se stranica za prikaz svih sala, napredni filter kao I sistem preporuke. Radi lakšeg snalaženja ovde imamo I paginaciju I filtere.



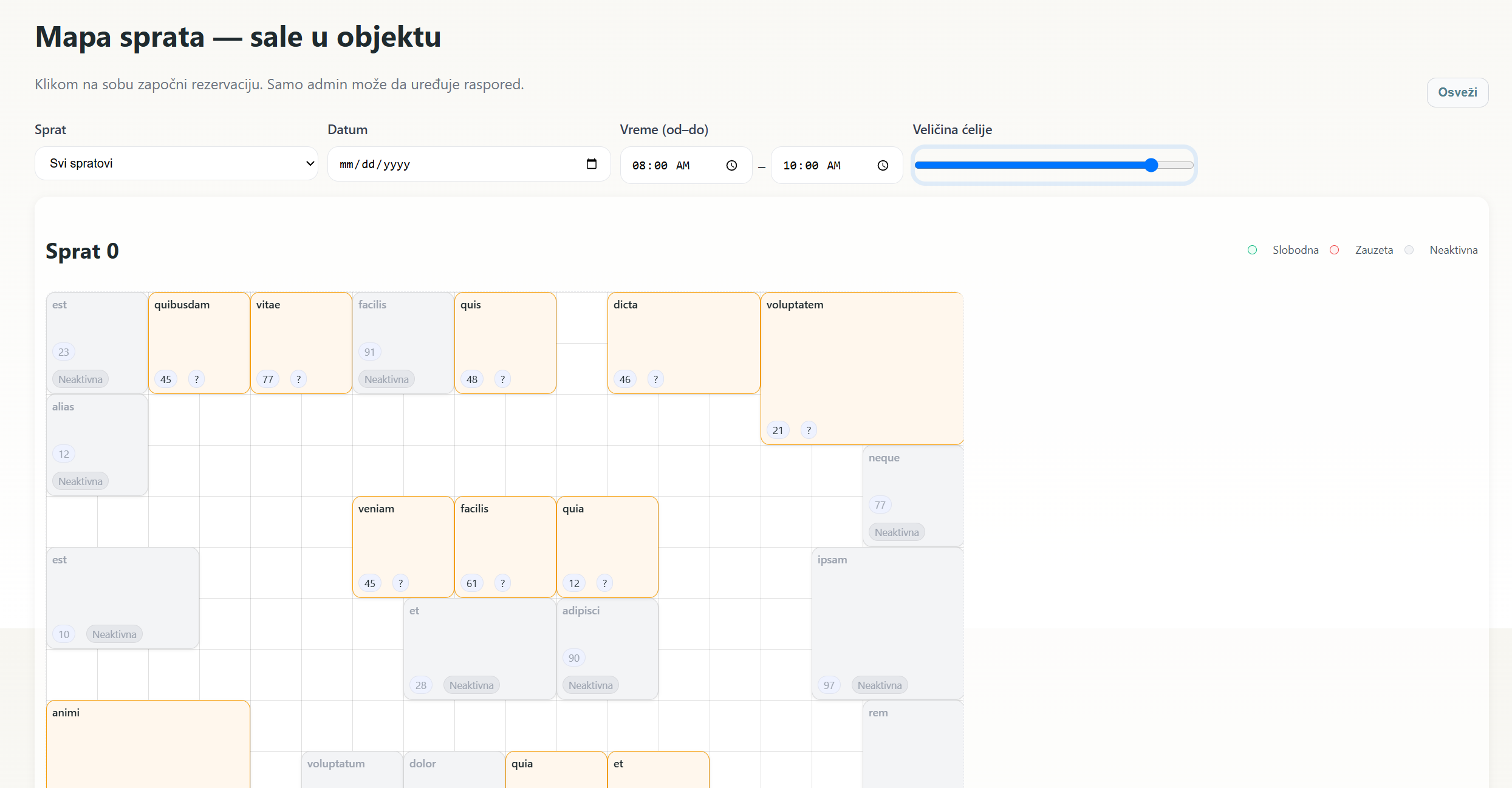
Slika 18 pregled sala

Na ovoj stranici je moguće dodati rezervaciju.



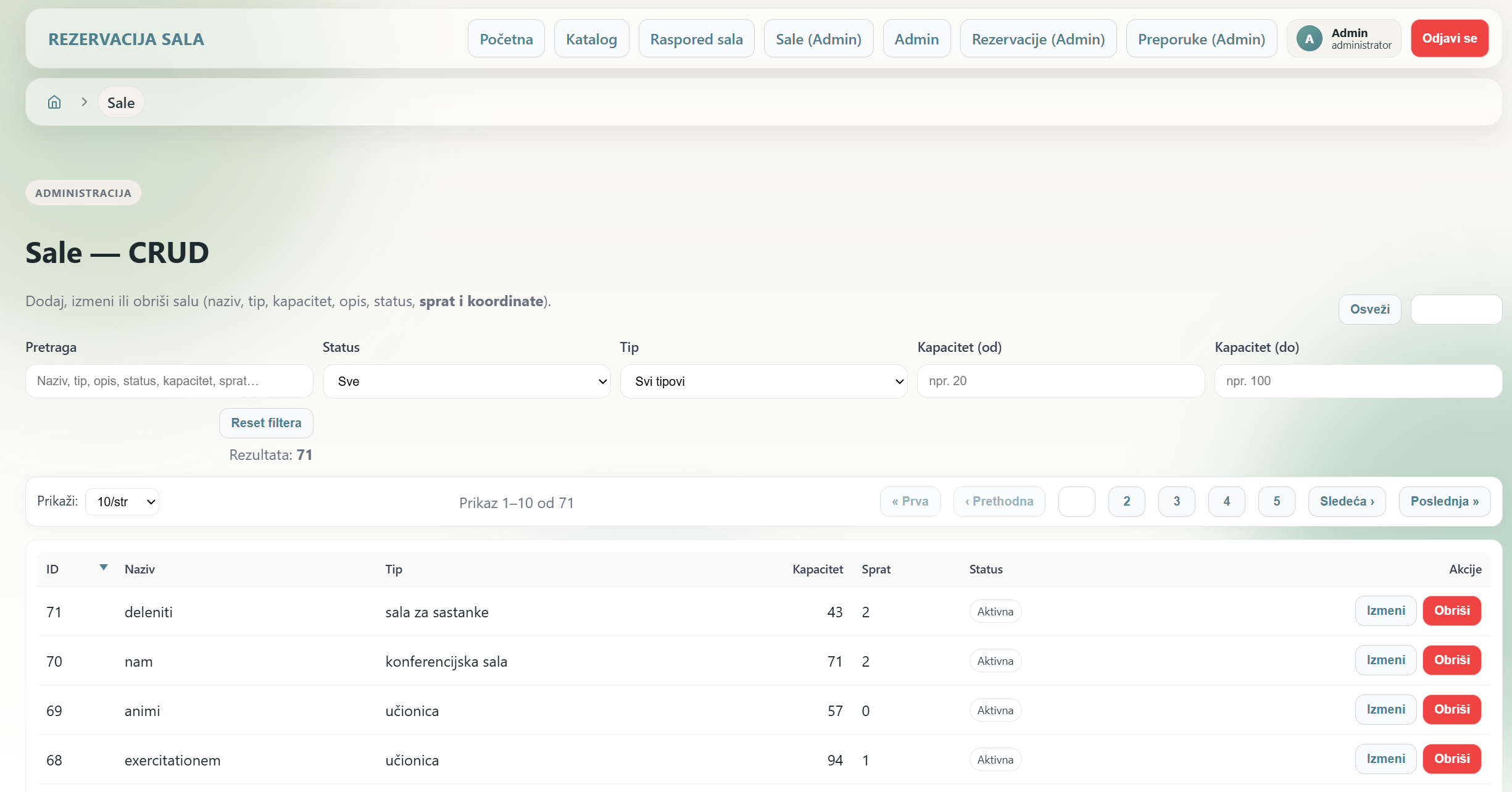
Slika 19 kreiranje rezervacije

Ulogovani korisnici mogu da gledaju prostorni raspored sala po spratovima.



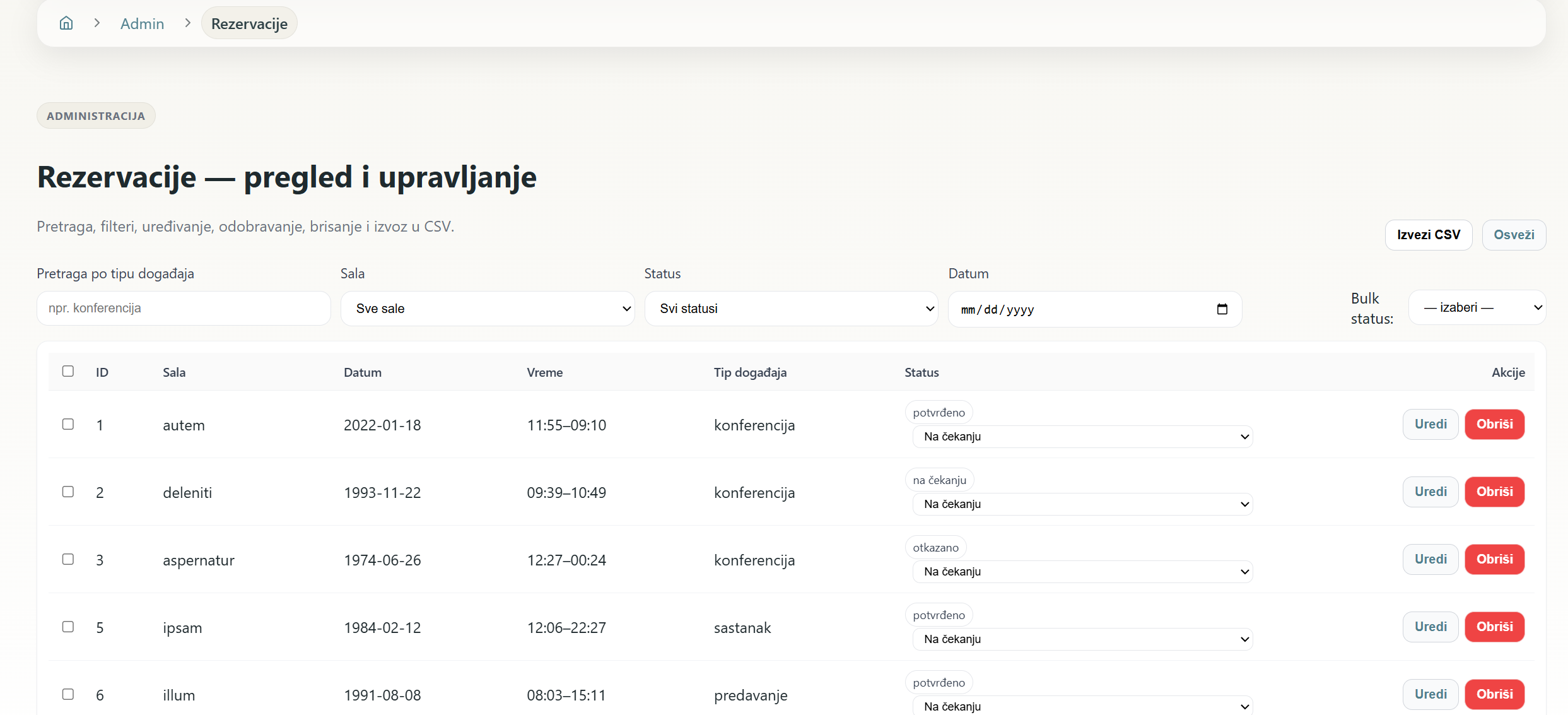
Slika 20 mapa sala

Ukoliko je u aplikaciju ulogovan admin, on na istoj stranici može menjati raspored sala. Osim toga, admin ima stranicu za upravljanje salama.

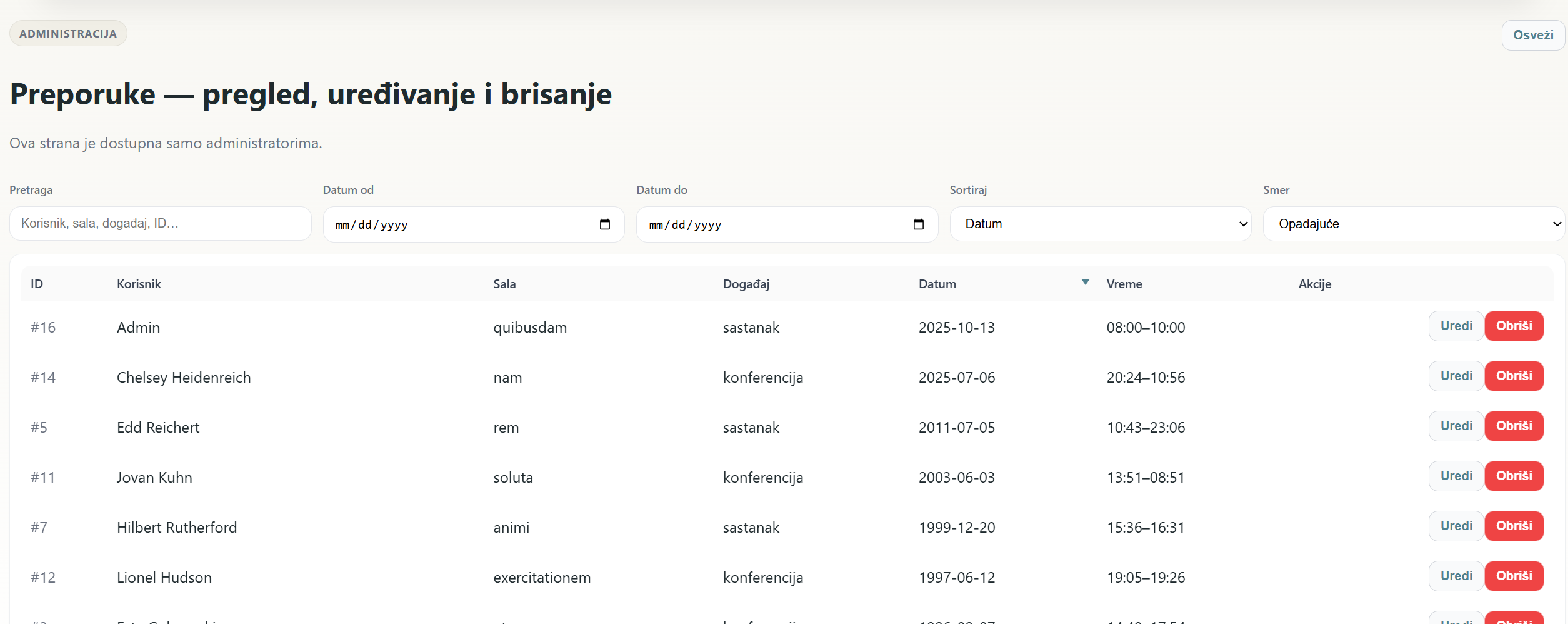


Slika 21 upravljanje salama

Takodje, admin ima mogućnost pregleda upravljanaj rezervacijama I preporukama.

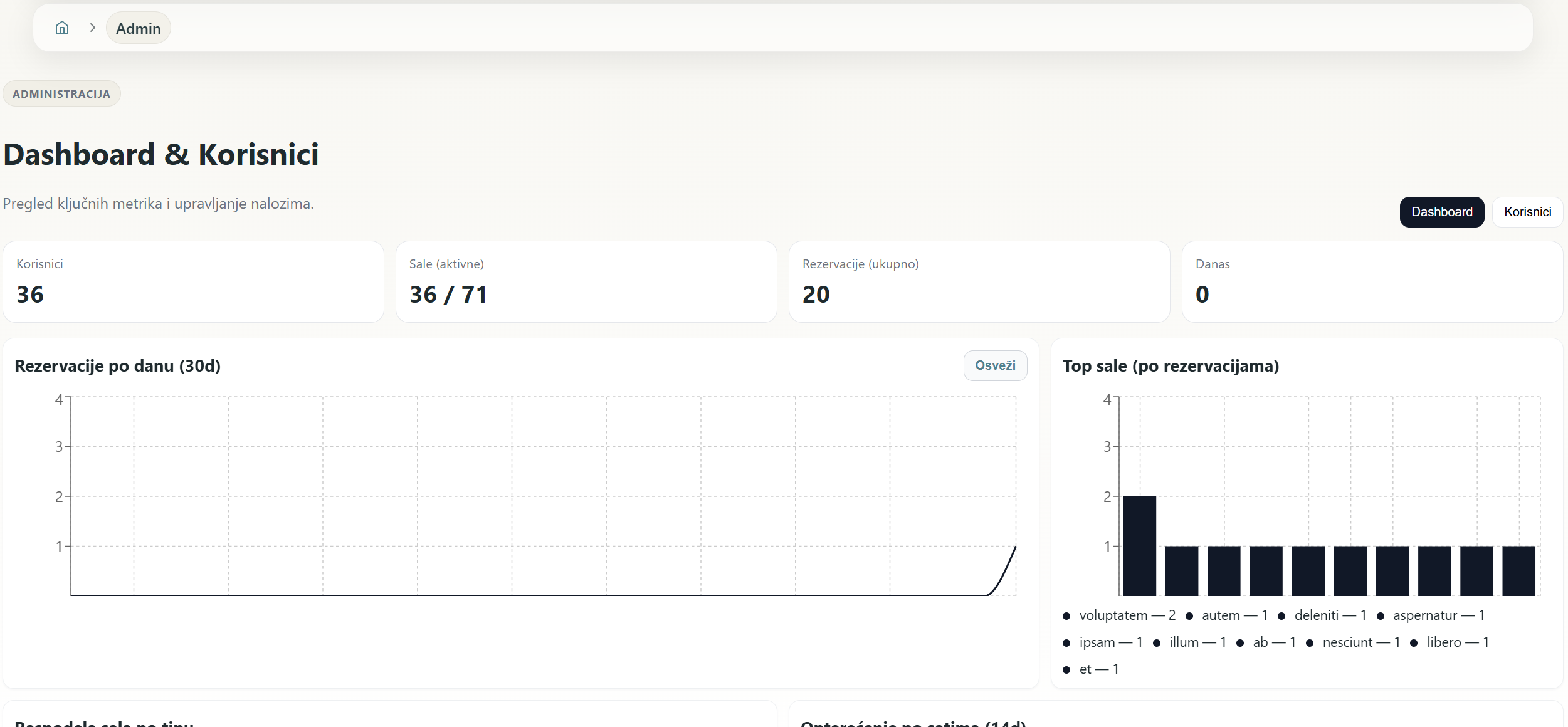


Slika 22 upravljanje rezervacijama

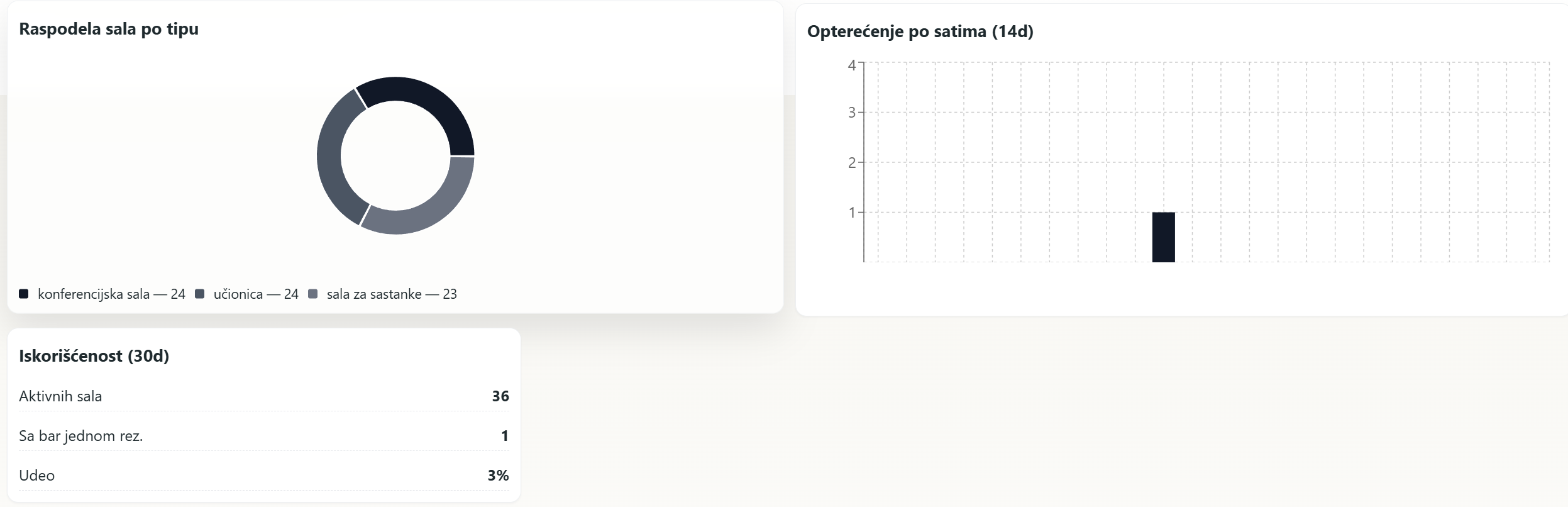


Slika 23 upravljanje preporukama

Na kraju, admin ima pregled statistika.



Slika 24 admin statistike 1



Slika 25 admin statistike 2

### Projektovanje aplikacione logike

Projektovanje aplikacione logike predstavlja jedan od ključnih koraka u izradi softverskog sistema, jer se u ovoj fazi definiše na koji način aplikacija obrađuje podatke, izvršava poslovna pravila i komunicira između različitih slojeva sistema. Aplikaciona logika se nalazi između korisničkog interfejsa i baze podataka, i odgovorna je za upravljanje tokovima podataka, validaciju unosa, izvršavanje operacija nad bazom i pripremu odgovora koji se šalju klijentskoj strani.

U okviru ovog poglavlja biće objašnjeni osnovni pojmovi vezani za aplikacionu logiku, kao što su kontroleri, servisi, modeli i rute. Takođe će biti priloženi primeri koda koji ilustruju kako su određene funkcionalnosti implementirane u okviru sistema – npr. kreiranje zahteva, obrada kupovine, autentifikacija korisnika i rad sa bazom podataka. Logika je implementirana na backend strani korišćenjem **Laravel framework-a**, koji koristi princip MVC (Model-View-Controller) arhitekture, čime je postignuto jasno razdvajanje odgovornosti i olakšano održavanje aplikacije.

#### Laravel Kontroleri

Laravel kontroleri su klase koje predstavljaju “posrednike” između HTTP zahteva i logike aplikacije. Ruta prosleđuje zahtev odgovarajućoj metodi kontrolera, a kontroler tu validira ulaz, poziva servise ili Eloquent modele, obrađuje poslovna pravila i vraća odgovor (JSON, view, redirect…). Na taj način se logika odvaja od definicije ruta i šablona, kod postaje organizovaniji i lakši za testiranje. Laravel nudi i „resource“ kontrolere koji prate REST konvencije (index, show, store, update, destroy), kao i middleware podršku na nivou kontrolera/metoda, pa je jednostavno centralizovati autentikaciju, autorizaciju i druge provere. U ovoj aplikaciji su kreirani sledeci kontroleri: AdminController, SaleController,PasswordController, RezervacijeController, PreporukaController.

##### SalaController

Ovaj **SalaController** je REST kontroler koji upravlja salama (prostorijama) kroz tipične CRUD operacije i nekoliko praktičnih dodataka. Svi odgovori su JSON i prolaze kroz **SalaResource**, tako da klijent uvek dobija konzistentno formatirane podatke.

Metoda **index()** čita sve sale iz baze, ali rezultat kešira pod ključem sale\_all na 60 minuta. Sledeći pozivi čitaju iz keša umesto baze, što ubrzava listanje. Kad god dođe do izmene (store/update/destroy/upload/updateLayout), kontroler briše taj keš (cache()->forget('sale\_all')) da bi se sledeći put učitao svež sadržaj. **show($id)** dohvaća jednu salu po ID-ju.

Metode **store(Request $request)** i **update(Request $request, $id)** rade istu stvar nad novom ili postojećom salom: prvo striktna **validacija** (naziv, tip, kapacitet, status, kao i obavezni meta-podaci za mapu: floor, layout\_x/y/w/h), zatim **sanitizacija** vrednosti (npr. osigurava da su dimenzije i koordinate nenegativne, kapacitet ≥ 1, format cene u decimalu). Nakon uspeha, model se kreira ili ažurira i vraća se kroz resource. Oba endpointa time garantuju ispravne i dosledne podatke koji se koriste i na vizuelnoj mapi sprata.

**destroy($id)** briše salu i vraća poruku o uspehu. Nakon brisanja se čisti keš liste da bi klijent video ažurirano stanje.

Metoda **uploadFile(Request $request, $id)** rešava **upload pratećeg fajla** za konkretnu salu. Prihvata samo određene tipove (jpg,png,pdf,doc,docx,zip) do 5 MB, snima fajl u storage/app/public/sala\_files (disk public), upisuje relativnu putanju u polje file\_path i vraća javni URL (asset('storage/...')) zajedno sa svežim stanjem sale. I ovde se keš liste sala resetuje.

Na kraju, **updateLayout(Request $request, Sala $sala)** je ciljano **PATCH** ažuriranje rasporeda na mapi. Zahteva samo polja koja menja (sometimes|integer|min pravila), konvertuje ih u int i ažurira salu. Ovo je praktično kad admin u vizuelnom editoru prevuče sobu: front šalje nove koordinate/dimenzije, backend ih sigurno upisuje i momentalno vraća osvežen **SalaResource**. Route-model binding (Sala $sala) pojednostavljuje kod jer Laravel sam pronalazi instancu modela na osnovu parametra u ruti.

<?php

namespace App\Http\Controllers;

use App\Http\Resources\SalaResource;

use App\Models\Sala;

use Illuminate\Http\Request;

class SalaController extends Controller

{

    public function index()

    {

        $sale = cache()->remember('sale\_all', 60, fn () => Sala::query()->get());

        return response()->json(SalaResource::collection($sale), 200);

    }

    public function show($id)

    {

        $sala = Sala::findOrFail($id);

        return response()->json(new SalaResource($sala), 200);

    }

    public function store(Request $request)

    {

        // VALIDACIJA

        $validated = $request->validate([

            'naziv'      => 'required|string|max:255',

            'tip'        => 'required|string|max:255',

            'kapacitet'  => 'required|integer|min:1',

            'opis'       => 'nullable|string',

            'status'     => 'required|boolean',

            // NOVO – obavezni sprat i koordinate/veličina

            'floor'      => 'required|integer|min:0',

            'layout\_x'   => 'required|integer|min:0',

            'layout\_y'   => 'required|integer|min:0',

            'layout\_w'   => 'required|integer|min:1',

            'layout\_h'   => 'required|integer|min:1',

            // opciono

            'cena'       => 'nullable|numeric|min:0',

            'file\_path'  => 'nullable|string|max:1024',

        ]);

        // SANITIZACIJA (in-place, bez helpera)

        $validated['kapacitet'] = (int) max(1, (int) $validated['kapacitet']);

        $validated['floor']     = (int) max(0, (int) $validated['floor']);

        $validated['layout\_x']  = (int) max(0, (int) $validated['layout\_x']);

        $validated['layout\_y']  = (int) max(0, (int) $validated['layout\_y']);

        $validated['layout\_w']  = (int) max(1, (int) $validated['layout\_w']);

        $validated['layout\_h']  = (int) max(1, (int) $validated['layout\_h']);

        if (array\_key\_exists('cena', $validated)) {

            $validated['cena'] = (float) $validated['cena'];

        }

        $sala = Sala::create($validated);

        cache()->forget('sale\_all');

        return response()->json(new SalaResource($sala), 201);

    }

    public function update(Request $request, $id)

    {

        $sala = Sala::findOrFail($id);

        // VALIDACIJA (ista pravila kao za store)

        $validated = $request->validate([

            'naziv'      => 'required|string|max:255',

            'tip'        => 'required|string|max:255',

            'kapacitet'  => 'required|integer|min:1',

            'opis'       => 'nullable|string',

            'status'     => 'required|boolean',

            'floor'      => 'required|integer|min:0',

            'layout\_x'   => 'required|integer|min:0',

            'layout\_y'   => 'required|integer|min:0',

            'layout\_w'   => 'required|integer|min:1',

            'layout\_h'   => 'required|integer|min:1',

            'cena'       => 'nullable|numeric|min:0',

            'file\_path'  => 'nullable|string|max:1024',

        ]);

        // SANITIZACIJA

        $validated['kapacitet'] = (int) max(1, (int) $validated['kapacitet']);

        $validated['floor']     = (int) max(0, (int) $validated['floor']);

        $validated['layout\_x']  = (int) max(0, (int) $validated['layout\_x']);

        $validated['layout\_y']  = (int) max(0, (int) $validated['layout\_y']);

        $validated['layout\_w']  = (int) max(1, (int) $validated['layout\_w']);

        $validated['layout\_h']  = (int) max(1, (int) $validated['layout\_h']);

        if (array\_key\_exists('cena', $validated)) {

            $validated['cena'] = (float) $validated['cena'];

        }

        $sala->update($validated);

        cache()->forget('sale\_all');

        return response()->json(new SalaResource($sala), 200);

    }

    public function destroy($id)

    {

        $sala = Sala::findOrFail($id);

        $sala->delete();

        cache()->forget('sale\_all');

        return response()->json(['message' => 'Sala je uspešno obrisana.'], 200);

    }

    public function uploadFile(Request $request, $id)

    {

        $sala = Sala::findOrFail($id);

        $request->validate([

            'file' => 'required|file|mimes:jpg,png,pdf,doc,docx,zip|max:5120',

        ]);

        $path = $request->file('file')->store('sala\_files', 'public');

        $sala->update(['file\_path' => $path]);

        cache()->forget('sale\_all');

        return response()->json([

            'message'   => 'Fajl je uspešno otpremljen i povezan sa salom!',

            'file\_path' => asset('storage/' . $path),

            'sala'      => new SalaResource($sala->fresh()),

        ], 200);

    }

    public function updateLayout(Request $request, Sala $sala)

    {

        $data = $request->validate([

            'floor'     => 'sometimes|integer|min:0',

            'layout\_x'  => 'sometimes|integer|min:0',

            'layout\_y'  => 'sometimes|integer|min:0',

            'layout\_w'  => 'sometimes|integer|min:1',

            'layout\_h'  => 'sometimes|integer|min:1',

        ]);

        // clamp opcionalno (ako želiš da ograničiš max)

        foreach (['layout\_x','layout\_y','layout\_w','layout\_h','floor'] as $k) {

            if (array\_key\_exists($k, $data)) $data[$k] = (int) $data[$k];

        }

        $sala->update($data);

        cache()->forget('sale\_all');

        return response()->json(new \App\Http\Resources\SalaResource($sala->fresh()), 200);

    }

}

##### RezervacijaController

Kontroler upravlja CRUD operacijama nad rezervacijama i sve odgovore vraća kao JSON, standardizovane kroz **RezervacijaResource**. Metoda **index()** vraća kompletnu listu rezervacija, a **show($id)** čita i vraća jednu rezervaciju po ID-ju. U oba slučaja rezultat se omotava u resource kako bi front dobio doslednu strukturu podataka.

Kreiranje nove rezervacije radi **store(Request $request)**. Najpre se radi **validacija** unosa (sala mora postojati, datum i vremena su ispravnog formata, vreme\_do je posle vreme\_od, obavezan je tip događaja i status). Ako validacija padne, vraća se **422** sa porukama grešaka. Ako prođe, kontroler dopunjuje podatke sa user\_id = auth()->id() (rezervacija se vezuje za trenutno prijavljenog korisnika) i pravi zapis u bazi, a odgovor je **201** sa *RezervacijaResource*.

Izmena postojeće rezervacije je u **update(Request $request, $id)** i prati istu šemu kao *store*: dohvat po ID-ju, validacija svih polja, ažuriranje i povrat ažuriranog resource-a uz **200**. Brisanje radi **destroy($id)** – pronalazi zapis, briše ga i vraća poruku o uspehu.

Za listanje sa stranicama i osnovnim filterima tu je **paginatedAndFiltered(Request $request)**. Na osnovu opcionalnih parametara (tip\_dogadjaja, datum) gradi se upit, zatim se poziva paginate() (sa podrazumevanih 10 po strani ili vrednošću iz per\_page). Rezultat je klasičan Laravel paginator JSON (podatke + meta o stranicama), što je zgodno za tabele na frontu.

Metoda **exportToCsv()** služi za izvoz svih rezervacija u CSV. Koristi **StreamedResponse** – otvara php://output, upisuje zaglavlje kolona i svaku rezervaciju kao red, pa podešava *Content-Type* i *Content-Disposition* tako da pregledač preuzme fajl (ime uključuje datum i vreme). Ovaj pristup je memorijski efikasan i pogodan i za veće skupove podataka.

<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

use App\Http\Resources\RezervacijaResource;

use App\Models\Rezervacija;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Validator;

use Symfony\Component\HttpFoundation\StreamedResponse;

class RezervacijaController extends Controller

{

    public function index()

    {

        $rezervacije = Rezervacija::all();

        return response()->json(RezervacijaResource::collection($rezervacije), 200);

    }

    public function show($id)

    {

        $rezervacija = Rezervacija::findOrFail($id);

        return response()->json(new RezervacijaResource($rezervacija), 200);

    }

    public function store(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'sala\_id' => 'required|exists:salas,id',

            'datum' => 'required|date',

            'vreme\_od' => 'required|date\_format:H:i',

            'vreme\_do' => 'required|date\_format:H:i|after:vreme\_od',

            'tip\_dogadjaja' => 'required|string|max:255',

            'status' => 'required|string|max:255',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 422);

        }

        $data = $request->all();

        $data['user\_id'] = auth()->id(); // Koristi ID ulogovanog korisnika

        $rezervacija = Rezervacija::create($data);

        return response()->json(new RezervacijaResource($rezervacija), 201);

    }

    public function update(Request $request, $id)

    {

        $rezervacija = Rezervacija::findOrFail($id);

        $validator = Validator::make($request->all(), [

            'sala\_id' => 'required|exists:salas,id',

            'datum' => 'required|date',

            'vreme\_od' => 'required|date\_format:H:i',

            'vreme\_do' => 'required|date\_format:H:i|after:vreme\_od',

            'tip\_dogadjaja' => 'required|string|max:255',

            'status' => 'required|string|max:255',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 422);

        }

        $rezervacija->update($request->all());

        return response()->json(new RezervacijaResource($rezervacija), 200);

    }

    public function destroy($id)

    {

        $rezervacija = Rezervacija::findOrFail($id);

        $rezervacija->delete();

        return response()->json(['message' => 'Rezervacija je uspešno obrisana.'], 200);

    }

    public function paginatedAndFiltered(Request $request)

    {

        $query = Rezervacija::query();

        if ($request->has('tip\_dogadjaja')) {

            $query->where('tip\_dogadjaja', 'like', '%' . $request->tip\_dogadjaja . '%');

        }

        if ($request->has('datum')) {

            $query->where('datum', $request->datum);

        }

        $rezervacije = $query->paginate($request->get('per\_page', 10));

        return response()->json($rezervacije, 200);

    }

    public function exportToCsv()

    {

        $fileName = 'rezervacije\_' . now()->format('Y\_m\_d\_H\_i\_s') . '.csv';

        // Streamovani odgovor za preuzimanje fajla

        $response = new StreamedResponse(function () {

            // Kreiraj pokazivač na izlazni tok podataka (output)

            $handle = fopen('php://output', 'w');

            fputcsv($handle, ['ID', 'Sala ID', 'Datum', 'Vreme Od', 'Vreme Do', 'Tip Događaja', 'Status']);

            $rezervacije = Rezervacija::all();

            foreach ($rezervacije as $rezervacija) {

                fputcsv($handle, [

                    $rezervacija->id,

                    $rezervacija->sala\_id,

                    $rezervacija->datum,

                    $rezervacija->vreme\_od,

                    $rezervacija->vreme\_do,

                    $rezervacija->tip\_dogadjaja,

                    $rezervacija->status,

                ]);

            }

            fclose($handle);

        });

        $response->headers->set('Content-Type', 'text/csv');

        $response->headers->set('Content-Disposition', 'attachment; filename="' . $fileName . '"');

        return $response;

    }

}

##### PreporukaController

Kontroler je zadužen za CRUD nad entitetom *Preporuka* i svuda vraća JSON, standardizovan preko **PreporukaResource**. Metoda **index()** vraća sve preporuke iz baze kao kolekciju resursa uz status **200**, dok **show($id)** dohvaća jednu preporuku po ID-ju i vraća je kao pojedinačni resource.

Kreiranje nove preporuke radi **store(Request $request)**. Najpre se pokreće **validacija**: obavezno je da postoji sala (exists:salas,id), datum je ispravan, vremena su u formatu H:i i da je vreme\_do posle vreme\_od, kao i da je naveden tip\_dogadjaja. Ako validacija padne, vraća se **422** sa porukama grešaka. Ako prođe, iz tela zahteva se formira payload i dopuni **user\_id** vrednošću trenutno ulogovanog korisnika (**auth()->id()**), čime se preporuka vezuje za autora; zatim se kreira zapis i vraća **201** sa *PreporukaResource*.

Izmena postojeće preporuke je u **update(Request $request, $id)**. Preporuka se prvo dohvaća (404 ako ne postoji), zatim se primenjuju ista validaciona pravila kao kod kreiranja. Po uspehu se vrši update() i vraća ažurirani resource uz **200**.

Brisanje je u **destroy($id)**: pronalazi se zapis, briše i vraća se poruka o uspehu uz **200**.

<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

use App\Http\Resources\PreporukaResource;

use App\Models\Preporuka;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Validator;

class PreporukaController extends Controller

{

    public function index()

    {

        $preporuke = Preporuka::all();

        return response()->json(PreporukaResource::collection($preporuke), 200);

    }

    public function show($id)

    {

        $preporuka = Preporuka::findOrFail($id);

        return response()->json(new PreporukaResource($preporuka), 200);

    }

    public function store(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'sala\_id' => 'required|exists:salas,id',

            'datum' => 'required|date',

            'vreme\_od' => 'required|date\_format:H:i',

            'vreme\_do' => 'required|date\_format:H:i|after:vreme\_od',

            'tip\_dogadjaja' => 'required|string|max:255',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 422);

        }

        $data = $request->all();

        $data['user\_id'] = auth()->id(); // Koristi ID ulogovanog korisnika

        $preporuka = Preporuka::create($data);

        return response()->json(new PreporukaResource($preporuka), 201);

    }

    public function update(Request $request, $id)

    {

        $preporuka = Preporuka::findOrFail($id);

        $validator = Validator::make($request->all(), [

            'sala\_id' => 'required|exists:salas,id',

            'datum' => 'required|date',

            'vreme\_od' => 'required|date\_format:H:i',

            'vreme\_do' => 'required|date\_format:H:i|after:vreme\_od',

            'tip\_dogadjaja' => 'required|string|max:255',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 422);

        }

        $preporuka->update($request->all());

        return response()->json(new PreporukaResource($preporuka), 200);

    }

    public function destroy($id)

    {

        $preporuka = Preporuka::findOrFail($id);

        $preporuka->delete();

        return response()->json(['message' => 'Preporuka je uspešno obrisana.'], 200);

    }

}

##### AdminDasboardController

Ovaj **AdminDashboardController** služi kao backend izvor podataka za administratorsku kontrolnu tablu i vraća agregirane statistike u JSON formatu. Metoda **kpis()** sabira osnovne pokazatelje uspeha – ukupan broj korisnika, sala, aktivnih sala, sve rezervacije i rezervacije za današnji dan – i kešira rezultat na 60 sekundi kako bi se rasteretila baza. Metoda **reservationsDaily()** priprema seriju za linijski graf: za poslednjih *N* dana (podrazumevano 30) računa broj rezervacija po datumu, dopunjava prazne dane nulama i vraća paralelne nizove labels (YYYY-MM-DD) i data (brojevi).

Metoda **topRooms()** služi za bar chart „najzauzetijih“ sala: grupiše rezervacije po sala\_id, uzima prvih *limit* (10 po difoltu), pa preko modela Sala dovozi čitljiva imena i vraća labels (nazivi sala) i data (broj rezervacija). Funkcija **roomsByType()** formira podatke za pie/donut graf – broji koliko sala ima po tipu (npr. sala za sastanke, učionica…) i vraća liste tipova i odgovarajućih brojčanih vrednosti.

Za analizu po satima, **hourly()** gleda poslednjih *N* dana (14 po difoltu), grupiše rezervacije po početnom satu vreme\_od (00–23) i vraća niz od 24 stavke, zgodan za heatmapu ili kolone. Na kraju, **utilization()** računa „iskorišćenost“: koliko je aktivnih sala u sistemu i koliko ih je imalo bar jednu rezervaciju u poslednjih 30 dana, zatim izračuna odnos busy/active kao metrik uvida. Sve metode vraćaju jednostavne, „chart-friendly“ JSON strukture (labels+data), spremne za direktno povezivanje sa grafičkim komponentama na frontendu.

<?php

namespace App\Http\Controllers;

use App\Models\Rezervacija;

use App\Models\Sala;

use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Cache;

use Illuminate\Support\Facades\DB;

class AdminDashboardController extends Controller

{

    // Osnovni KPI-ji (keš 60s)

    public function kpis()

    {

        $data = Cache::remember('admin.kpis', 60, function () {

            return [

                'users\_total'        => User::count(),

                'rooms\_total'        => Sala::count(),

                'rooms\_active'       => Sala::where('status', true)->count(),

                'reservations\_total' => Rezervacija::count(),

                'reservations\_today' => Rezervacija::whereDate('datum', today())->count(),

            ];

        });

        return response()->json($data, 200);

    }

    // Linijski graf: broj rezervacija po danu (poslednjih N dana)

    public function reservationsDaily(Request $request)

    {

        $days = (int) $request->get('days', 30);

        $from = now()->subDays($days - 1)->startOfDay();

        $rows = Rezervacija::select([

                DB::raw('DATE(datum) as day'),

                DB::raw('COUNT(\*) as cnt'),

            ])

            ->whereDate('datum', '>=', $from)

            ->groupBy('day')

            ->orderBy('day')

            ->get();

        // popuni sve dane

        $series = [];

        for ($i = 0; $i < $days; $i++) {

            $d = $from->copy()->addDays($i)->toDateString();

            $series[$d] = 0;

        }

        foreach ($rows as $r) $series[$r->day] = (int) $r->cnt;

        return response()->json([

            'labels' => array\_keys($series),

            'data'   => array\_values($series),

        ], 200);

    }

    // Bar chart: top sale po broju rezervacija

    public function topRooms(Request $request)

    {

        $limit = (int) $request->get('limit', 10);

        $rows = Rezervacija::select('sala\_id', DB::raw('COUNT(\*) as cnt'))

            ->groupBy('sala\_id')

            ->orderByDesc('cnt')

            ->limit($limit)

            ->get();

        $names = Sala::whereIn('id', $rows->pluck('sala\_id'))

            ->pluck('naziv', 'id');

        return response()->json([

            'labels' => $rows->map(fn($r) => $names[$r->sala\_id] ?? ('Sala #'.$r->sala\_id)),

            'data'   => $rows->pluck('cnt')->map(fn($x) => (int) $x),

        ], 200);

    }

    // Pie/donut: raspodela sala po tipu

    public function roomsByType()

    {

        $rows = Sala::select('tip', DB::raw('COUNT(\*) as cnt'))

            ->groupBy('tip')

            ->orderByDesc('cnt')

            ->get();

        return response()->json([

            'labels' => $rows->pluck('tip'),

            'data'   => $rows->pluck('cnt')->map(fn($x) => (int) $x),

        ], 200);

    }

    // Heatmap/kolona: broj rezervacija po satu (početni sat), poslednjih N dana

    public function hourly(Request $request)

    {

        $days = (int) $request->get('days', 14);

        $from = now()->subDays($days - 1)->startOfDay();

        $rows = Rezervacija::select([

                DB::raw("DATE\_FORMAT(vreme\_od, '%H') as hour"),

                DB::raw('COUNT(\*) as cnt'),

            ])

            ->whereDate('datum', '>=', $from)

            ->groupBy('hour')

            ->orderBy('hour')

            ->get();

        $series = array\_fill(0, 24, 0);

        foreach ($rows as $r) {

            $h = (int) $r->hour;

            if ($h >= 0 && $h <= 23) $series[$h] = (int) $r->cnt;

        }

        return response()->json([

            'labels' => range(0, 23),

            'data'   => $series,

        ], 200);

    }

    // Utilization: koliko aktivnih sala je imalo makar jednu rezervaciju u 30 dana

    public function utilization()

    {

        $active = Sala::where('status', true)->count();

        $busy   = Rezervacija::whereDate('datum', '>=', now()->subDays(30))

                    ->distinct('sala\_id')->count('sala\_id');

        return response()->json([

            'active\_rooms' => $active,

            'busy\_rooms'   => $busy,

            'ratio'        => $active > 0 ? round($busy / $active, 3) : 0,

        ], 200);

    }

}

#### Laravel Autentifikacija

Autentifikacija korisnika je ključni deo svake web aplikacije, a Laravel poseduje ugrađen sistem koji ovaj proces čini jednostavnim. Laravel-ov ugrađeni mehanizam za autentifikaciju **automatizuje** mnoge uobičajene zadatke poput registracije novih korisnika, prijavljivanja (login), resetovanja zaboravljene lozinke i upravljanja sesijama korisnika. Ove funkcionalnosti omogućavaju programerima da se fokusiraju na izgradnju specifičnih osobina aplikacije, dok framework brine o bezbednom čuvanju lozinki (heširanjem), validaciji korisničkih kredencijala, čuvanju podataka o prijavi u sesiji i zaštiti ruta koje zahtevaju prijavljenog korisnika. Laravel-ova autentifikacija je usko integrisana sa sistemom rutiranja i midlvera – npr. dovoljno je dodati midlver auth na određenu rutu da bi se pristup dozvolio samo prijavljenim korisnicima.

U pozadini, Laravel-ov sistem autentifikacije koristi koncepte **“guard”** i **“provider”**. *Guard* definiše na koji način se proverava da li je korisnik autentifikovan (npr. putem sesije za web, ili tokena za API), dok *provider* određuje odakle se učitavaju korisnički podaci (npr. iz koje tabele baze). Podrazumevana autentifikacija koristi session cookie (web guard) za klasične web aplikacije. Laravel takođe podržava **token-based** autentifikaciju za API-je: paket **Laravel Sanctum** omogućava jednostavno izdavanje i verifikaciju API tokena pogodnih za single-page aplikacije ili mobilne klijente, dok **Laravel Passport** nudi kompletnu OAuth2 implementaciju za složenije potrebe (uključujući izdavanje pristupnih tokena za third-party klijente). Na ovaj način, framework obezbeđuje fleksibilnu autentifikaciju – od jednostavnog login sistema do zaštite REST API poziva – uz visok nivo bezbednosti (heširanje lozinki, zaštita od CSRF napada, itd.).

##### UserController

Kontroler **UserController** obezbeđuje CRUD logiku nad korisnicima uz osnovne zaštite i validaciju. Metoda **index** prima opciono parametre q (tekst za pretragu) i per\_page (veličina strane). Ako je prosleđen upit, pretraga se vrši nad imenom, email-om i ulogom, rezultati se sortiraju opadajuće po ID-ju i paginiraju, a odgovor vraća standardni Laravel paginator JSON (sa metapodacima o strani). Metoda **show** vraća JSON prikaz konkretnog korisnika preko route model binding-a.

Metoda **store** validira ulaz (obavezni name, jedinstveni email, lozinka minimalne dužine i role ograničena na korisnik|menadzer|administrator). Lozinka se pre snimanja hešuje (Hash::make), pa se kreira novi zapis i vraća se 201 Created sa telom novog korisnika. Metoda **update** dozvoljava delimične izmene: validira svako polje ako je prosleđeno, vodi računa da je email jedinstven uz izuzetak trenutnog korisnika (ignore($user->id)), a lozinku hešuje samo ako nije prazna (u suprotnom je uklanja iz seta izmena). Dodatno, ugrađena je zaštita da korisnik ne može sebi skinuti administratorsku ulogu: ako je autentifikovani korisnik isti kao korisnik koji se menja i nova uloga nije administrator, vraća se 422 sa porukom. Nakon uspešnog ažuriranja, vraća se 200 sa ažuriranim JSON-om.

Metoda **destroy** takođe ima zaštitu od samobrisanja naloga: ako pokušate obrisati nalog koji je trenutno prijavljen (auth()->id() === $user->id), vraća se 422 sa porukom; u suprotnom, zapis se briše i vraća se 200 sa kratkom potvrdom. U celini, kontroler jasno razdvaja odgovornosti: pretraga i paginacija za listanje, stroga validacija i heširanje lozinki za kreiranje/izmenu, pravila integriteta za uloge i zabrana samobrisanja, a svi odgovori su dosledni JSON-u i smislenim HTTP status kodovima.

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Hash;

use Illuminate\Validation\Rule;

class UserController extends Controller

{

    public function index(Request $request)

    {

        $q   = trim((string) $request->get('q', ''));

        $per = (int) $request->get('per\_page', 15);

        $users = User::query()

            ->when($q !== '', function ($query) use ($q) {

                $query->where(function ($w) use ($q) {

                    $w->where('name', 'like', "%$q%")

                      ->orWhere('email', 'like', "%$q%")

                      ->orWhere('role', 'like', "%$q%");

                });

            })

            ->orderByDesc('id')

            ->paginate($per);

        return response()->json($users, 200);

    }

    public function show(User $user)

    {

        return response()->json($user, 200);

    }

    public function store(Request $request)

    {

        $validated = $request->validate([

            'name'     => ['required','string','max:255'],

            'email'    => ['required','email','max:255','unique:users,email'],

            'password' => ['required','string','min:6'],

            'role'     => ['required', Rule::in(['korisnik','menadzer','administrator'])],

        ]);

        $validated['password'] = Hash::make($validated['password']);

        $user = User::create($validated);

        return response()->json($user, 201);

    }

    public function update(Request $request, User $user)

    {

        $validated = $request->validate([

            'name'     => ['sometimes','string','max:255'],

            'email'    => ['sometimes','email','max:255', Rule::unique('users','email')->ignore($user->id)],

            'password' => ['sometimes','nullable','string','min:6'],

            'role'     => ['sometimes', Rule::in(['korisnik','menadzer','administrator'])],

        ]);

        if (isset($validated['password']) && $validated['password']) {

            $validated['password'] = Hash::make($validated['password']);

        } else {

            unset($validated['password']);

        }

        // zaštita: ne menjaš sam sebi admin ulogu

        if (auth()->id() === $user->id && isset($validated['role']) && $validated['role'] !== 'administrator') {

            return response()->json(['message' => 'Ne možete sebi skinuti administratorsku ulogu.'], 422);

        }

        $user->update($validated);

        return response()->json($user, 200);

    }

    public function destroy(User $user)

    {

        if (auth()->id() === $user->id) {

            return response()->json(['message' => 'Ne možete obrisati sopstveni nalog.'], 422);

        }

        $user->delete();

        return response()->json(['message' => 'Korisnik obrisan.'], 200);

    }

}

##### PasswordController

Kontroler **PasswordController** sadrži dva glavna endpointa za tok „zaboravljena lozinka“. Metoda **posaljiLink** najpre validira da je email prosleđen, ispravan i da postoji u tabeli korisnika. Zatim, preko Password::broker()->sendResetLink() zatraži od Laravel password brokera da generiše token i pošalje email sa linkom za reset lozinke. Na osnovu statusa vraća jasne HTTP odgovore: 200 kada je link uspešno poslat, 429 ako je korisnik premašio dozvoljeni broj pokušaja (throttling), 404 za nepostojećeg korisnika (što se u praksi ne bi desilo zbog exists pravila), a za sve neočekivane ishode vraća 400 i upisuje upozorenje u log. U slučaju izuzetka (npr. problem sa mailerom) metoda hvata grešku, loguje detalje i vraća 500 sa porukom da je došlo do greške pri slanju email-a.

Metoda **restartujLozinku** obrađuje drugi korak procesa: reset same lozinke. Ponovo se radi validacija ulaza — proverava se email, prisustvo i format tokena, kao i nova lozinka sa potvrdom i minimalnom dužinom. Zatim se poziva Password::reset() koji, ako je token važeći i email odgovara korisniku, u prosleđenom kolbeku bezbedno menja lozinku (bcrypt) i snima je u bazu. Ako je sve prošlo u redu, vraća se 200 uz poruku da je lozinka promenjena; u suprotnom se vraća 400 sa status kodom koje je broker vratio (npr. istekao/nevažeći token). Ovakav pristup jasno razdvaja generisanje i slanje linka od samog resetovanja, obezbeđuje konzistentne HTTP statuse, logging za dijagnostiku i striktna validaciona pravila radi bezbednosti.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Password;

use Illuminate\Validation\ValidationException;

use Illuminate\Support\Facades\Log;

class PasswordController extends Controller

{

    public function posaljiLink(Request $request)

    {

        // 1) validacija (ako padne, vrati 422 sa detaljima)

        $request->validate([

            'email' => 'required|email|exists:users,email',

        ]);

        try {

            // 2) pošalji link

            $status = Password::broker()->sendResetLink(

                $request->only('email')

            );

            // 3) mapiranje statusa na jasne odgovore

            switch ($status) {

                case Password::RESET\_LINK\_SENT:

                    return response()->json([

                        'message' => 'Link za reset lozinke je poslat na email.',

                        'status'  => $status,

                    ], 200);

                case Password::RESET\_THROTTLED: // previše pokušaja

                    return response()->json([

                        'message' => 'Previše pokušaja. Pokušajte ponovo za nekoliko minuta.',

                        'status'  => $status,

                    ], 429);

                case Password::INVALID\_USER: // teoretski ne bi trebalo jer exists:users,email

                    return response()->json([

                        'message' => 'Korisnik sa tim email-om ne postoji.',

                        'status'  => $status,

                    ], 404);

                default:

                    // loguj nepoznat status

                    Log::warning('Password reset unexpected status', ['status' => $status]);

                    return response()->json([

                        'message' => 'Neuspešno slanje linka za reset lozinke.',

                        'status'  => $status,

                    ], 400);

            }

        } catch (\Throwable $e) {

            // ako mailer padne ili nešto drugo

            Log::error('Password reset exception', ['error' => $e->getMessage()]);

            return response()->json([

                'message' => 'Greška pri slanju email-a za reset lozinke.',

                'error'   => app()->environment('local') ? $e->getMessage() : null,

            ], 500);

        }

    }

    public function restartujLozinku(Request $request)

    {

        $request->validate([

            'email'    => 'required|email|exists:users,email',

            'password' => 'required|string|min:8|confirmed',

            'token'    => 'required|string',

        ]);

        $status = Password::reset(

            $request->only('email', 'password', 'password\_confirmation', 'token'),

            function ($user, $password) {

                $user->forceFill(['password' => bcrypt($password)])->save();

            }

        );

        if ($status === Password::PASSWORD\_RESET) {

            return response()->json(['message' => 'Lozinka je uspešno promenjena.'], 200);

        }

        return response()->json([

            'message' => 'Neuspešno resetovanje lozinke.',

            'status'  => $status

        ], 400);

    }

}

#### Laravel Resursi

Laravel resursi (Resource) su sloj za predstavljanje podataka ka API-ju: umesto da direktno vraćaš Eloquent modele ili nizove, preko resursa tačno definišeš kako će izgledati JSON izlaz. Time odvajaš unutrašnju strukturu baze od javnog formata odgovora, pa možeš bezbedno da sakriješ osetljiva polja (npr. password, interni ID-evi), preimenuješ atribute, formatiraš datume i dosledno isporučuješ isti oblik podataka kroz ceo API.

Resurs je klasa koja nasleđuje Illuminate\Http\Resources\Json\JsonResource i implementira metodu toArray($request) u kojoj mapiraš model na željeni JSON. Za liste se koristi Resource kolekcija (::collection($models) ili posebna ResourceCollection) koja obezbeđuje isti, ujednačen oblik za više stavki i lepo sarađuje sa paginacijom (automatski dodaje links i meta kada vratiš paginator).

Velika prednost je i uslovno uključivanje polja: helperi poput when($uslov, $vrednost) i whenLoaded('relacija') omogućavaju da dodaš atribute ili relacije samo kad su stvarno dostupni ili traženi (npr. kada si eager-loadovao relaciju). Preko with() i additional() možeš da dodaš globalne meta-podatke ili linkove na vrhu odgovora, što je korisno za HATEOAS stil ili prenos pomoćnih informacija.

U praksi, kontroler vrati nešto poput return new UserResource($user); ili return UserResource::collection($users);, a cela transformacija ostaje centralizovana u Resource klasama. Rezultat su stabilni, verzionisani API izlazi koji su jednostavni za održavanje, sigurniji (jer ne „cure” interna polja) i konzistentni za frontend tim.

#### Laravel Rutiranje

API rutiranje je način na koji aplikacija mapira dolazne HTTP zahteve na odgovarajuću logiku u backendu. HTTP je protokol preko koga klijent (npr. browser ili frontend SPA) razgovara sa serverom koristeći metode poput GET, POST, PUT i DELETE. Svaki zahtev ide na konkretan URL put (npr. /rezervacije/15) i nosi zaglavlja, telo ili query parametre; server na osnovu rute i metode odlučuje koju kontrolersku akciju da pozove i kakav JSON odgovor da vrati. U Laravelu se to definiše u fajlovima sa rutama, gde se navode putanje, metode, kontroleri i middleware slojevi. Middleware je zaštitni filter koji se izvršava pre ili posle kontrolera (autentikacija, provera uloge, throttling…), a resursne rute prate REST konvencije pa automatski povezuju standardne HTTP metode sa CRUD operacijama.

U prikazanom api.php najpre se podešavaju rute za autentikaciju i identitet korisnika. Ruta /user je zaštićena Sanctum middleware-om i vraća trenutno prijavljenog korisnika. Zatim slede javne rute za registraciju i prijavu, dok je odjava zaštićena jer zahteva validan token. Mehanizam „zaboravljena lozinka” ima dve rute: jedna prima email i šalje link sa tokenom, druga prima token, email i novu lozinku i menja je. Ove rute rade bez prijave jer služe upravo za pristup kada korisnik nema validnu sesiju.

Sekcija za sale je podeljena na javni deo i administratorski deo. Javno su dostupni pregled liste i detalja sale (katalog i mapa treba da rade svima), a operacije kreiranja, izmene, brisanja i upload fajla su u grupi zaštićenoj Sanctum autentikacijom i „role” middleware-om koji dopušta pristup samo administratoru i menadžeru. Tu je i dodatna GET ruta za serversku pretragu sala. Ovakva podela omogućava da korisnici vide ponudu, dok samo ovlašćeni akteri upravljaju podacima.

Rute za rezervacije stavljene su u grupu koja traži prijavu i jednu od uloga „administrator, menadžer ili korisnik”. Unutar te grupe nalaze se paginacija i CSV izvoz za administrativne potrebe, kao i kompletan CRUD. Time se štiti privatnost i sprečava da rezervacije budu javno dostupne. Slično je i sa preporukama: čitanje, kreiranje, izmena i brisanje zahtevaju prijavu, a dodatno postoji serverska akcija za „suggest” koja bi mogla da računa i vraća optimalne sale/termine na osnovu nekih pravila ili istorije.

|  |  |
| --- | --- |
| Opis funkcije | **Pregled sala** |
| HTTP metoda | GET |
| URL | /api/sale |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | [  {  "id": 1,  "naziv": "aut",  "tip": "sala za sastanke",  "kapacitet": 41,  "opis": "Numquam voluptatum qui modi vero eos itaque.",  "status": 1,  "cena": null,  "file\_path": null,  "manager\_id": null,  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  },  {  "id": 2,  "naziv": "accusamus",  "tip": "konferencijska sala",  "kapacitet": 40,  "opis": "Necessitatibus explicabo ea omnis adipisci iste adipisci accusantium.",  "status": 1,  "cena": null,  "file\_path": null,  "manager\_id": null,  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  }, |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Pregled sale** |
| HTTP metoda | GET |
| URL | /api/sale/1 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "id": 1,  "naziv": "aut",  "tip": "sala za sastanke",  "kapacitet": 41,  "opis": "Numquam voluptatum qui modi vero eos itaque.",  "status": 1,  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z",  "cena": null  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Dodavanje sale** |
| HTTP metoda | POST |
| URL | /api/sale |
| URL parametri | (nema) |
| HTTP body parametri | {  "naziv": "Velika sala",  "tip": "konferencijska",  "kapacitet": 100,  "opis": "Sala za velike skupove sa kompletnom tehničkom opremom.",  "status": true,  "cena": 500.00  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 47,  "naziv": "Velika sala",  "tip": "konferencijska",  "kapacitet": 100,  "opis": "Sala za velike skupove sa kompletnom tehničkom opremom.",  "status": true,  "created\_at": "2024-12-15T21:33:47.000000Z",  "updated\_at": "2024-12-15T21:33:47.000000Z",  "cena": 500  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Dodavanje fajla u salu** |
| HTTP metoda | POST |
| URL | /api/sale/upload/46 |
| URL parametri | (nema) |
| HTTP body parametri | {  "file": "sala.png"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "message": "Fajl je uspešno otpremljen i povezan sa salom!",  "file\_path": "http://127.0.0.1:8000/storage/sala\_files/aQ2jkfU74g6lsZtazVGsQV21S7StlznlFMT04oQI.png",  "sala": {  "id": 46,  "naziv": "Velika sala",  "tip": "konferencijska",  "kapacitet": 100,  "opis": "Sala za velike skupove sa kompletnom tehničkom opremom.",  "status": 1,  "cena": "500.00",  "file\_path": "sala\_files/aQ2jkfU74g6lsZtazVGsQV21S7StlznlFMT04oQI.png",  "manager\_id": null,  "created\_at": "2024-12-15T21:27:06.000000Z",  "updated\_at": "2024-12-15T21:28:20.000000Z"  }  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Izmena sale** |
| HTTP metoda | PUT |
| URL | /api/sale/46 |
| URL parametri | (nema) |
| HTTP body parametri | {  "naziv": "Velika sala",  "tip": "konferencijska",  "kapacitet": 100,  "opis": "Sala za velike skupove sa kompletnom tehničkom opremom.",  "status": true,  "cena": 501.00  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 46,  "naziv": "Velika sala",  "tip": "konferencijska",  "kapacitet": 100,  "opis": "Sala za velike skupove sa kompletnom tehničkom opremom.",  "status": true,  "created\_at": "2024-12-15T21:27:06.000000Z",  "updated\_at": "2024-12-15T21:28:46.000000Z",  "cena": 501  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Brisanje sale** |
| HTTP metoda | DELETE |
| URL | /api/sale/46 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "message": "Sala je uspešno obrisana."  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Prikaz svih rezervacija** |
| HTTP metoda | GET |
| URL | /api/rezervacije |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | [  {  "id": 1,  "sala\_id": 11,  "user\_id": 1,  "datum": "2002-03-06",  "vreme\_od": "18:07:09",  "vreme\_do": "13:55:00",  "tip\_dogadjaja": "predavanje",  "status": "potvrđeno",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  },  {  "id": 2,  "sala\_id": 12,  "user\_id": 2,  "datum": "1981-10-21",  "vreme\_od": "11:51:35",  "vreme\_do": "10:05:00",  "tip\_dogadjaja": "konferencija",  "status": "na čekanju",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  }, |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Prikaz jedne rezervacije** |
| HTTP metoda | GET |
| URL | /api/rezervacije/1 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "id": 1,  "sala\_id": 11,  "user\_id": 1,  "datum": "2002-03-06",  "vreme\_od": "18:07:09",  "vreme\_do": "13:55:00",  "tip\_dogadjaja": "predavanje",  "status": "potvrđeno",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Izvoz rezervacija u CSV** |
| HTTP metoda | GET |
| URL | /api/rezervacije/export/csv |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | ID,"Sala ID",Datum,"Vreme Od","Vreme Do","Tip Događaja",Status  1,11,2005-06-27,22:54:45,23:18:00,predavanje,"na čekanju"  2,12,2003-04-08,12:49:10,13:21:00,konferencija,"na čekanju"  3,13,2008-06-17,00:20:24,19:32:00,predavanje,otkazano  4,14,1984-09-29,17:17:22,19:21:00,sastanak,"na čekanju"  5,15,1975-11-17,21:10:15,18:15:00,sastanak,"na čekanju"  6,16,1978-08-20,01:05:34,18:37:00,predavanje,otkazano |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Filter i paginacija rezervacije** |
| HTTP metoda | GET |
| URL | /api/rezervacije/paginacija |
| URL parametri | tip\_dogadjaja=predavanje |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "current\_page": 1,  "data": [  {  "id": 1,  "sala\_id": 11,  "user\_id": 1,  "datum": "2002-03-06",  "vreme\_od": "18:07:09",  "vreme\_do": "13:55:00",  "tip\_dogadjaja": "predavanje",  "status": "potvrđeno",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  },  {  "id": 4,  "sala\_id": 14,  "user\_id": 4,  "datum": "2014-08-18",  "vreme\_od": "20:25:37",  "vreme\_do": "05:05:00",  "tip\_dogadjaja": "predavanje",  "status": "otkazano",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  }, |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Dodavanje rezervacije** |
| HTTP metoda | POST |
| URL | /api/rezervacije |
| URL parametri | (nema) |
| HTTP body parametri | {  "sala\_id": 1,  "datum": "2024-12-15",  "vreme\_od": "10:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Sastanak",  "status": "na čekanju"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 21,  "sala\_id": 1,  "user\_id": 36,  "datum": "2024-12-15",  "vreme\_od": "10:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Sastanak",  "status": "na čekanju",  "created\_at": "2024-12-15T21:30:50.000000Z",  "updated\_at": "2024-12-15T21:30:50.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Izmena rezervacije** |
| HTTP metoda | PUT |
| URL | /api/rezervacije/21 |
| URL parametri | (nema) |
| HTTP body parametri | {  "sala\_id": 1,  "datum": "2024-12-15",  "vreme\_od": "10:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Sastanak",  "status": "potvrđeno"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 21,  "sala\_id": 1,  "user\_id": 36,  "datum": "2024-12-15",  "vreme\_od": "10:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Sastanak",  "status": "potvrđeno",  "created\_at": "2024-12-15T21:30:50.000000Z",  "updated\_at": "2024-12-15T21:31:12.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Brisanje rezervacije** |
| HTTP metoda | DELETE |
| URL | /api/rezervacije/21 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "message": "Rezervacija je uspešno obrisana."  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Prikaz svih preporuka** |
| HTTP metoda | GET |
| URL | /api/preporuke |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | [  {  "id": 1,  "sala\_id": 31,  "user\_id": 21,  "datum": "1982-12-05",  "vreme\_od": "10:51:20",  "vreme\_do": "04:16:00",  "tip\_dogadjaja": "predavanje",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  },  {  "id": 2,  "sala\_id": 32,  "user\_id": 22,  "datum": "1972-08-10",  "vreme\_od": "07:13:39",  "vreme\_do": "22:31:00",  "tip\_dogadjaja": "sastanak",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  }, |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Prikaz jedne preporuke** |
| HTTP metoda | GET |
| URL | /api/preporuke/1 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "id": 1,  "sala\_id": 31,  "user\_id": 21,  "datum": "1982-12-05",  "vreme\_od": "10:51:20",  "vreme\_do": "04:16:00",  "tip\_dogadjaja": "predavanje",  "created\_at": "2024-12-15T21:19:54.000000Z",  "updated\_at": "2024-12-15T21:19:54.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Dodavanje preporuke** |
| HTTP metoda | POST |
| URL | /api/preporuke |
| URL parametri | (nema) |
| HTTP body parametri | {  "sala\_id": 1,  "datum": "2024-12-20",  "vreme\_od": "09:00",  "vreme\_do": "11:00",  "tip\_dogadjaja": "Predavanje"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 16,  "sala\_id": 1,  "user\_id": 36,  "datum": "2024-12-20",  "vreme\_od": "09:00",  "vreme\_do": "11:00",  "tip\_dogadjaja": "Predavanje",  "created\_at": "2024-12-15T21:32:41.000000Z",  "updated\_at": "2024-12-15T21:32:41.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Izmena preporuke** |
| HTTP metoda | PUT |
| URL | /api/preporuke/16 |
| URL parametri | (nema) |
| HTTP body parametri | {  "sala\_id": 1,  "datum": "2024-12-20",  "vreme\_od": "09:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Predavanje"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "id": 16,  "sala\_id": 1,  "user\_id": 36,  "datum": "2024-12-20",  "vreme\_od": "09:00",  "vreme\_do": "12:00",  "tip\_dogadjaja": "Predavanje",  "created\_at": "2024-12-15T21:32:41.000000Z",  "updated\_at": "2024-12-15T21:32:59.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Brisanje preporuke** |
| HTTP metoda | DELETE |
| URL | /api/preporuke/16 |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "message": "Preporuka je uspešno obrisana."  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Registracija** |
| HTTP metoda | POST |
| URL | /api/registracija |
| URL parametri | (nema) |
| HTTP body parametri | {  "name": "Jelena",  "email": "jelena@gmail.com",  "password": "jela1234",  "password\_confirmation": "jela1234"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "user": {  "id": 36,  "name": "Jelena",  "email": "jelena@gmail.com",  "created\_at": "2024-12-15T21:20:30.000000Z",  "updated\_at": "2024-12-15T21:20:30.000000Z"  },  "token": "1|FV3RLtJLsqINEp7QovcXh7xRjt49aYlu1seEtCwj",  "message": "Registracija je uspešna!"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Prijava** |
| HTTP metoda | POST |
| URL | /api/prijava |
| URL parametri | (nema) |
| HTTP body parametri | {  "email": "jelena@gmail.com",  "password": "jela1234"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "user": {  "id": 36,  "name": "Jelena",  "email": "jelena@gmail.com",  "created\_at": "2024-12-15T21:20:30.000000Z",  "updated\_at": "2024-12-15T21:20:30.000000Z"  },  "token": "2|5rGWZlJritTnkeAvCTqsDlX9qr7yBM9HF77ZWNwn",  "message": "Prijava je uspešna!"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Odjava** |
| HTTP metoda | POST |
| URL | /api/odjava |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "message": "Uspešno ste se odjavili!"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Konverzija valuta** |
| HTTP metoda | POST |
| URL | /api/sala/47/konverzija/EUR |
| URL parametri | (nema) |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {  "sala": "Velika sala",  "osnovna\_cena": "500.00 USD",  "konvertovana\_cena": "476.50 EUR",  "kurs": 0.953  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Slanje mejla za reset lozinke** |
| HTTP metoda | POST |
| URL | /api/password/email |
| URL parametri | (nema) |
| HTTP body parametri | {  "email": "jelena@gmail.com"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "message": "Link za resetovanje lozinke je poslat na email."  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Resetovanje lozinke** |
| HTTP metoda | POST |
| URL | /api/password/reset |
| URL parametri | (nema) |
| HTTP body parametri | {  "email": "jelena@gmail.com",  "password": "nova\_lozinka",  "password\_confirmation": "nova\_lozinka",  "token": "2dfce29d34e35e76d47f6b2a991b4eedaf9666500f45e682daa6432473de3c25"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {  "message": "Lozinka je uspešno promenjena."  } |
| Format izlaznih parametara | application/json |

#### Laravel Database Seeder

DatabaseSeeder je centralna “polazna” klasa u Laravelu zadužena za pokretanje svih seedera – skripti koje popunjavaju bazu početnim ili test podacima. Kada pokreneš seeding, Laravel izvršava upravo ovu klasu, a ona u svom run() metodu poziva ostale seeder klase (npr. UsersTableSeeder, RoomsTableSeeder…), čime se sistematski i u kontrolisanom redosledu pune sve relevantne tabele.

Svrha seeding-a je da obezbedi pouzdanu, ponovljivu inicijalizaciju podataka: možeš brzo da postaviš admin nalog, osnovne šifre/šifarnike, demo sale i uzorke rezervacija – bez ručnog unosa. To je posebno važno za razvoj i testiranje, jer tim uvek može da podigne “čisto” okruženje sa istim početnim setom podataka i tako lakše reprodukuje probleme ili pokreće automatizovane testove.

DatabaseSeeder često kombinuje seedere sa fabrikama (model factories), što omogućava generisanje realističnih, ali sintetičkih podataka u velikim količinama. Na primer, možeš u jednoj liniji da napraviš 50 korisnika i za svakog po nekoliko rezervacija, uz slučajne datume i statuse. Tako dobijeni podaci lepo pune interfejs i otkrivaju rubne slučajeve u UI/validacijama.

U praksi, DatabaseSeeder se pokreće Artisan komandama (npr. php artisan db:seed ili php artisan migrate:fresh --seed). Preporuka je da seederi budu idempotentni ili da pažljivo resetuješ bazu pre seeding-a, kako bi izbegao dupliranje ili konflikte. U produkciji seeding se koristi štedljivo (npr. za inicijalne šifarnike i admin nalog), dok je u dev/test okruženju standardni deo toka rada.

    public function run()

    {

        $this->call([

            SalaSeeder::class,

            RezervacijaSeeder::class,

            PreporukaSeeder::class,

        ]);

        // ADMIN korisnik

        User::updateOrCreate(

            ['email' => 'admin@gmail.com'],

            [

                'name' => 'Admin',

                'password' => Hash::make('admin12345'),

                'role' => 'administrator',

            ]

        );

    }

#### React Hooks

React hooks su API mehanizam koji omogućava funkcijskim komponentama da imaju stanje, životni ciklus i druge mogućnosti bez pisanja klasa. Ideja je da “zakačiš” (hook) logiku za React-ov render tok i tako deliš i kombinuješ ponašanja kroz male, ponovo upotrebljive komade koda (mogu i kao prilagođeni – “custom” hookovi). Postoje osnovne “kuke”: useState za stanje, useEffect za sporedne efekte (pozivi API-ja, subscriptioni…), useContext za čitanje konteksta, useReducer za složenije upravljanje stanjem, useRef za mutable reference i pristup DOM-u, useMemo i useCallback za memorisanje vrednosti i funkcija radi performansi, useLayoutEffect za efekte pre iscrtavanja, useImperativeHandle za izlaganje kontrolisanog imperativnog API-ja kroz forwardRef, useDebugValue za bolji prikaz u DevTools-u, useId za stabilne ID-eve, te noviji hookovi kao useTransition i useDeferredValue (prioritizacija/render sa odlaganjem), useSyncExternalStore (stabilna pretplata na eksterne store-ove) i useInsertionEffect (ubacivanje stilova pre layout-a). Pravila korišćenja su prosta ali stroga: hookove pozivaj samo na vrhu funkcijske komponente ili unutar drugih hookova (ne u if/for blokovima i ne uslovno), i uvek u istom redosledu između rendera; time React može tačno da uparuje interno stanje s tvojim pozivima.

##### Korisnički definisana kuka

Ovo je prilagođeni React hook (useSalesData) koji enkapsulira kompletan tok “učitaj sale sa servera → zapamti u state → izloži API komponentama”. Hook prima opcioni objekat sa svojstvom auto (podrazumevano true). Kada je auto=true, hook će odmah po montiranju komponente sam da pozove učitavanje; ako je false, kontrolu osvežavanja prepuštaš ručnoj funkciji reload.

Unutra drži tri stanja: items (niz sala), loading (da li traje poziv), i serverMsg (user-friendly poruka o grešci ili statusu). Funkcija reload je memoizovana pomoću useCallback tako da referenca ostaje stabilna između rendera. Ona resetuje poruku, postavlja loading=true, šalje GET /sale preko tvog api klijenta i rezultat smešta u items. Odgovor parsira fleksibilno (res.data?.data || res.data || []) kako bi pokrio i slučaj Laravel Resource kolekcije (gde je payload pod data) i “plain” JSON. U slučaju greške, postavlja čitljivu poruku (npr. problem sa autentikacijom/ulogom), a u finally sekciji vraća loading=false, čime su svi izlazi uvek konzistentni.

useEffect služi da, ako je auto uključeno, izvrši inicijalno učitavanje na montiranje komponente. Ima “mounted” zastavicu kao zaštitu od eventualnog setovanja state-a nakon unmount-a. Zavisan je od auto i reload, pa će ponovo pokrenuti učitavanje ako se ti parametri promene.

Na kraju, hook vraća objekat: { items, setItems, loading, serverMsg, setServerMsg, reload }. Tako componenta koja ga koristi dobija i podatke i “kontrole”: može da prikaže spiner dok je loading, da ispiše serverMsg ako je došlo do problema, da ručno pozove reload() kada želi da osveži spisak, pa čak i da radi optimističke izmene nad items preko setItems pre nego što backend potvrdi.

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

import api from "../api/client";

/\*\*

 \* Učitavanje liste sala (GET /api/sale) sa state-om i retry-jem.

 \* Vraća: items, setItems, loading, serverMsg, setServerMsg, reload

 \*/

export default function useSalesData({ auto = true } = {}) {

  const [items, setItems] = useState([]);

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

  const [serverMsg, setServerMsg] = useState("");

  const reload = useCallback(async () => {

    setLoading(true);

    setServerMsg("");

    try {

      const res = await api.get("/sale"); // GET /api/sale

      setItems(res.data?.data || res.data || []);

    } catch (err) {

      setServerMsg("Ne mogu da učitam sale. Proveri prijavu i ulogu (administrator/menadžer).");

    } finally {

      setLoading(false);

    }

  }, []);

  useEffect(() => {

    if (!auto) return;

    let mounted = true;

    (async () => {

      await reload();

      if (!mounted) return;

    })();

    return () => { mounted = false; };

  }, [auto, reload]);

  return { items, setItems, loading, serverMsg, setServerMsg, reload };

}

#### React Komponente

Komponente su temelj React aplikacija: samostalni, višekratno upotrebljivi delovi korisničkog interfejsa sa sopstvenom logikom i prikazom. Umesto jedne velike stranice, UI se razlaže na manje „kockice” – npr. Header, Footer, UserList, UserListItem – koje se uklapaju u celinu. Ovakva modularnost olakšava održavanje, jer se svaka komponenta razvija, testira i menja nezavisno od ostatka sistema.

U praksi, komponenta je najčešće JavaScript funkcija koja vraća JSX, opis izgleda koji se pretvara u stvarne DOM elemente. Komponente primaju ulazne podatke kroz props kako bi roditelj mogao da utiče na njihov izgled i ponašanje, a sopstveno stanje (state) čuvaju unutar sebe – promena state-a automatski izaziva ponovni render onog dela interfejsa na koji utiče.

Komponente se prirodno ugnežđuju, pa jedna može da renderuje druge i tako nastaje stablo koje odražava strukturu ekrana. Ista komponenta može se koristiti na više mesta, čime se smanjuje dupliranje koda i čuva konzistentnost prikaza i ponašanja.

Savremeni React favorizuje funkcionalne komponente sa Hooks-ima zbog jednostavnije sintakse i jasnog modela upravljanja stanjem i efektima, iako je korisno poznavati i klasni pristup koji je prisutan u starijim kodnim bazama. Suština ostaje ista: komponenta enkapsulira i prikaz i logiku određenog dela UI-a, pa o interfejsu možemo razmišljati segment po segment.

##### Registracija korisnika

Ova React komponenta predstavlja stranicu za registraciju i koristi React Router-ov useNavigate kako bi nakon uspešne registracije preusmerila korisnika na rutu prosleđenu kroz redirectTo (podrazumevano „/sale“). Iz konteksta autentikacije preko useAuth uzima se login funkcija kako bi se nakon uspešnog odgovora sa servera ažuriralo globalno auth stanje i lokalna memorija. Stilski sloj je izdvojen u auth.css, a u markupu postoje dekorativni „blob“ elementi smešteni u kontejner sa aria-hidden="true" i glavni sadržaj obeležen role="main" radi pristupačnosti.

Interno stanje forme čuva se u objektu form sa poljima name, email, password i password\_confirmation, dok su errors, serverMsg i dva indikatora učitavanja (loading i loadingRandom) zasebna „piece-of-state“ polja. Promena bilo kog polja ide kroz handleChange, koji sinhronizuje vrednost, briše eventualnu grešku za to polje i čisti serversku poruku. Tu je i pomoćna funkcija firstError koja iz Laravel-ovog formata grešaka (niz stringova po polju) uzima samo prvi string kako bi se jasno prikazala prva relevantna poruka u UI-ju. Klijentska validacija u validate proverava da li su polja popunjena, da li lozinka ima bar 8 karaktera i da li se lozinke poklapaju; greške se upisuju u errors, a funkcija vraća true samo ako nema nijedne greške.

Komponenta sadrži opcioni „quality-of-life“ dodatak fillWithRandom, koji preko fetch poziva prema https://randomuser.me/api/ nasumično popunjava ime i email i čuva sličicu u stanju avatar. Avatar je isključivo dekorativan i ne šalje se backendu. Za ovaj proces koristi se zaseban „loading“ indikator (loadingRandom) i korisniku se kroz serverMsg daje informacija o uspehu ili neuspehu učitavanja. Glavno slanje forme je u handleSubmit: zaustavlja se podrazumevano ponašanje, pokreće se klijentska validacija i, ako je sve u redu, šalje se POST ka /registracija putem našeg api klijenta. Očekuje se objekat sa user, token i message; nakon uspeha poziva se login({ user, token }) da bi se upisao auth kontekst, prikaže se poruka i izvrši navigacija na redirectTo. U slučaju greške 422, serverske validacione poruke se mapiraju u errors i prikazuje se poruka da se polja provere; za ostale greške prikazuje se generična poruka. Tokom slanja forme koristi se loading da bi se onemogućili dugmići i sprečilo višestruko slanje.

Interfejs forme je složen iz naših reusable komponenti kako bi kod bio čist, dosledan i lak za održavanje: ceo sadržaj je u FormCard koji prima onSubmit i prikazuje „eyebrow“, naslov i podnaslov. Za povratne poruke koristi se Alert koji dobija prop kind dinamički, tako da detektuje greške po tekstu poruke i po potrebi prikaže varijantu „danger“, a u suprotnom „ok“. Unosni elementi su realizovani kroz Input za „Ime i prezime“ i „Email“, kao i PasswordInput za lozinku i potvrdu lozinke; svaka od ovih komponenti dobija aktuelnu vrednost, onChange, required, autoComplete i odgovarajuću poruku greške preko error={firstError(errors.polje)} kako bi validacija bila prikazana tačno pored polja. Akcije su izvedene pomoću Button komponente: „ghost“ dugme koje poziva fillWithRandom i „primary“ dugme za slanje forme, pri čemu oba reaguju na loading indikatore i menjaju labelu („Učitavam…“ / „Kreiranje naloga…“) radi jasnog feedback-a. Ako avatar postoji, prikazuje se mali blok iznad polja sa sličicom i napomenom da je to samo primer. N

Kod ove komponente je dat ispod.

// src/pages/RegisterPage.jsx

import React, { useState } from "react";

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

import api from "../api/client";

import "./auth.css";

import Input from "../components/ui/Input";

import FormCard from "../components/ui/FormCard";

import Alert from "../components/ui/Alert";

import PasswordInput from "../components/ui/PasswordInput";

import Button from "../components/ui/Button";

import { useAuth } from "../context/AuthContext";

export default function RegisterPage({ redirectTo = "/sale" }) {

  const navigate = useNavigate();

  const { login } = useAuth();

  const [form, setForm] = useState({

    name: "",

    email: "",

    password: "",

    password\_confirmation: "",

  });

  const [errors, setErrors] = useState({});

  const [serverMsg, setServerMsg] = useState("");

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

  const [loadingRandom, setLoadingRandom] = useState(false);

  const [avatar, setAvatar] = useState(null); // dekorativno, ne šaljemo backendu

  const firstError = (v) => (Array.isArray(v) ? v[0] : v);

  const handleChange = (e) => {

    setForm((f) => ({ ...f, [e.target.name]: e.target.value }));

    setErrors((prev) => ({ ...prev, [e.target.name]: null }));

    setServerMsg("");

  };

  const validate = () => {

    const e = {};

    if (!form.name?.trim()) e.name = "Unesite ime i prezime.";

    if (!form.email?.trim()) e.email = "Unesite email.";

    if (!form.password?.trim()) e.password = "Unesite lozinku.";

    if (form.password && form.password.length < 8) e.password = "Lozinka mora imati najmanje 8 karaktera.";

    if (!form.password\_confirmation?.trim()) e.password\_confirmation = "Potvrdite lozinku.";

    if (form.password?.trim() && form.password\_confirmation?.trim() && form.password !== form.password\_confirmation) {

      e.password\_confirmation = "Lozinke se ne poklapaju.";

    }

    setErrors(e);

    return Object.keys(e).length === 0;

  };

  async function fillWithRandom() {

    setLoadingRandom(true);

    setServerMsg("");

    try {

      const r = await fetch("https://randomuser.me/api/?nat=gb,us,fr,de,au");

      const d = await r.json();

      const u = d?.results?.[0];

      if (!u) throw new Error("No user");

      setForm((f) => ({

        ...f,

        name: `${u.name.first} ${u.name.last}`,

        email: u.email,

      }));

      setAvatar(u.picture?.thumbnail || u.picture?.medium || null);

      setServerMsg("Popunio sam ime i email nasumičnim korisnikom.");

    } catch {

      setServerMsg("Nije moguće preuzeti nasumičnog korisnika.");

    } finally {

      setLoadingRandom(false);

    }

  }

  const handleSubmit = async (e) => {

    e.preventDefault();

    if (!validate()) return;

    setLoading(true);

    setServerMsg("");

    try {

      const res = await api.post("/registracija", {

        name: form.name,

        email: form.email,

        password: form.password,

        password\_confirmation: form.password\_confirmation,

      });

      const { user, token, message } = res.data || {};

      // upiši u AuthContext + localStorage

      login({ user, token });

      setServerMsg(message || "Registracija je uspešna!");

      navigate(redirectTo, { replace: true }); // -> /sale

    } catch (err) {

      if (err?.response?.status === 422) {

        setErrors(err.response.data.errors || {});

        setServerMsg("Proverite polja i pokušajte ponovo.");

      } else {

        setServerMsg("Došlo je do greške. Pokušajte ponovo.");

      }

    } finally {

      setLoading(false);

    }

  };

  return (

    <div className="lp">

      {/\* Pozadinski blubovi \*/}

      <div aria-hidden="true" className="lp-bg">

        <div className="blob blob-1" />

        <div className="blob blob-2" />

        <div className="blob blob-3" />

      </div>

      <main className="auth" role="main">

        <FormCard

          onSubmit={handleSubmit}

          eyebrow="Registracija"

          title="Kreiraj nalog"

          subtitle="Popuni podatke da započneš sa radom u aplikaciji."

        >

          {serverMsg && (

            <Alert kind={serverMsg.toLowerCase().includes("greš") || serverMsg.toLowerCase().includes("pogreš") ? "danger" : "ok"}>

              {serverMsg}

            </Alert>

          )}

          {/\* Dekorativni preview avatara (ako ga imamo iz random user-a) \*/}

          {avatar && (

            <div style={{ display: "flex", alignItems: "center", gap: 10, marginBottom: 8 }}>

              <img

                src={avatar}

                alt="Avatar podrazumevanog korisnika"

                style={{ width: 40, height: 40, borderRadius: 12, border: "1px solid var(--border)" }}

              />

              <span className="muted" style={{ fontSize: 13 }}>Nasumičan avatar (nije obavezan, služi kao primer)</span>

            </div>

          )}

          <Input

            label="Ime i prezime"

            name="name"

            type="text"

            placeholder="Petar Petrović"

            value={form.name}

            onChange={handleChange}

            autoComplete="name"

            required

            error={firstError(errors.name)}

          />

          <Input

            label="Email"

            name="email"

            type="email"

            placeholder="you@example.com"

            value={form.email}

            onChange={handleChange}

            autoComplete="email"

            required

            error={firstError(errors.email)}

          />

          <PasswordInput

            label="Lozinka"

            name="password"

            value={form.password}

            onChange={handleChange}

            placeholder="Najmanje 8 karaktera"

            autoComplete="new-password"

            required

            error={firstError(errors.password)}

          />

          <PasswordInput

            label="Potvrda lozinke"

            name="password\_confirmation"

            value={form.password\_confirmation}

            onChange={handleChange}

            placeholder="Ponovite lozinku"

            autoComplete="new-password"

            required

            error={firstError(errors.password\_confirmation)}

          />

          <div style={{ display: "flex", gap: 10, marginTop: 4 }}>

            <Button type="button" variant="ghost" onClick={fillWithRandom} disabled={loading || loadingRandom}>

              {loadingRandom ? "Učitavam..." : " Popuni nasumičnim korisnikom"}

            </Button>

            <Button type="submit" variant="primary" fullWidth disabled={loading}>

              {loading ? "Kreiranje naloga..." : "Registruj se"}

            </Button>

          </div>

          <p className="meta">

            Već imaš nalog? <a href="/login">Prijavi se</a>

          </p>

        </FormCard>

      </main>

    </div>

  );

}

##### Prijava korisnika

Slično kao i kod komponente za registraciju, stanje ove forme je minimalno i sastoji se od polja email i password, uz prateća stanja errors, serverMsg i loading. Funkcija handleChange sinhronizuje unos u form, resetuje eventualnu grešku za odgovarajuće polje i čisti serverSKU poruku nakon svake promene, čime se obezbeđuje da korisnik ne vidi zastarele poruke. Brza klijentska validacija u validate proverava da su oba polja popunjena i puni objekat errors ako nešto nedostaje; povratna vrednost funkcije je istinita samo kada nema grešaka.

Slanje podataka obrađuje handleSubmit, koja zaustavlja podrazumevano ponašanje forme, pokreće validaciju i, ukoliko prođe, postavlja loading i poziva api.post("/prijava") sa emailom i lozinkom. Odgovor servera očekuje user, token i eventualnu message poruku; nakon uspeha pozivamo login({ user, token }) da bismo ažurirali autentikaciju na nivou aplikacije, prikazujemo poruku o uspešnoj prijavi i pozivamo navigate sa replace: true kako bismo prešli na ciljnu rutu bez mogućnosti povratka na login pritiskom na Back. Obrada grešaka je diferencirana: za 422 mapiramo validacione greške sa servera u errors i dajemo poruku da se polja provere, za 401 prikazujemo poruku o pogrešnim kredencijalima iz serverovog odgovora ako postoji, a za sve ostalo prikazujemo generičnu poruku o grešci. Uvek u finally blokU gasimo loading kako bismo vratili interfejs u normalno stanje i sprečili dvostruko slanje.

Korisnički interfejs sastavljen je iz naših reusable komponenti radi konzistentnog dizajna i lakšeg održavanja. FormCard preuzima onSubmit i renderuje „eyebrow“, naslov i podnaslov, pa dobijamo uredan okvir za formu bez suvišnog HTML-a. Povratne poruke prikazujemo kroz Alert, kome dinamički biramo kind na osnovu teksta: ako sadrži trag za grešku (npr. „greš“/„pogreš“), prikazuje se u „danger“ varijanti, u suprotnom u „ok“ varijanti. Za unos koristimo Input za email i PasswordInput za lozinku, oba povezana na form i errors, sa odgovarajućim autoComplete vrednostima kako bismo poboljšali UX i sigurnost. Akcioni element je Button u „primary“ varijanti, koji po potrebi prikazuje „Prijavljivanje…“ dok je loading aktivan i onemogućava dodatne klikove. Na dnu karte je kratka meta napomena sa linkom ka registraciji za korisnike bez naloga. Ovakva struktura sa FormCard, Input, PasswordInput, Alert i Button osigurava dosledan izgled i ponašanje, a useAuth i useNavigate daju jasan tok od unosa do autentikacije i preusmeravanja.

import React, { useState } from "react";

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

import api from "../api/client";

import Input from "../components/ui/Input";

import FormCard from "../components/ui/FormCard";

import Alert from "../components/ui/Alert";

import PasswordInput from "../components/ui/PasswordInput";

import Button from "../components/ui/Button";

import { useAuth } from "../context/AuthContext";

import "./auth.css";

export default function LoginPage({ redirectTo = "/sale" }) {

  const navigate = useNavigate();

  const { login } = useAuth();

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

  const [errors, setErrors] = useState({});

  const [serverMsg, setServerMsg] = useState("");

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

  const handleChange = (e) => {

    setForm((f) => ({ ...f, [e.target.name]: e.target.value }));

    setErrors((prev) => ({ ...prev, [e.target.name]: null }));

    setServerMsg("");

  };

  const validate = () => {

    const e = {};

    if (!form.email?.trim()) e.email = "Unesite email.";

    if (!form.password?.trim()) e.password = "Unesite lozinku.";

    setErrors(e);

    return Object.keys(e).length === 0;

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    if (!validate()) return;

    setLoading(true);

    setServerMsg("");

    try {

      const res = await api.post("/prijava", {

        email: form.email,

        password: form.password,

      });

      const { user, token, message } = res.data || {};

      login({ user, token }); // snimi u context + localStorage

      setServerMsg(message || "Prijava je uspešna.");

      navigate(redirectTo, { replace: true });

    } catch (err) {

      if (err?.response?.status === 422) {

        setErrors(err.response.data.errors || {});

        setServerMsg("Proverite polja i pokušajte ponovo.");

      } else if (err?.response?.status === 401) {

        setServerMsg(err.response.data?.message || "Pogrešni podaci za prijavu.");

      } else {

        setServerMsg("Došlo je do greške. Pokušajte ponovo.");

      }

    } finally {

      setLoading(false);

    }

  };

  return (

    <div className="lp">

      <div aria-hidden="true" className="lp-bg">

        <div className="blob blob-1" />

        <div className="blob blob-2" />

        <div className="blob blob-3" />

      </div>

      <main className="auth" role="main">

        <FormCard

          onSubmit={handleSubmit}

          eyebrow="Prijava"

          title="Dobrodošli nazad"

          subtitle="Ulogujte se da nastavite sa rezervacijama, salama i preporukama."

        >

          {serverMsg && (

            <Alert kind={serverMsg.toLowerCase().includes("greš") || serverMsg.toLowerCase().includes("pogreš") ? "danger" : "ok"}>

              {serverMsg}

            </Alert>

          )}

          <Input

            label="Email"

            name="email"

            type="email"

            placeholder="you@example.com"

            value={form.email}

            onChange={handleChange}

            autoComplete="email"

            required

            error={errors.email}

          />

          <PasswordInput

            label="Lozinka"

            name="password"

            value={form.password}

            onChange={handleChange}

            placeholder="Unesite lozinku"

            autoComplete="current-password"

            required

            error={errors.password}

          />

          <Button type="submit" variant="primary" fullWidth disabled={loading}>

            {loading ? "Prijavljivanje..." : "Prijavi se"}

          </Button>

          <p className="meta">

            Nemaš nalog? <a href="/registracija">Registruj se</a>

          </p>

        </FormCard>

      </main>

    </div>

  );

}

##### Katalog sala

Ova React komponenta prikazuje “katalog sala” sa filtrima, sortiranjem, paginacijom, preporukama i brzim kreiranjem rezervacije. Na vrhu održava interni state: listu sala, indikatore učitavanja/poruke, vrednosti filtera (tekstualna pretraga, tip sale, kapacitet od/do), vremenske preferencije (datum, vreme od–do), tip događaja i broj učesnika. Tu su i kontrola sortiranja (kolona + smer), paginacije (stranica, veličina strane), kao i stanje za modal za rezervaciju i evidenciju “preporuka” snimljenih u bazu.

Podaci o salama se učitavaju iz back-enda pozivom GET /sale preko zajedničkog axios klijenta (api). Učitavanje je upakovano u load (memoizovan useCallback) i poziva se pri montiranju (useEffect). U slučaju greške prikazuje se poruka korisniku. Nakon što se sale nađu u state-u, komponenta iz njih dinamički pravi listu dostupnih “tipova sale” za padajući meni.

Prikaz tabela je uvek izveden nad “obrađenom” listom: prvo se primeni višekriterijumsko filtriranje (tekst pretražuje naziv/opis/kapacitet/status; tip sužava na odabranu kategoriju; kapacitet po opsegu), zatim sortiranje po nazivu ili kapacitetu (uz prebacivanje smera na klik zaglavlja). Rezultat se potom deli na strane: izračunava se ukupan broj, broj strana i opseg za trenutnu stranicu, pa se renderuje samo odgovarajući isečak. Kontrole za paginaciju nude brzo skakanje na prvu/poslednju, kao i kretanje unazad/napred uz ažuriranje stanja.

Komponenta sadrži i mehanizam “preporuka” zasnovan na unetim preferencijama. Iz trenutnog skupa sala prvo se izbacuju one koje ne upadaju u ogradu kapaciteta (od/do). Zatim se svakoj preostaloj sali dodeljuje “score” prema nizu heuristika: poklapanje tipa sale sa preferencijom, naznačen tip događaja (heuristično poređenje sa tipom sale), blizina željenom broju učesnika, aktivnost sale, kao i činjenica da je termin popunjen (datum i ispravan interval vremena). Ovaj skor se sabira i prema njemu se formira rang lista; na ekranu se prikazuje prvih pet predloga sa brzim akcijama.

Za svaku preporučenu (ili bilo koju iz tabele) salu korisnik može odmah otvoriti modal za rezervaciju. Klik na “Rezerviši” popunjava formu početnim vrednostima (sala, termin, tip događaja), a ReservationForm obavlja validaciju i slanje rezerva u back-end; po uspehu modal se zatvara, lista osvežava i prikazuje se kratka poruka. Druga akcija je “Sačuvaj preporuku” koja šalje POST /preporuke sa salom i preferiranim terminom/tipom događaja. Komponenta obeležava salu kao “sačuvanu” (lokalni Set) kako dugme ne bi bilo dostupno dva puta; tokom slanja prikazuje se “Čuvam…”, a po uspehu “Sačuvano ✓”.

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

import api from "../api/client";

import ReservationForm from "../components/ReservationForm";

import "./sale.css";

/\* ---------------- helpers ---------------- \*/

function toMin(t) { const [h,m]=String(t||"").split(":"); return (+h)\*60+(+m||0); }

/\* ---------------- component ---------------- \*/

export default function SaleCatalog() {

  const [sale, setSale]   = useState([]);

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

  const [msg, setMsg] = useState("");

  // UI state — filteri / preferencije

  const [query, setQuery]   = useState("");

  const [tip, setTip]       = useState("all");

  const [capMin, setCapMin] = useState("");

  const [capMax, setCapMax] = useState("");

  const [date, setDate]     = useState("");

  const [from, setFrom]     = useState("08:00");

  const [to, setTo]         = useState("10:00");

  const [kind, setKind]     = useState("sastanak");

  const [people, setPeople] = useState("");

  // sort

  const [sortBy, setSortBy]   = useState("naziv");

  const [sortDir, setSortDir] = useState("asc");

  // paginacija

  const [page, setPage]       = useState(1);

  const [perPage, setPerPage] = useState(10);

  // modal rezervacije

  const [openRes, setOpenRes] = useState(null); // { sala, date, from, to, tip\_dogadjaja }

  // lokano stanje za snimanje preporuka

  const [savingId, setSavingId] = useState(null);     // id sale koja se trenutno snima

  const [savedIds, setSavedIds] = useState(new Set()); // set sala\_id koje smo uspešno snimili

  // Poeni za preporuke

  const WEIGHTS = {

    roomTypeMatch: 3,

    eventKindHint: 2,

    capacityProximity: 5,

    activeRoom: 0.5,

    timePrefGiven: 0.5,

  };

  /\* ------------ učitavanje podataka ------------ \*/

  const load = useCallback(async () => {

    setLoading(true); setMsg("");

    try{

      const sRes = await api.get("/sale");

      setSale(sRes.data?.data || sRes.data || []);

    }catch(e){

      setMsg("Ne mogu da učitam sale. Proveri prijavu i prava.");

    }finally{ setLoading(false); }

  }, []);

  useEffect(()=>{ load(); },[load]);

  /\* ------------ dinamički tipovi ------------ \*/

  const tipOptions = useMemo(

    () => ["all", ...Array.from(new Set((sale||[]).map(s=>String(s.tip||"")).filter(Boolean)))],

    [sale]

  );

  /\* ------------ filtracija za tabelu ------------ \*/

  const filtered = useMemo(()=>{

    let list = Array.isArray(sale) ? [...sale] : [];

    const q = query.trim().toLowerCase();

    if(q){

      list = list.filter(s=>{

        const hay = [

          s.naziv, s.tip, s.opis, String(s.kapacitet),

          s.status ? "aktivna" : "neaktivna"

        ].join(" ").toLowerCase();

        return hay.includes(q);

      });

    }

    if(tip !== "all") list = list.filter(s => String(s.tip||"") === tip);

    const min = capMin!=="" ? Number(capMin) : null;

    const max = capMax!=="" ? Number(capMax) : null;

    if(min!==null) list = list.filter(s => Number(s.kapacitet||0) >= min);

    if(max!==null) list = list.filter(s => Number(s.kapacitet||0) <= max);

    // Sort

    list.sort((a,b)=>{

      const dir = sortDir==="asc" ? 1 : -1;

      const A = a[sortBy], B = b[sortBy];

      if(typeof A === "string" || typeof B === "string"){

        return String(A||"").localeCompare(String(B||""), "sr", {sensitivity:"base"}) \* dir;

      }

      return ((Number(A)||0) - (Number(B)||0)) \* dir;

    });

    return list;

  },[sale, query, tip, capMin, capMax, sortBy, sortDir]);

  /\* ------------ paginacija ------------ \*/

  const total = filtered.length;

  const totalPages = Math.max(1, Math.ceil(total / perPage));

  const cur = Math.min(page, totalPages);

  const start = (cur-1)\*perPage, end = Math.min(start+perPage, total);

  const pageItems = filtered.slice(start, end);

  function toggleSort(col){

    if(sortBy===col) setSortDir(d=>d==="asc"?"desc":"asc");

    else { setSortBy(col); setSortDir(col==="kapacitet"?"desc":"asc"); }

  }

  function resetFilters(){

    setQuery(""); setTip("all"); setCapMin(""); setCapMax("");

    setDate(""); setFrom("08:00"); setTo("10:00"); setKind("sastanak"); setPeople("");

    setSortBy("naziv"); setSortDir("asc"); setPage(1);

  }

  /\* ------------ preporuke (po preferencijama + kapacitet kao ograda) ------------ \*/

  const scoredSuggestions = useMemo(()=>{

    const min = capMin!=="" ? Number(capMin) : null;

    const max = capMax!=="" ? Number(capMax) : null;

    const pool = (sale||[]).filter(s => {

      const cap = Number(s.kapacitet)||0;

      if (min!==null && cap < min) return false;

      if (max!==null && cap > max) return false;

      return true;

    });

    const target = (() => {

      const p = Number(people||0);

      if (p > 0) return p;

      if (min!==null && max!==null) return (min + max)/2;

      if (min!==null) return min;

      if (max!==null) return max;

      return 0;

    })();

    const hasTime = Boolean(date && from && to && toMin(to) > toMin(from));

    const rankList = pool.map(s => {

      let score = 0;

      if (tip !== "all" && String(s.tip||"") === tip) score += WEIGHTS.roomTypeMatch;

      if (kind && String(s.tip||"").toLowerCase().includes(kind.toLowerCase())) score += WEIGHTS.eventKindHint;

      if (target > 0) {

        const delta = Math.abs((Number(s.kapacitet)||0) - target);

        score += Math.max(0, WEIGHTS.capacityProximity - delta / Math.max(target, 1));

      }

      if (s.status) score += WEIGHTS.activeRoom;

      if (hasTime) score += WEIGHTS.timePrefGiven;

      return { sala: s, score };

    });

    rankList.sort((a,b)=> b.score - a.score);

    return (rankList.length ? rankList : (sale||[]).map(s=>({sala:s,score:0}))).slice(0,5);

  }, [sale, tip, kind, people, capMin, capMax, date, from, to]);

  /\* ------------ snimi preporuku u bazu (POST /preporuke) ------------ \*/

  async function saveRecommendation(sala) {

    // backend očekuje: sala\_id, datum, vreme\_od, vreme\_do, tip\_dogadjaja

    if (!sala?.id) return;

    try {

      setSavingId(sala.id);

      setMsg("");

      const payload = {

        sala\_id: sala.id,

        datum: date || new Date().toISOString().slice(0,10), // ako korisnik nije uneo datum, danas

        vreme\_od: from || "08:00",

        vreme\_do: to   || "09:00",

        tip\_dogadjaja: kind || "dogadjaj",

      };

      const res = await api.post("/preporuke", payload);

      // obeleži snimljeno

      setSavedIds(prev => new Set(prev).add(sala.id));

      setMsg("Preporuka je sačuvana.");

      return res?.data;

    } catch (e) {

      setMsg(e?.response?.data?.message || "Ne mogu da sačuvam preporuku.");

    } finally {

      setSavingId(null);

    }

  }

  /\* ------------ auto rezerviši top predlog (opciono) ------------ \*/

  function autoReserveTop() {

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

    const top = scoredSuggestions[0].sala;

    setOpenRes({

      sala: top,

      date: date || "",

      from,

      to,

      tip\_dogadjaja: kind

    });

  }

  return (

    <main className="sales" role="main">

      <div className="sales-head">

        <div>

          <p className="eyebrow">Dostupne prostorije</p>

          <h1>Sale — pregled i rezervacija</h1>

          <p className="muted">

            Unesi svoje preferencije (tip prostorije, tip događaja, broj učesnika i termin).

            Preporuke poštuju <strong>Kapacitet (od/do)</strong> kao ogradu i zatim rangiraju najbolje opcije.

          </p>

        </div>

        <div className="sales-actions">

          <button className="btn btn--ghost" onClick={load}>Osveži</button>

          <button className="btn btn--ghost" onClick={resetFilters}>Reset filtera</button>

        </div>

      </div>

      {/\* FILTERI / PREFERENCIJE \*/}

      <section className="filters" aria-label="Filteri i preferencije">

        <div className="filters-grid">

          <div className="filter">

            <label className="filter-label">Pretraga</label>

            <input value={query} onChange={e=>setQuery(e.target.value)} placeholder="Naziv, opis, status…" />

          </div>

          <div className="filter">

            <label className="filter-label">Tip prostorije (preferencija)</label>

            <select value={tip} onChange={e=>setTip(e.target.value)}>

              {tipOptions.map(t => <option key={t} value={t}>{t==="all"?"Svi tipovi":t}</option>)}

            </select>

          </div>

          <div className="filter">

            <label className="filter-label">Kapacitet (od)</label>

            <input type="number" min="1" value={capMin} onChange={e=>setCapMin(e.target.value)} placeholder="npr. 50" />

          </div>

          <div className="filter">

            <label className="filter-label">Kapacitet (do)</label>

            <input type="number" min="1" value={capMax} onChange={e=>setCapMax(e.target.value)} placeholder="npr. 80" />

          </div>

          <div className="filter">

            <label className="filter-label">Datum (preferencija)</label>

            <input type="date" value={date} onChange={e=>setDate(e.target.value)} />

          </div>

          <div className="filter">

            <label className="filter-label">Vreme (od–do) (preferencija)</label>

            <div className="field-row">

              <input type="time" value={from} onChange={e=>setFrom(e.target.value)} />

              <span>–</span>

              <input type="time" value={to} onChange={e=>setTo(e.target.value)} />

            </div>

          </div>

          <div className="filter">

            <label className="filter-label">Tip događaja (preferencija)</label>

            <input value={kind} onChange={e=>setKind(e.target.value)} placeholder="npr. sastanak, radionica…" />

          </div>

          <div className="filter">

            <label className="filter-label">Broj učesnika (preferencija)</label>

            <input type="number" min="1" value={people} onChange={e=>setPeople(e.target.value)} placeholder="npr. 60" />

          </div>

          <div className="filter">

            <label className="filter-label">Sortiraj</label>

            <select value={sortBy} onChange={e=>setSortBy(e.target.value)}>

              <option value="naziv">Naziv</option>

              <option value="kapacitet">Kapacitet</option>

            </select>

          </div>

          <div className="filter">

            <label className="filter-label">Smer</label>

            <select value={sortDir} onChange={e=>setSortDir(e.target.value)}>

              <option value="asc">Rastuće</option>

              <option value="desc">Opadajuće</option>

            </select>

          </div>

        </div>

      </section>

      {msg && <div className="alert">{msg}</div>}

      {loading && <div className="alert">Učitavanje…</div>}

      {/\* PREPORUKE + Sačuvaj u bazu \*/}

      {(scoredSuggestions.length>0) && (

        <div className="card" style={{margin:"10px 0 14px", padding:"10px 12px"}}>

          <div style={{display:"flex",justifyContent:"space-between",alignItems:"center"}}>

            <strong>Predlog za vas (po unetim preferencijama):</strong>

            <button className="btn btn--ghost" onClick={autoReserveTop}>

              Rezerviši top predlog

            </button>

          </div>

          <ul style={{margin:"8px 0 0 18px"}}>

            {scoredSuggestions.map(({ sala: s, score })=>(

              <li key={s.id} style={{marginBottom:8}}>

                <div style={{display:"flex", gap:8, alignItems:"center", flexWrap:"wrap"}}>

                  <span>{s.naziv} — {s.tip} · kap. {s.kapacitet}</span>

                  <span className="muted">poeni: {score.toFixed(2)}</span>

                  {(date && from && to) && (

                    <span className="muted"> · termin: {date} {from}–{to}</span>

                  )}

                  <button

                    className="btn btn--ghost"

                    onClick={()=>setOpenRes({sala:s, date: date || "", from, to, tip\_dogadjaja: kind})}

                  >

                    Rezerviši

                  </button>

                  <button

                    className="btn"

                    disabled={savingId===s.id || savedIds.has(s.id)}

                    onClick={()=>saveRecommendation(s)}

                    title="Sačuvaj ovu preporuku u bazi"

                  >

                    {savedIds.has(s.id)

                      ? "Sačuvano ✓"

                      : (savingId===s.id ? "Čuvam…" : "Sačuvaj preporuku")}

                  </button>

                </div>

              </li>

            ))}

          </ul>

        </div>

      )}

      {/\* TABELA \*/}

      <div className="card">

        <div className="table-wrap">

          <table className="tbl">

            <thead>

              <tr>

                <th

                  className={`sortable ${sortBy==="naziv" ? `sorted ${sortDir}` : ""}`}

                  onClick={()=>toggleSort("naziv")}

                >

                  Naziv

                </th>

                <th>Tip</th>

                <th className="right sortable" onClick={()=>toggleSort("kapacitet")}>Kapacitet</th>

                <th>Status</th>

                <th>Dostupnost</th>

                <th className="right">Akcije</th>

              </tr>

            </thead>

            <tbody>

              {pageItems.length===0 ? (

                <tr><td colSpan={6} className="center muted">Nema podataka.</td></tr>

              ) : pageItems.map(s=> (

                <tr key={s.id}>

                  <td>{s.naziv}</td>

                  <td>{s.tip}</td>

                  <td className="right">{s.kapacitet}</td>

                  <td>

                    <span className={`badge ${s.status ? "badge--ok":"badge--muted"}`}>

                      {s.status ? "Aktivna":"Neaktivna"}

                    </span>

                  </td>

                  <td><span className="muted">—</span></td>

                  <td className="right">

                    <button

                      className="btn btn--ghost"

                      onClick={()=>setOpenRes({sala:s, date: date || "", from, to, tip\_dogadjaja: kind})}

                    >

                      Rezerviši

                    </button>

                    <button

                      className="btn btn--ghost"

                      disabled={savingId===s.id || savedIds.has(s.id)}

                      onClick={()=>saveRecommendation(s)}

                    >

                      {savedIds.has(s.id) ? "Sačuvano ✓" : (savingId===s.id ? "Čuvam…" : "Sačuvaj preporuku")}

                    </button>

                  </td>

                </tr>

              ))}

            </tbody>

          </table>

        </div>

      </div>

      {/\* PAGINACIJA \*/}

      <div className="pagination">

        <div className="page-size">

          <label>Prikaži:</label>

          <select value={perPage} onChange={e=>{setPerPage(+e.target.value); setPage(1);}}>

            {[5,10,20,50,100].map(n=><option key={n} value={n}>{n}/str</option>)}

          </select>

        </div>

        <div className="page-info"><span>Prikaz {total===0?0:start+1}–{end} od {total}</span></div>

        <div className="page-ctrls">

          <button className="btn btn--ghost" onClick={()=>setPage(1)} disabled={cur===1}>« Prva</button>

          <button className="btn btn--ghost" onClick={()=>setPage(p=>Math.max(1,p-1))} disabled={cur===1}>‹ Prethodna</button>

          <button className="btn btn--ghost" onClick={()=>setPage(p=>Math.min(totalPages,p+1))} disabled={cur===totalPages}>Sledeća ›</button>

          <button className="btn btn--ghost" onClick={()=>setPage(totalPages)} disabled={cur===totalPages}>Poslednja »</button>

        </div>

      </div>

      {/\* MODAL: Rezervacija \*/}

      {openRes && (

        <div className="modal" role="dialog" aria-modal="true">

          <div className="modal-card">

            <ReservationForm

              initial={{

                sala\_id: openRes.sala.id,

                datum: openRes.date || "",

                vreme\_od: openRes.from || "",

                vreme\_do: openRes.to || "",

                tip\_dogadjaja: openRes.tip\_dogadjaja || "",

                status: "pending"

              }}

              salaName={openRes.sala.naziv}

              onCancel={()=>setOpenRes(null)}

              onSuccess={()=>{

                setOpenRes(null);

                load();

                setMsg("Rezervacija je uspešno poslata.");

              }}

            />

          </div>

        </div>

      )}

    </main>

  );

}

##### Raspored sala

Ovaj par komponenti zajedno pravi interaktivnu “mapu sprata” sa salama, uz mogućnost rezervacije i (samo za administratore) uređivanje rasporeda „drag & drop“. Spoljašnja komponenta FloorPlan vodi poslovnu logiku i dozvole, dok FloorMap brine o vizuelnom prikazu, mreži i prevlačenju soba.

FloorPlan prvo čita ulogu iz useAuth() i na osnovu toga određuje da li korisnik sme da uređuje raspored (canEdit). Zatim drži state sa salama, rezervacijama, filterima (sprat, datum, vreme, veličina mrežne ćelije), porukama i modalom za rezervaciju. Podaci se učitavaju iz back-enda (/sale i /rezervacije) u load() i pozivaju pri montiranju. Ako korisnik nije admin, komponenta dodatno obezbeđuje da je “edit mode” isključen (guard i u UI-ju i u handlerima). Opcije sprata se dinamički izvlače iz preuzetih sala, dok se dostupnost pojedine sale u terminu računa lokalno – za izabrani datum filtriraju se rezervacije te sale i proverava preklapanje intervala (“overlaps”). Kada admin menja raspored, saveLayout radi optimističko ažuriranje na UI-ju i šalje PATCH /sale/{id}/layout; u slučaju greške vraća stare podatke i prikaže poruku. U normalnom režimu (bez uređivanja), klik na salu otvara modal ReservationForm sa popunjenim poljima (sala, datum i vreme); po uspešnom slanju modal se zatvara i lista osvežava. Time FloorPlan objedinjuje: dozvole, učitavanje, filtriranje po spratu, validaciju dostupnosti i orkestraciju rezervacionog modala.

FloorMap je čisti, reaktivni prikaz spratova i soba, sa mrežom i DnD mehanikom. Sobe se grupišu po spratu (byFloor), a za svaki sprat se izračuna potrebna “ploča” (broj kolona i redova) na osnovu pozicija i dimenzija soba kako bi platno automatski obuhvatilo sve elemente. Vizuelno, mreža je crtana tankim linijama, a svaka soba je apsolutno pozicioniran “tile” čija veličina i položaj potiču iz layout\_x/layout\_y/layout\_w/layout\_h, skalirani cellSize-om. Stanje sobe (slobodna, zauzeta, neaktivna, nepoznata) određuje se kombinacijom status i rezultata isFree() koji komponenta dobija kao prop iz FloorPlan; to utiče na oznake i da li klik otvara modal. Prevlačenje radi preko @dnd-kit/core: ako je editable uključeno (što FloorPlan prosleđuje samo adminu u edit modu), soba postaje “draggable”, a na završetku prevlačenja se preračunaju nove layout\_x/layout\_y koordinate na mreži (uz “snap” po ćelijama i ograničenje da soba ostane unutar platna). Te vrednosti se vraćaju kroz onDrop(id, layout) nazad u FloorPlan, koji ih potom čuva na serveru. Time FloorMap ostaje generičan prikazni sloj: ne brine o dozvolama, pozivima API-ja ni modalima, već samo ispravno prikazuje i emituje događaje (klik za rezervaciju u ne-edit režimu, drag end u edit režimu).

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

import api from "../api/client";

import FloorMap from "../components/FloorMap";

import ReservationForm from "../components/ReservationForm";

import { useAuth } from "../context/AuthContext";

import "./sale.css";

/\* helpers \*/

function toMin(t) { const [h,m]=String(t||"").split(":"); return (+h)\*60+(+m||0); }

function overlaps(aFrom,aTo,bFrom,bTo){ return Math.max(aFrom,bFrom) < Math.min(aTo,bTo); }

export default function FloorPlan() {

  const { isAdmin } = useAuth();

  const canEdit = Boolean(isAdmin);

  const [sale, setSale] = useState([]);

  const [rez, setRez] = useState([]);

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

  const [msg, setMsg] = useState("");

  // UI filteri

  const [floor, setFloor] = useState("all");

  const [date, setDate] = useState("");

  const [from, setFrom] = useState("08:00");

  const [to, setTo] = useState("10:00");

  const [cellSize, setCellSize] = useState(32);

  // Edit mode (drag&drop layout) – dozvoljen samo adminu

  const [editMode, setEditMode] = useState(false);

  const [openRes, setOpenRes] = useState(null); // { sala, date, from, to }

  const load = useCallback(async () => {

    setLoading(true); setMsg("");

    try {

      const sRes = await api.get("/sale");

      setSale(sRes.data?.data || sRes.data || []);

    } catch (e) {

      setSale([]);

      setMsg("Ne mogu da učitam sale.");

    }

    try {

      const rRes = await api.get("/rezervacije");

      setRez(rRes.data?.data || rRes.data || []);

    } catch (e) {

      if (e?.response?.status === 403) setRez([]);

      else setMsg(m => m || "Ne mogu da učitam rezervacije.");

    } finally {

      setLoading(false);

    }

  }, []);

  useEffect(()=>{ load(); },[load]);

  // Ako korisnik nije admin, osiguraj da je editMode isključen

  useEffect(() => {

    if (!canEdit && editMode) setEditMode(false);

  }, [canEdit, editMode]);

  // sprat opcije

  const floorOptions = useMemo(() => {

    const set = new Set();

    (sale||[]).forEach(s => {

      const f = s.floor ?? s.sprat;

      if (f !== null && f !== undefined) set.add(Number(f));

    });

    const arr = Array.from(set).sort((a,b)=>a-b);

    return ["all", ...arr];

  }, [sale]);

  // da li je sala slobodna u terminu

  const isFree = useCallback((sala, datumISO, tFrom, tTo) => {

    if(!datumISO) return true;

    const F = toMin(tFrom), T = toMin(tTo);

    const todayRez = rez.filter(r => String(r.sala\_id) === String(sala.id) && String(r.datum) === String(datumISO));

    return todayRez.every(r => !overlaps(F, T, toMin(r.vreme\_od), toMin(r.vreme\_do)));

  }, [rez]);

  // filtriraj po izabranom spratu

  const shown = useMemo(() => {

    if (floor === "all") return sale;

    const f = Number(floor);

    return (sale||[]).filter(s => Number(s.floor ?? s.sprat ?? 0) === f);

  }, [sale, floor]);

  // snimi layout (PATCH /sale/{id}/layout) — samo admin

  const saveLayout = useCallback(async (id, layout) => {

    if (!canEdit) return; // hard guard

    setSale(prev => prev.map(s => s.id === id ? { ...s, ...layout } : s)); // optimistic

    try {

      await api.patch(`/sale/${id}/layout`, layout);

    } catch (e) {

      // vrati stanje ako backend odbije (npr. 403)

      await load();

      const err = e?.response?.data?.message || "Greška pri čuvanju rasporeda.";

      setMsg(err);

    }

  }, [canEdit, load]);

  const editable = canEdit && editMode; // jedino admin + uključen prekidač

  return (

    <main className="sales" role="main">

      <div className="sales-head">

        <div>

          <p className="eyebrow">Vizuelni raspored</p>

          <h1>Mapa sprata — sale u objektu</h1>

          <p className="muted">

            Klikom na sobu započni rezervaciju. {canEdit ? "U edit modu prevuci sobu po mreži." : "Samo admin može da uređuje raspored."}

          </p>

        </div>

        <div className="sales-actions" style={{display:"flex",gap:12,alignItems:"center"}}>

          {canEdit && (

            <label style={{display:"flex",gap:8,alignItems:"center",fontSize:14}}>

              <input

                type="checkbox"

                checked={editMode}

                onChange={(e)=>setEditMode(e.target.checked)}

              />

              Uredi raspored (drag & drop)

            </label>

          )}

          <button className="btn btn--ghost" onClick={load}>Osveži</button>

        </div>

      </div>

      {/\* FILTERI \*/}

      <section className="filters" aria-label="Filteri mape">

        <div className="filters-grid">

          <div className="filter">

            <label className="filter-label">Sprat</label>

            <select value={floor} onChange={e=>setFloor(e.target.value)}>

              {floorOptions.map(f => (

                <option key={String(f)} value={String(f)}>

                  {f === "all" ? "Svi spratovi" : `Sprat ${f}`}

                </option>

              ))}

            </select>

          </div>

          <div className="filter">

            <label className="filter-label">Datum</label>

            <input type="date" value={date} onChange={e=>setDate(e.target.value)} />

          </div>

          <div className="filter">

            <label className="filter-label">Vreme (od–do)</label>

            <div className="field-row">

              <input type="time" value={from} onChange={e=>setFrom(e.target.value)} />

              <span>–</span>

              <input type="time" value={to} onChange={e=>setTo(e.target.value)} />

            </div>

          </div>

          <div className="filter">

            <label className="filter-label">Veličina ćelije</label>

            <input

              type="range" min="20" max="60" step="4"

              value={cellSize}

              onChange={e=>setCellSize(Number(e.target.value))}

            />

          </div>

        </div>

      </section>

      {msg && <div className="alert">{msg}</div>}

      {loading && <div className="alert">Učitavanje…</div>}

      {/\* MAPA \*/}

      <FloorMap

        sale={shown}

        isFree={isFree}

        date={date}

        from={from}

        to={to}

        cellSize={cellSize}

        editable={editable}                            // ⬅️ samo admin u edit modu

        onDrop={(salaId, layout) => canEdit && saveLayout(salaId, layout)}

        onReserve={(sala) => !editable && setOpenRes({ sala, date, from, to, tip\_dogadjaja: "" })}

      />

      {/\* MODAL: Rezervacija \*/}

      {openRes && (

        <div className="modal" role="dialog" aria-modal="true">

          <div className="modal-card">

            <ReservationForm

              initial={{

                sala\_id: openRes.sala.id,

                datum: openRes.date || "",

                vreme\_od: openRes.from || "",

                vreme\_do: openRes.to || "",

                tip\_dogadjaja: openRes.tip\_dogadjaja || "",

                status: "pending",

              }}

              salaName={openRes.sala.naziv}

              onCancel={()=>setOpenRes(null)}

              onSuccess={()=>{ setOpenRes(null); load(); }}

            />

          </div>

        </div>

      )}

    </main>

  );

}

import React, { useMemo } from "react";

import {

  DndContext,

  useDraggable,

  useSensors,

  useSensor,

  PointerSensor,

  TouchSensor,

} from "@dnd-kit/core";

import { restrictToParentElement } from "@dnd-kit/modifiers";

import "./floor-map.css";

/\* helpers \*/

function num(...vals) {

  for (const v of vals) {

    if (v === 0 || v === "0") return 0;

    if (v !== undefined && v !== null && v !== "") return Number(v);

  }

  return undefined;

}

function clamp(v, min, max){ return Math.max(min, Math.min(max, v)); }

/\* Draggable + clickable tile \*/

function RoomTile({ r, state, x, y, w, h, cellSize, editable, canOpen, onOpen }) {

  const id = String(r.id);

  const { attributes, listeners, setNodeRef, transform, isDragging } = useDraggable({

    id,

    disabled: !editable,

  });

  const style = {

    left: x \* cellSize,

    top:  y \* cellSize,

    width:  w \* cellSize,

    height: h \* cellSize,

    transform: transform ? `translate3d(${transform.x}px, ${transform.y}px, 0)` : undefined,

  };

  const handleClick = (e) => {

    // Ako je edit mode ili je bilo draga — ignoriši.

    if (editable || isDragging) return;

    if (!canOpen) return; // neaktivna ili zauzeta

    onOpen?.();

  };

  return (

    <div

      ref={setNodeRef}

      {...attributes}

      {...listeners}

      className={`room ${state} ${editable ? "draggable" : ""} ${isDragging ? "dragging" : ""}`}

      style={style}

      onClick={handleClick}

      title={

        editable

          ? "Prevuci da promeniš poziciju"

          : canOpen ? "Klikni za rezervaciju" : "Nije dostupna"

      }

      aria-label={`${r.naziv||"Sala"}, kapacitet ${r.kapacitet||"-"}, ${state}`}

    >

      <div className="room-name">{r.naziv || r.name || "Sala"}</div>

      <div className="room-meta">

        <span className="chip">{r.kapacitet||"-"}</span>

        <span className={`chip ${state}`}>

          {state==="free"?"Slobodna": state==="busy"?"Zauzeta": state==="inactive"?"Neaktivna":"?"}

        </span>

      </div>

    </div>

  );

}

export default function FloorMap({

  sale, isFree, date, from, to, onReserve, cellSize = 32,

  editable = false, onDrop

}) {

  const sensors = useSensors(

    useSensor(PointerSensor, { activationConstraint: { distance: 3 } }),

    useSensor(TouchSensor,   { activationConstraint: { distance: 3 } }),

  );

  const byFloor = useMemo(() => {

    const m = new Map();

    (sale || []).forEach(s => {

      const raw = s.floor ?? s.sprat ?? 0;

      const f = Number(raw);

      if (!m.has(f)) m.set(f, []);

      m.get(f).push(s);

    });

    return Array.from(m.entries()).sort((a,b)=>a[0]-b[0]);

  }, [sale]);

  function bounds(rooms) {

    let maxX=0, maxY=0;

    rooms.forEach((r, i) => {

      const x = num(r.layout\_x, r.layoutX, r.x, r.col) ?? (i % 10);

      const y = num(r.layout\_y, r.layoutY, r.y, r.row) ?? Math.floor(i / 10);

      const w = Math.max(1, num(r.layout\_w, r.layoutW, r.w, r.width) ?? 1);

      const h = Math.max(1, num(r.layout\_h, r.layoutH, r.h, r.height) ?? 1);

      maxX = Math.max(maxX, x + w);

      maxY = Math.max(maxY, y + h);

    });

    return { cols: Math.max(10, maxX), rows: Math.max(6, maxY) };

  }

  return (

    <section className="floor-section">

      {byFloor.map(([floor, rooms])=>{

        const {cols, rows} = bounds(rooms);

        function handleDragEnd(event) {

          if (!editable) return;

          const { active, delta } = event;

          const id = active?.id;

          if (!id) return;

          const r = rooms.find(rr => String(rr.id) === String(id));

          if (!r) return;

          const x = num(r.layout\_x, r.layoutX, r.x, r.col) ?? 0;

          const y = num(r.layout\_y, r.layoutY, r.y, r.row) ?? 0;

          const w = Math.max(1, num(r.layout\_w, r.layoutW, r.w, r.width) ?? 1);

          const h = Math.max(1, num(r.layout\_h, r.layoutH, r.h, r.height) ?? 1);

          const nx = clamp(x + Math.round((delta?.x || 0) / cellSize), 0, cols - w);

          const ny = clamp(y + Math.round((delta?.y || 0) / cellSize), 0, rows - h);

          if ((nx !== x || ny !== y) && onDrop) {

            onDrop(r.id, { floor: Number(r.floor ?? floor), layout\_x: nx, layout\_y: ny });

          }

        }

        return (

          <div key={floor} className="floor-card">

            <div className="floor-head">

              <h2>Sprat {floor}</h2>

              <div className="legend">

                <span className="dot free" /> Slobodna

                <span className="dot busy" /> Zauzeta

                <span className="dot inactive" /> Neaktivna

              </div>

            </div>

            <DndContext sensors={sensors} onDragEnd={handleDragEnd} modifiers={[restrictToParentElement]}>

              <div

                className="floor-canvas"

                style={{

                  "--cell": `${cellSize}px`,

                  width: `calc(${cols} \* var(--cell))`,

                  height: `calc(${rows} \* var(--cell))`,

                }}

                role="group"

                aria-label={`Mapa sprata ${floor}`}

              >

                {/\* grid \*/}

                {[...Array(rows)].map((\_,y)=>

                  <div key={`r${y}`} className="grid-row" style={{ top: y\*cellSize }} />

                )}

                {[...Array(cols)].map((\_,x)=>

                  <div key={`c${x}`} className="grid-col" style={{ left: x\*cellSize }} />

                )}

                {rooms.map((r, i)=>{

                  const x = (num(r.layout\_x, r.layoutX, r.x, r.col) ?? (i % 10));

                  const y = (num(r.layout\_y, r.layoutY, r.y, r.row) ?? Math.floor(i / 10));

                  const w = Math.max(1, num(r.layout\_w, r.layoutW, r.w, r.width) ?? 1);

                  const h = Math.max(1, num(r.layout\_h, r.layoutH, r.h, r.height) ?? 1);

                  const active = r.status === true || r.status === 1 || String(r.status).toLowerCase().includes("aktiv");

                  const hasTerm = Boolean(date && from && to);

                  const free = hasTerm ? isFree(r, date, from, to) : null;

                  let state = "inactive";

                  if (active) state = free===null ? "unknown" : (free ? "free" : "busy");

                  const canOpen = !editable && active && free !== false;

                  return (

                    <RoomTile

                      key={r.id ?? `${floor}-${i}`}

                      r={r}

                      state={state}

                      x={x} y={y} w={w} h={h}

                      cellSize={cellSize}

                      editable={editable}

                      canOpen={canOpen}

                      onOpen={() => onReserve?.(r)}

                    />

                  );

                })}

              </div>

            </DndContext>

          </div>

        );

      })}

    </section>

  );

}

##### Upravljanje salama

Ova stranica je admin panel za upravljanje salama (CRUD) i zasniva se na hook-u useSalesData, koji učitava podatke o salama, drži “loading” i serverske poruke i nudi reload. Lokalno stanje editing upravlja modal formom: null znači da modal nije otvoren, objekat označava izmenu postojeće sale, a false zatvaranje. Za kreiranje se koristi objekat blank sa podrazumevanim vrednostima, uključujući polja rasporeda (sprat i koordinate) kako bi forma imala kompletan set polja od starta.

Deo za filtere i pretragu omogućava brzo sužavanje liste: tekstualna pretraga prolazi kroz naziv, tip, opis, kapacitet, status i sprat; dodatno se filtrira po statusu (aktivna/neaktivna), tipu i opsegu kapaciteta. Opcije za tip se dinamički izvlače iz stvarnih podataka, pa UI uvek nudi realne vrednosti. Sortiranje radi nad više kolona (ID, naziv, tip, kapacitet, sprat, status) i prebacuje smer kada ponovo kliknemo istu kolonu; status se sortira kao boolean, a stringovi koriste lokalizovano poređenje. Kada se promene filteri, sort ili sama lista, stranica se resetuje na prvu, da ne ostanemo na “praznoj” strani.

Paginacija je potpuno lokalna: računa ukupan broj stavki nakon filtriranja/sortiranja, određuje prikazani opseg i gradi “prozor” brojeva stranica, uz kontrole za prvu, prethodnu, sledeću i poslednju stranu. To omogućava glatku navigaciju i sa većim skupovima podataka. U tabeli su zaglavlja klikabilna za sortiranje, a svaka stavka nudi akcije “Izmeni” (otvara modal sa popunjenim podacima) i “Obriši” (uz potvrdu, pa uklanjanje iz liste i poruku o uspehu). Vizuelne značke (badge) jasno pokazuju status sale.

Operacije kreiranja, izmene i brisanja prvo prolaze kroz blagu sanitizaciju normalizePayload (npr. obavezna numerika, minimalne vrednosti, čišćenje prazne cene), a zatim pogađaju odgovarajuće endpointe. Nakon uspeha, lokalna lista se odmah osvežava (prepend za kreiranje, zamena stavke za izmenu, filter-out za brisanje) i setuje se pozitivna poruka. Greške se hvataju i korisniku signalizuju, dok se posebna poruka koristi za “reload” i sistemske obaveštenja. Modal SaleForm služi i za kreiranje i za izmenu: šalje payload nazad roditeljskom komponentom, koja odlučuje da li ide POST ili PUT, i po uspehu zatvara modal.

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

import api from "../api/client";

import SaleForm from "../components/SaleForm";

import useSalesData from "../hooks/useSalesData";

import "./sale.css";

const SORTABLE = {

  id: "id",

  naziv: "naziv",

  tip: "tip",

  kapacitet: "kapacitet",

  status: "status",

  // NOVO: sprat je koristan za sortiranje/pregled

  floor: "floor",

};

export default function SalesAdminPage() {

  const { items, setItems, loading, serverMsg, setServerMsg, reload } = useSalesData();

  const [editing, setEditing] = useState(null); // null=create, object=edit, false=closed

  // -------- NOVO: podrazumevane vrednosti za novu salu --------

  const blank = {

    naziv: "",

    tip: "",

    kapacitet: 1,

    opis: "",

    status: true,

    cena: "",

    // layout polja

    floor: 0,

    layout\_x: 0,

    layout\_y: 0,

    layout\_w: 1,

    layout\_h: 1,

  };

  // Filters & search

  const [queryRaw, setQueryRaw] = useState("");

  const query = queryRaw.trim().toLowerCase();

  const [status, setStatus] = useState("all"); // all | active | inactive

  const [tip, setTip] = useState("all");       // dinamički

  const [capMin, setCapMin] = useState("");

  const [capMax, setCapMax] = useState("");

  // Sorting

  const [sortBy, setSortBy] = useState(SORTABLE.id);

  const [sortDir, setSortDir] = useState("desc"); // asc | desc

  // Pagination (lokalna)

  const [page, setPage] = useState(1);

  const [perPage, setPerPage] = useState(10); // 5,10,20,50...

  // ------ Helpers: blaga sanitizacija (bez menjanja forme) ------

  function normalizePayload(payload) {

    const n = { ...payload };

    // osiguraj numeriku i minimale

    n.kapacitet = Math.max(1, Number(n.kapacitet ?? 1));

    n.status = Boolean(n.status);

    n.floor    = Math.max(0, Number(n.floor ?? 0));

    n.layout\_x = Math.max(0, Number(n.layout\_x ?? 0));

    n.layout\_y = Math.max(0, Number(n.layout\_y ?? 0));

    n.layout\_w = Math.max(1, Number(n.layout\_w ?? 1));

    n.layout\_h = Math.max(1, Number(n.layout\_h ?? 1));

    if (n.cena !== undefined && n.cena !== null && n.cena !== "") {

      n.cena = Number(n.cena);

    } else {

      delete n.cena;

    }

    return n;

  }

  // Create/Update/Delete

  async function handleCreate(payload) {

    const body = normalizePayload({ ...blank, ...payload }); // popuni propuštena polja defaultima

    const res = await api.post("/sale", body);

    const created = res.data?.data || res.data;

    setItems((prev) => [created, ...prev]);

    setServerMsg("Sala je uspešno dodata.");

  }

  async function handleUpdate(id, payload) {

    const body = normalizePayload(payload);

    const res = await api.put(`/sale/${id}`, body);

    const updated = res.data?.data || res.data;

    setItems((prev) => prev.map((it) => (it.id === id ? updated : it)));

    setServerMsg("Izmene su sačuvane.");

  }

  async function handleDelete(id) {

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

    await api.delete(`/sale/${id}`);

    setItems((prev) => prev.filter((it) => it.id !== id));

    setServerMsg("Sala je obrisana.");

  }

  // Dinamičke opcije za "Tip"

  const tipOptions = useMemo(() => {

    const set = new Set();

    (items || []).forEach((x) => { if (x?.tip) set.add(String(x.tip)); });

    return ["all", ...Array.from(set)];

  }, [items]);

  // Filtriranje + pretraga + sortiranje

  const filteredSorted = useMemo(() => {

    let list = Array.isArray(items) ? [...items] : [];

    if (query) {

      list = list.filter((x) => {

        const hay = [

          x?.naziv ?? "",

          x?.tip ?? "",

          x?.opis ?? "",

          String(x?.kapacitet ?? ""),

          x?.status ? "aktivna" : "neaktivna",

          // NOVO: pretraga po spratu

          `sprat ${x?.floor ?? ""}`,

        ].join(" ").toLowerCase();

        return hay.includes(query);

      });

    }

    if (status !== "all") {

      const flag = status === "active";

      list = list.filter((x) => Boolean(x?.status) === flag);

    }

    if (tip !== "all") {

      list = list.filter((x) => String(x?.tip ?? "") === tip);

    }

    const min = capMin !== "" ? Number(capMin) : null;

    const max = capMax !== "" ? Number(capMax) : null;

    if (min !== null) list = list.filter((x) => Number(x?.kapacitet ?? 0) >= min);

    if (max !== null) list = list.filter((x) => Number(x?.kapacitet ?? 0) <= max);

    // Sort (dodali smo mogućnost po floor-u ako je izabran)

    list.sort((a, b) => {

      const dir = sortDir === "asc" ? 1 : -1;

      const A = a?.[sortBy];

      const B = b?.[sortBy];

      if (sortBy === "status") {

        return (Number(Boolean(A)) - Number(Boolean(B))) \* dir;

      }

      if (typeof A === "string" || typeof B === "string") {

        return String(A ?? "").localeCompare(String(B ?? ""), "sr", { sensitivity: "base" }) \* dir;

      }

      return ((Number(A) || 0) - (Number(B) || 0)) \* dir;

    });

    return list;

  }, [items, query, status, tip, capMin, capMax, sortBy, sortDir]);

  // Resetuj na prvu stranu kad se promene filteri/sort/lista

  useEffect(() => {

    setPage(1);

  }, [queryRaw, status, tip, capMin, capMax, sortBy, sortDir, items]);

  // Paginacija

  const total = filteredSorted.length;

  const totalPages = Math.max(1, Math.ceil(total / perPage));

  const currentPage = Math.min(page, totalPages);

  const startIdx = (currentPage - 1) \* perPage;

  const endIdx = Math.min(startIdx + perPage, total);

  const pageItems = filteredSorted.slice(startIdx, endIdx);

  function goto(p) {

    const clamped = Math.max(1, Math.min(p, totalPages));

    setPage(clamped);

  }

  function pageWindow(totalPages, current, max = 5) {

    let start = Math.max(1, current - Math.floor(max / 2));

    let end = Math.min(totalPages, start + max - 1);

    if (end - start + 1 < max) start = Math.max(1, end - max + 1);

    const arr = [];

    for (let i = start; i <= end; i++) arr.push(i);

    return arr;

  }

  function toggleSort(col) {

    if (sortBy === col) setSortDir((d) => (d === "asc" ? "desc" : "asc"));

    else { setSortBy(col); setSortDir(col === "id" ? "desc" : "asc"); }

  }

  function resetFilters() {

    setQueryRaw("");

    setStatus("all");

    setTip("all");

    setCapMin("");

    setCapMax("");

    setSortBy(SORTABLE.id);

    setSortDir("desc");

  }

  return (

    <div className="lp">

      <div aria-hidden="true" className="lp-bg">

        <div className="blob blob-1" />

        <div className="blob blob-2" />

        <div className="blob blob-3" />

      </div>

      <main className="sales" role="main">

        <div className="sales-head">

          <div>

            <p className="eyebrow">Administracija</p>

            <h1>Sale — CRUD</h1>

            <p className="muted">Dodaj, izmeni ili obriši salu (naziv, tip, kapacitet, opis, status, <strong>sprat i koordinate</strong>).</p>

          </div>

          <div className="sales-actions">

            <button className="btn btn--ghost" onClick={reload}> Osveži</button>

            <button className="btn btn--primary" onClick={() => setEditing({ ...blank })}>

              + Nova sala

            </button>

          </div>

        </div>

        {/\* FILTERI \*/}

        <section className="filters" aria-label="Filteri i pretraga">

          <div className="filters-grid">

            <div className="filter">

              <label className="filter-label">Pretraga</label>

              <input

                value={queryRaw}

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

                placeholder="Naziv, tip, opis, status, kapacitet, sprat…"

                aria-label="Pretraga"

              />

            </div>

            <div className="filter">

              <label className="filter-label">Status</label>

              <select value={status} onChange={(e) => setStatus(e.target.value)}>

                <option value="all">Sve</option>

                <option value="active">Aktivna</option>

                <option value="inactive">Neaktivna</option>

              </select>

            </div>

            <div className="filter">

              <label className="filter-label">Tip</label>

              <select value={tip} onChange={(e) => setTip(e.target.value)}>

                {tipOptions.map((t) => (

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

                    {t === "all" ? "Svi tipovi" : t}

                  </option>

                ))}

              </select>

            </div>

            <div className="filter">

              <label className="filter-label">Kapacitet (od)</label>

              <input

                type="number"

                min="1"

                placeholder="npr. 20"

                value={capMin}

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

              />

            </div>

            <div className="filter">

              <label className="filter-label">Kapacitet (do)</label>

              <input

                type="number"

                min="1"

                placeholder="npr. 100"

                value={capMax}

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

              />

            </div>

            <div className="filter filter-actions">

              <button type="button" className="btn btn--ghost" onClick={resetFilters}>

                Reset filtera

              </button>

              <span className="muted results-count">

                Rezultata: <strong>{total}</strong>

              </span>

            </div>

          </div>

        </section>

        {serverMsg && <div className="alert alert--ok">{serverMsg}</div>}

        {loading && <div className="alert">Učitavanje...</div>}

        {/\* Kontrole paginacije (top) \*/}

        <div className="pagination">

          <div className="page-size">

            <label>Prikaži:</label>

            <select value={perPage} onChange={(e) => { setPerPage(Number(e.target.value)); setPage(1); }}>

              {[5,10,20,50,100].map((n) => <option key={n} value={n}>{n}/str</option>)}

            </select>

          </div>

          <div className="page-info">

            <span>Prikaz {total === 0 ? 0 : startIdx + 1}–{endIdx} od {total}</span>

          </div>

          <div className="page-ctrls">

            <button className="btn btn--ghost" onClick={() => goto(1)} disabled={currentPage === 1}>« Prva</button>

            <button className="btn btn--ghost" onClick={() => goto(currentPage - 1)} disabled={currentPage === 1}>‹ Prethodna</button>

            {pageWindow(totalPages, currentPage, 5).map((p) => (

              <button

                key={p}

                className={`btn ${p === currentPage ? "btn--primary" : "btn--ghost"} page-btn`}

                onClick={() => goto(p)}

              >

                {p}

              </button>

            ))}

            <button className="btn btn--ghost" onClick={() => goto(currentPage + 1)} disabled={currentPage === totalPages}>Sledeća ›</button>

            <button className="btn btn--ghost" onClick={() => goto(totalPages)} disabled={currentPage === totalPages}>Poslednja »</button>

          </div>

        </div>

        <div className="card">

          <div className="table-wrap">

            <table className="tbl">

              <thead>

                <tr>

                  <th

                    className={`sortable ${sortBy === "id" ? `sorted ${sortDir}` : ""}`}

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

                    aria-sort={sortBy === "id" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("id")}

                  >

                    ID

                  </th>

                  <th

                    className={`sortable ${sortBy === "naziv" ? `sorted ${sortDir}` : ""}`}

                    onClick={() => toggleSort("naziv")}

                    aria-sort={sortBy === "naziv" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("naziv")}

                  >

                    Naziv

                  </th>

                  <th

                    className={`sortable ${sortBy === "tip" ? `sorted ${sortDir}` : ""}`}

                    onClick={() => toggleSort("tip")}

                    aria-sort={sortBy === "tip" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("tip")}

                  >

                    Tip

                  </th>

                  <th

                    className={`right sortable ${sortBy === "kapacitet" ? `sorted ${sortDir}` : ""}`}

                    onClick={() => toggleSort("kapacitet")}

                    aria-sort={sortBy === "kapacitet" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("kapacitet")}

                  >

                    Kapacitet

                  </th>

                  {/\* NOVO: prikaz sprata, minimalna izmena UI-a \*/}

                  <th

                    className={`sortable ${sortBy === "floor" ? `sorted ${sortDir}` : ""}`}

                    onClick={() => toggleSort("floor")}

                    aria-sort={sortBy === "floor" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("floor")}

                  >

                    Sprat

                  </th>

                  <th

                    className={`sortable ${sortBy === "status" ? `sorted ${sortDir}` : ""}`}

                    onClick={() => toggleSort("status")}

                    aria-sort={sortBy === "status" ? sortDir : "none"}

                    role="button"

                    tabIndex={0}

                    onKeyDown={(e) => e.key === "Enter" && toggleSort("status")}

                  >

                    Status

                  </th>

                  <th className="right">Akcije</th>

                </tr>

              </thead>

              <tbody>

                {pageItems.length === 0 ? (

                  <tr>

                    <td colSpan={7} className="muted center">Nema podataka.</td>

                  </tr>

                ) : (

                  pageItems.map((it) => (

                    <tr key={it.id}>

                      <td>{it.id}</td>

                      <td>{it.naziv}</td>

                      <td>{it.tip}</td>

                      <td className="right">{it.kapacitet}</td>

                      <td>{it.floor ?? 0}</td>

                      <td>

                        <span className={`badge ${it.status ? "badge--ok" : "badge--muted"}`}>

                          {it.status ? "Aktivna" : "Neaktivna"}

                        </span>

                      </td>

                      <td className="right actions">

                        <button className="btn btn--ghost" onClick={() => setEditing(it)}>Izmeni</button>

                        <button className="btn btn--danger" onClick={() => handleDelete(it.id)}>Obriši</button>

                      </td>

                    </tr>

                  ))

                )}

              </tbody>

            </table>

          </div>

        </div>

        {/\* Kontrole paginacije (bottom) \*/}

        <div className="pagination">

          <div className="page-size">

            <label>Prikaži:</label>

            <select value={perPage} onChange={(e) => { setPerPage(Number(e.target.value)); setPage(1); }}>

              {[5,10,20,50,100].map((n) => <option key={n} value={n}>{n}/str</option>)}

            </select>

          </div>

          <div className="page-info">

            <span>Prikaz {total === 0 ? 0 : startIdx + 1}–{endIdx} od {total}</span>

          </div>

          <div className="page-ctrls">

            <button className="btn btn--ghost" onClick={() => goto(1)} disabled={currentPage === 1}>« Prva</button>

            <button className="btn btn--ghost" onClick={() => goto(currentPage - 1)} disabled={currentPage === 1}>‹ Prethodna</button>

            {pageWindow(totalPages, currentPage, 5).map((p) => (

              <button

                key={p}

                className={`btn ${p === currentPage ? "btn--primary" : "btn--ghost"} page-btn`}

                onClick={() => goto(p)}

              >

                {p}

              </button>

            ))}

            <button className="btn btn--ghost" onClick={() => goto(currentPage + 1)} disabled={currentPage === totalPages}>Sledeća ›</button>

            <button className="btn btn--ghost" onClick={() => goto(totalPages)} disabled={currentPage === totalPages}>Poslednja »</button>

          </div>

        </div>

      </main>

      {/\* Modal forma za create/update \*/}

      {editing !== false && editing !== null && (

        <div className="modal" role="dialog" aria-modal="true">

          <div className="modal-card">

            <SaleForm

              // NOVO: za "novu" salu šaljemo blank sa floor/layout\_\* da forma ima polja i default

              initial={editing && editing.id ? editing : blank}

              onCancel={() => setEditing(false)}

              onSubmit={async (payload) => {

                try {

                  if (editing && editing.id) {

                    await handleUpdate(editing.id, payload);

                  } else {

                    await handleCreate(payload);

                  }

                  setEditing(false);

                } catch (e) {

                  console.error(e);

                  alert("Greška pri čuvanju. Proveri prava i backend validaciju.");

                }

              }}

            />

          </div>

        </div>

      )}

    </div>

  );

}

##### AdminStatistike

Ova komponenta predstavlja administratorsku stranicu sa dve glavne sekcije: “Dashboard” i “Korisnici”. Na vrhu održava lokalna stanja za metrike (KPI), skupove podataka za grafikone (dnevne rezervacije, top sale, raspodela po tipu, opterećenje po satima), iskorišćenost, indikator učitavanja i aktivnu karticu. Prelaskom na tab “Dashboard” učitava se set API poziva paralelno (Promise.all), pa se rezultati mapiraju u jednostavne objekte labels/data koje koriste prilagođene komponentе grafikona rađene preko Recharts biblioteke (Line/Bar/Donut). KPI pločice prikazuju zbirne brojke (korisnici, aktivne/ukupne sale, ukupno rezervacija i broj za danas), dok svakom grafu prati naslov i opcioni “Osveži” za ponovno dohvaćanje podataka.

Grafikoni su tanki omotači oko Recharts komponenti: ulazne nizove labels/data prevode u serije oblika { name, value } i renderuju linijski dijagram (trend rezervacija), stubičaste dijagrame (top sale i opterećenje po satima) i prstenasti (raspodela sala po tipu). Uključeni su grid, ose, tooltip i minimalna stilizacija; boje su definisane kroz mali niz COLORS za konzistentan izgled segmenata na donut grafiku. Pored nekih grafika postoji i tekstualna “legend-list”/”legend-col” sa vrednostima formatiranim pomoću Intl.NumberFormat.

Kartica “Korisnici” uvodi CRUD nad nalozima kroz tabelu, paginaciju i modal formu. Stanje drži trenutnu listu korisnika, meta-informacije o stranici (trenutna, poslednja, ukupno), upit za pretragu i modal indikator uz eventualni korisnik za izmenu. Učitavanje korisnika radi preko parametrizovanog API poziva (q, page, per\_page), a kartica nudi osnovnu paginaciju (prva/precedna/sledeća/poslednja) preko kontrola koje samo pozivaju loadUsers sa odgovarajućim parametrima. Modal UserModal služi i za kreiranje i za izmenu: formstate se puni na open, a pri slanju poziva onSave koji delegira na createUser ili updateUser; za izmenu se prazna lozinka ignoriše (ne menja postojeću). Brisanje ima potvrdu i potom osvežava listu (uz korekciju strane ako je poslednji red obrisan).

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

import api from "../api/client";

import "./admin-dashboard.css";

/\* ----------------------------- Recharts ----------------------------- \*/

import {

  ResponsiveContainer,

  LineChart as RLineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip,

  BarChart as RBarChart, Bar,

  PieChart, Pie, Cell

} from "recharts";

/\* ----------------------------- helpers ----------------------------- \*/

function fmt(n) { return new Intl.NumberFormat("sr-RS").format(n ?? 0); }

const toSeries = (labels = [], data = []) =>

  labels.map((l, i) => ({ name: l, value: Number(data[i] || 0) }));

const COLORS = ["#111827", "#4b5563", "#6b7280", "#9ca3af", "#d1d5db", "#e5e7eb"];

// LINE

function LineChart({ labels = [], data = [], height = 220 }) {

  const series = toSeries(labels, data);

  return (

    <div style={{ width: "100%", height }}>

      <ResponsiveContainer>

        <RLineChart data={series} margin={{ top: 8, right: 12, left: 0, bottom: 0 }}>

          <CartesianGrid strokeDasharray="3 3" />

          <XAxis dataKey="name" hide />

          <YAxis allowDecimals={false} />

          <Tooltip />

          <Line type="monotone" dataKey="value" stroke="#111827" strokeWidth={2} dot={false} />

        </RLineChart>

      </ResponsiveContainer>

    </div>

  );

}

// BAR

function BarChart({ labels = [], data = [], height = 220 }) {

  const series = toSeries(labels, data);

  return (

    <div style={{ width: "100%", height }}>

      <ResponsiveContainer>

        <RBarChart data={series} margin={{ top: 8, right: 12, left: 0, bottom: 0 }}>

          <CartesianGrid strokeDasharray="3 3" />

          <XAxis dataKey="name" hide />

          <YAxis allowDecimals={false} />

          <Tooltip />

          <Bar dataKey="value" fill="#111827" />

        </RBarChart>

      </ResponsiveContainer>

    </div>

  );

}

// DONUT

function DonutChart({ labels = [], data = [], height = 220 }) {

  const series = toSeries(labels, data);

  return (

    <div style={{ width: "100%", height }}>

      <ResponsiveContainer>

        <PieChart>

          <Pie

            data={series}

            dataKey="value"

            nameKey="name"

            innerRadius="55%"

            outerRadius="80%"

            paddingAngle={1}

          >

            {series.map((\_, i) => (

              <Cell key={i} fill={COLORS[i % COLORS.length]} />

            ))}

          </Pie>

          <Tooltip />

        </PieChart>

      </ResponsiveContainer>

    </div>

  );

}

/\* ----------------------------- Users Modal ----------------------------- \*/

function UserModal({ open, initial, onClose, onSave }) {

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

  const [err, setErr] = useState(null);

  useEffect(() => {

    if (!open) return;

    if (initial) setForm({ name: initial.name || "", email: initial.email || "", password: "", role: initial.role || "korisnik" });

    else setForm({ name: "", email: "", password: "", role: "korisnik" });

    setErr(null);

  }, [open, initial]);

  async function submit(e) {

    e.preventDefault();

    setErr(null);

    try {

      await onSave(form);

      onClose?.();

    } catch (e) {

      setErr(e?.response?.data?.message || "Greška pri čuvanju.");

    }

  }

  if (!open) return null;

  return (

    <div className="modal" role="dialog" aria-modal="true">

      <div className="modal-card">

        <form onSubmit={submit} className="user-form" noValidate>

          <h3>{initial ? "Izmeni korisnika" : "Novi korisnik"}</h3>

          {err && <div className="alert">{String(err)}</div>}

          <div className="field">

            <label>Ime i prezime</label>

            <input value={form.name} onChange={e => setForm(f => ({ ...f, name: e.target.value }))} required />

          </div>

          <div className="field">

            <label>Email</label>

            <input type="email" value={form.email} onChange={e => setForm(f => ({ ...f, email: e.target.value }))} required />

          </div>

          <div className="field">

            <label>Lozinka {initial && <span className="muted">(ostavi prazno ako ne menjaš)</span>}</label>

            <input type="password" value={form.password} onChange={e => setForm(f => ({ ...f, password: e.target.value }))} />

          </div>

          <div className="field">

            <label>Uloga</label>

            <select value={form.role} onChange={e => setForm(f => ({ ...f, role: e.target.value }))}>

              <option value="korisnik">Korisnik</option>

              <option value="menadzer">Menadžer</option>

              <option value="administrator">Administrator</option>

            </select>

          </div>

          <div className="form-actions">

            <button type="button" className="btn btn--ghost" onClick={onClose}>Otkaži</button>

            <button type="submit" className="btn btn--primary">{initial ? "Sačuvaj" : "Kreiraj"}</button>

          </div>

        </form>

      </div>

    </div>

  );

}

/\* ----------------------------- Main Page ----------------------------- \*/

export default function AdminDashboardPage() {

  // KPIs & charts

  const [kpi, setKpi] = useState(null);

  const [daily, setDaily] = useState({ labels: [], data: [] });

  const [topRooms, setTopRooms] = useState({ labels: [], data: [] });

  const [byType, setByType] = useState({ labels: [], data: [] });

  const [hourly, setHourly] = useState({ labels: [], data: [] });

  const [util, setUtil] = useState(null);

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

  const [tab, setTab] = useState("dashboard"); // dashboard | users

  // Users

  const [users, setUsers] = useState([]);

  const [uMeta, setUMeta] = useState({ current\_page: 1, last\_page: 1, per\_page: 15, total: 0 });

  const [q, setQ] = useState("");

  const [modalOpen, setModalOpen] = useState(false);

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

  /\* -------------- data loaders -------------- \*/

  const loadDashboard = useCallback(async () => {

    setLoading(true);

    try {

      const [k, d, t, bt, h, u] = await Promise.all([

        api.get("/admin/kpis"),

        api.get("/admin/reservations/daily?days=30"),

        api.get("/admin/rooms/top?limit=10"),

        api.get("/admin/rooms/by-type"),

        api.get("/admin/hourly?days=14"),

        api.get("/admin/utilization")

      ]);

      setKpi(k.data);

      setDaily(d.data);

      setTopRooms(t.data);

      setByType(bt.data);

      setHourly(h.data);

      setUtil(u.data);

    } finally { setLoading(false); }

  }, []);

  const loadUsers = useCallback(async (page = 1, query = q) => {

    const res = await api.get(`/users`, { params: { q: query, page, per\_page: 10 } });

    // Laravel paginate: { data:[], current\_page, last\_page, per\_page, total... }

       const payload = res.data;

    const list = payload.data ?? payload; // fallback

    setUsers(list);

    setUMeta({

      current\_page: payload.current\_page ?? 1,

      last\_page: payload.last\_page ?? 1,

      per\_page: payload.per\_page ?? 10,

      total: payload.total ?? list.length

    });

  }, [q]);

  useEffect(() => { loadDashboard(); }, [loadDashboard]);

  useEffect(() => { if (tab === "users") loadUsers(1, q); }, [tab, loadUsers]);

  /\* -------------- users actions -------------- \*/

  async function createUser(body) {

    const res = await api.post("/users", body);

    await loadUsers(uMeta.current\_page, q);

    return res.data;

  }

  async function updateUser(id, body) {

    const payload = { ...body };

    if (!payload.password) delete payload.password;

    const res = await api.put(`/users/${id}`, payload);

    await loadUsers(uMeta.current\_page, q);

    return res.data;

  }

  async function deleteUser(id) {

    if (!window.confirm("Obrisati korisnika?")) return;

    await api.delete(`/users/${id}`);

    const newPage = (users.length === 1 && uMeta.current\_page > 1) ? uMeta.current\_page - 1 : uMeta.current\_page;

    await loadUsers(newPage, q);

  }

  /\* -------------- render -------------- \*/

  return (

    <main className="admin" role="main">

      <div className="admin-head">

        <div>

          <p className="eyebrow">Administracija</p>

          <h1>Dashboard & Korisnici</h1>

          <p className="muted">Pregled ključnih metrika i upravljanje nalozima.</p>

        </div>

        <div className="tabs">

          <button className={`tab ${tab === "dashboard" ? "active" : ""}`} onClick={() => setTab("dashboard")}>Dashboard</button>

          <button className={`tab ${tab === "users" ? "active" : ""}`} onClick={() => setTab("users")}>Korisnici</button>

        </div>

      </div>

      {tab === "dashboard" && (

        <>

          {/\* KPI tiles \*/}

          <section className="kpis">

            <div className="kpi">

              <div className="kpi-title">Korisnici</div>

              <div className="kpi-value">{kpi ? fmt(kpi.users\_total) : "—"}</div>

            </div>

            <div className="kpi">

              <div className="kpi-title">Sale (aktivne)</div>

              <div className="kpi-value">

                {kpi ? `${fmt(kpi.rooms\_active)} / ${fmt(kpi.rooms\_total)}` : "—"}

              </div>

            </div>

            <div className="kpi">

              <div className="kpi-title">Rezervacije (ukupno)</div>

              <div className="kpi-value">{kpi ? fmt(kpi.reservations\_total) : "—"}</div>

            </div>

            <div className="kpi">

              <div className="kpi-title">Danas</div>

              <div className="kpi-value">{kpi ? fmt(kpi.reservations\_today) : "—"}</div>

            </div>

          </section>

          {/\* charts \*/}

          <div className="cards-grid">

            <div className="card">

              <div className="card-head">

                <h3>Rezervacije po danu (30d)</h3>

                <button className="btn btn--ghost" onClick={loadDashboard} disabled={loading}>Osveži</button>

              </div>

              <LineChart labels={daily.labels} data={daily.data} />

            </div>

            <div className="card">

              <div className="card-head"><h3>Top sale (po rezervacijama)</h3></div>

              <BarChart labels={topRooms.labels} data={topRooms.data} />

              <div className="legend-list">

                {topRooms.labels?.map((l, i) => (

                  <span key={i} className="legend-item"><span className="dot" /> {l} — {fmt(topRooms.data?.[i])}</span>

                ))}

              </div>

            </div>

            <div className="card">

              <div className="card-head"><h3>Raspodela sala po tipu</h3></div>

              <div className="donut-wrap">

                <DonutChart labels={byType.labels} data={byType.data} />

                <div className="legend-col">

                  {byType.labels?.map((l, i) => (

                    <span key={i} className="legend-item"><span className={`swatch seg-${(i % 6) + 1}`} /> {l} — {fmt(byType.data?.[i])}</span>

                  ))}

                </div>

              </div>

            </div>

            <div className="card">

              <div className="card-head"><h3>Opterećenje po satima (14d)</h3></div>

              <BarChart labels={hourly.labels} data={hourly.data} />

            </div>

            <div className="card">

              <div className="card-head"><h3>Iskorišćenost (30d)</h3></div>

              <div className="util">

                <div className="util-row">

                  <div>Aktivnih sala</div><strong>{util ? fmt(util.active\_rooms) : "—"}</strong>

                </div>

                <div className="util-row">

                  <div>Sa bar jednom rez.</div><strong>{util ? fmt(util.busy\_rooms) : "—"}</strong>

                </div>

                <div className="util-row">

                  <div>Udeo</div><strong>{util ? `${Math.round(util.ratio \* 100)}%` : "—"}</strong>

                </div>

              </div>

            </div>

          </div>

        </>

      )}

      {tab === "users" && (

        <section className="users">

          <div className="users-head">

            <div className="field">

              <label>Pretraga</label>

              <input value={q} onChange={e => setQ(e.target.value)} placeholder="ime, email, uloga…" />

            </div>

            <div className="actions">

              <button className="btn btn--ghost" onClick={() => loadUsers(1, q)}>Traži</button>

              <button className="btn btn--primary" onClick={() => { setEditing(null); setModalOpen(true); }}>+ Novi</button>

            </div>

          </div>

          <div className="card">

            <div className="table-wrap">

              <table className="tbl">

                <thead>

                  <tr>

                    <th>ID</th><th>Ime</th><th>Email</th><th>Uloga</th><th className="right">Akcije</th>

                  </tr>

                </thead>

                <tbody>

                  {users.length === 0 ? (

                    <tr><td colSpan={5} className="center muted">Nema rezultata.</td></tr>

                  ) : users.map(u => (

                    <tr key={u.id}>

                      <td>{u.id}</td>

                      <td>{u.name}</td>

                      <td>{u.email}</td>

                      <td><span className="badge">{u.role}</span></td>

                      <td className="right">

                        <button className="btn btn--ghost" onClick={() => { setEditing(u); setModalOpen(true); }}>Izmeni</button>

                        <button className="btn btn--danger" onClick={() => deleteUser(u.id)}>Obriši</button>

                      </td>

                    </tr>

                  ))}

                </tbody>

              </table>

            </div>

            <div className="pagination">

              <span>Strana {uMeta.current\_page} / {uMeta.last\_page} • {fmt(uMeta.total)} zapisa</span>

              <div className="page-ctrls">

                <button className="btn btn--ghost" disabled={uMeta.current\_page <= 1} onClick={() => loadUsers(1, q)}>« Prva</button>

                <button className="btn btn--ghost" disabled={uMeta.current\_page <= 1} onClick={() => loadUsers(uMeta.current\_page - 1, q)}>‹ Preth</button>

                <button className="btn btn--ghost" disabled={uMeta.current\_page >= uMeta.last\_page} onClick={() => loadUsers(uMeta.current\_page + 1, q)}>Sled ›</button>

                <button className="btn btn--ghost" disabled={uMeta.current\_page >= uMeta.last\_page} onClick={() => loadUsers(uMeta.last\_page, q)}>Posl »</button>

              </div>

            </div>

          </div>

          <UserModal

            open={modalOpen}

            initial={editing}

            onClose={() => setModalOpen(false)}

            onSave={(body) => editing ? updateUser(editing.id, body) : createUser(body)}

          />

        </section>

      )}

    </main>

  );

}

##### AdminPreporuke

Ova React stranica je administratorski pregled preporuka sa mogućnošću filtriranja, sortiranja, paginacije i „inline” uređivanja/brisanja. Na početku definišu se stanja za podatke (items), indikatore učitavanja i poruka, kao i interfejs stanja za filtere (tekstualna pretraga, opseg datuma), kriterijume sortiranja (kolona i smer) i lokalnu paginaciju (strana i broj redova po strani). Tu su i pomoćne funkcije toISO i hhmm koje normalizuju datum i vreme u formate pogodne za prikaz i poređenje.

Podaci se učitavaju iz backenda pozivom GET /preporuke u load funkciji (memoizovanoj useCallback). Rezultat se bezbedno „raspakuje” (bilo da stigne kao {data:[...]} ili [...]) i smešta u items. U slučaju greške, ispisuje se poruka i gasi loading. Efekat useEffect pokreće load pri montiranju komponente, tako da administrator odmah vidi tabelu.

Filtriranje i sortiranje sprovodi se nad items putem useMemo. Tekstualna pretraga poredi objedinjeni „haystack” (ime korisnika, naziv sale, tip događaja, vreme i ID), dok filteri datuma ograničavaju skup po opsegu dateFrom–dateTo (uz normalizaciju toISO). Sortiranje podržava više ključeva (datum, vreme, korisnik, sala, događaj) i funkcioniše tako što poredi pripremljene vrednosti (npr. toISO(datum), hhmm(vreme\_od), user\_name.toLowerCase()) uz smer asc/desc.

Paginacija je lokalna: izračunava se totalPages iz veličine filtriranog niza i perPage, a za prikaz se uzima presečak pageItems. Kontrole na dnu („Prva/Prethodna/Sledeća/Poslednja” i izbor veličine strane) samo pomeraju stanje page ili menjaju perPage, uz reset strane na 1 kada je potrebno. Brojač u sredini prikazuje opseg redova (npr. „Prikaz 11–20 od 58”).

Uređivanje je „inline”: klikom na „Uredi” red prelazi u režim sa input poljima za sala\_id, tip\_dogadjaja, datum, vreme\_od i vreme\_do. Ta stanja se drže u form, a identifikator aktivnog reda u editId. Klik na „Sačuvaj” šalje PUT /preporuke/{id} sa validnim payload-om, pa — ako prođe — optimistički ažurira odgovarajući element u items i zatvara režim izmene. Eventualne greške iz backenda se sabiraju i prikazuju kroz msg. „Otkaži” vraća red u normalan prikaz bez slanja zahteva.

Brisanje koristi potvrdu preko window.confirm, zatim šalje DELETE /preporuke/{id} i po uspehu uklanja red iz stanja. Dugme je privremeno onemogućeno (deletingId) dok traje operacija, a poruke o uspehu/grešci pojavljuju se u msg panelu iznad tabele. Na vrhu stranice nalazi se jednostavan toolbar sa naslovom i „Osveži” koji ponovo poziva load

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

import api from "../api/client";

import "./admin-preporuke.css";

/\* utils \*/

function toISO(d) {

  if (!d) return "";

  if (typeof d === "string" && /^\d{4}-\d{2}-\d{2}$/.test(d)) return d;

  try { return new Date(d).toISOString().slice(0, 10); } catch { return ""; }

}

function hhmm(t) { return String(t || "").slice(0, 5); } // od "HH:mm:ss" napravi "HH:mm"

export default function AdminPreporuke() {

  // data

  const [items, setItems] = useState([]);

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

  const [msg, setMsg] = useState("");

  // filters/sort

  const [q, setQ] = useState("");

  const [dateFrom, setDateFrom] = useState("");

  const [dateTo, setDateTo] = useState("");

  const [sortBy, setSortBy] = useState("datum"); // datum | vreme\_od | korisnik | sala | dogadjaj

  const [sortDir, setSortDir] = useState("desc"); // asc | desc

  // pagination

  const [page, setPage] = useState(1);

  const [perPage, setPerPage] = useState(10);

  // inline edit

  const [editId, setEditId] = useState(null);

  const [form, setForm] = useState({

    sala\_id: "",

    datum: "",

    vreme\_od: "",

    vreme\_do: "",

    tip\_dogadjaja: "",

  });

  // action states

  const [saving, setSaving] = useState(false);

  const [deletingId, setDeletingId] = useState(null);

  const load = useCallback(async () => {

    setLoading(true); setMsg("");

    try {

      const res = await api.get("/preporuke");

      // backend može vratiti {data:[...]} ili direktno [...]

      const data = res?.data?.data || res?.data || [];

      setItems(Array.isArray(data) ? data : []);

    } catch (e) {

      setMsg("Ne mogu da učitam preporuke (proveri autentikaciju/permission).");

    } finally {

      setLoading(false);

    }

  }, []);

  useEffect(() => { load(); }, [load]);

  function startEdit(row) {

    setEditId(row.id);

    setForm({

      sala\_id: row.sala\_id ?? "",

      datum: toISO(row.datum) || "",

      vreme\_od: hhmm(row.vreme\_od),

      vreme\_do: hhmm(row.vreme\_do),

      tip\_dogadjaja: row.tip\_dogadjaja || "",

    });

  }

  function cancelEdit() {

    setEditId(null);

    setForm({ sala\_id:"", datum:"", vreme\_od:"", vreme\_do:"", tip\_dogadjaja:"" });

  }

  async function saveEdit(id) {

    setSaving(true); setMsg("");

    try {

      const payload = {

        sala\_id: String(form.sala\_id).trim(),

        datum: form.datum,

        vreme\_od: form.vreme\_od,

        vreme\_do: form.vreme\_do,

        tip\_dogadjaja: form.tip\_dogadjaja,

      };

      await api.put(`/preporuke/${id}`, payload);

      // optimistički update

      setItems(prev => prev.map(x => x.id === id ? {

        ...x,

        sala\_id: payload.sala\_id,

        datum: payload.datum,

        vreme\_od: payload.vreme\_od,

        vreme\_do: payload.vreme\_do,

        tip\_dogadjaja: payload.tip\_dogadjaja,

      } : x));

      setMsg("Preporuka je uspešno ažurirana.");

      cancelEdit();

    } catch (e) {

      const er = e?.response?.data?.errors;

      if (er) {

        const flat = Object.values(er).flat().join(" ");

        setMsg(flat || "Greška pri ažuriranju.");

      } else {

        setMsg(e?.response?.data?.message || "Greška pri ažuriranju.");

      }

    } finally { setSaving(false); }

  }

  async function remove(id) {

    if (!window.confirm("Obrisati preporuku?")) return;

    setDeletingId(id); setMsg("");

    try {

      await api.delete(`/preporuke/${id}`);

      setItems(prev => prev.filter(x => x.id !== id));

      setMsg("Preporuka je obrisana.");

    } catch (e) {

      setMsg(e?.response?.data?.message || "Ne mogu da obrišem preporuku.");

    } finally { setDeletingId(null); }

  }

  // filter + sort (koristimo user\_name i sala\_naziv)

  const filtered = useMemo(() => {

    const query = q.trim().toLowerCase();

    let list = [...items];

    if (query) {

      list = list.filter(r => {

        const userName = r.user\_name || String(r.user\_id || "");

        const salaName = r.sala\_naziv || String(r.sala\_id || "");

        const hay = [

          userName, salaName, r.tip\_dogadjaja, r.vreme\_od, r.vreme\_do, String(r.id)

        ].join(" ").toLowerCase();

        return hay.includes(query);

      });

    }

    if (dateFrom) list = list.filter(r => (toISO(r.datum) || "") >= dateFrom);

    if (dateTo)   list = list.filter(r => (toISO(r.datum) || "") <= dateTo);

    list.sort((a,b) => {

      const dir = sortDir === "asc" ? 1 : -1;

      const aUser = String(a.user\_name || "").toLowerCase();

      const bUser = String(b.user\_name || "").toLowerCase();

      const aSala = String(a.sala\_naziv || "").toLowerCase();

      const bSala = String(b.sala\_naziv || "").toLowerCase();

      switch (sortBy) {

        case "korisnik": return aUser.localeCompare(bUser, "sr") \* dir;

        case "sala":     return aSala.localeCompare(bSala, "sr") \* dir;

        case "dogadjaj": return String(a.tip\_dogadjaja||"").localeCompare(String(b.tip\_dogadjaja||""), "sr") \* dir;

        case "vreme\_od": return String(hhmm(a.vreme\_od)).localeCompare(String(hhmm(b.vreme\_od)), "sr") \* dir;

        case "datum":

        default:

          return String(toISO(a.datum)||"").localeCompare(String(toISO(b.datum)||""), "sr") \* dir;

      }

    });

    return list;

  }, [items, q, dateFrom, dateTo, sortBy, sortDir]);

  // pagination

  const total = filtered.length;

  const totalPages = Math.max(1, Math.ceil(total / perPage));

  const cur = Math.min(page, totalPages);

  const start = (cur - 1) \* perPage;

  const end = Math.min(start + perPage, total);

  const pageItems = filtered.slice(start, end);

  function toggleSort(col) {

    if (sortBy === col) setSortDir(d => (d === "asc" ? "desc" : "asc"));

    else { setSortBy(col); setSortDir(col === "datum" ? "desc" : "asc"); }

  }

  return (

    <main className="admin-prep" role="main">

      <div className="admin-prep\_\_head">

        <div>

          <p className="eyebrow">Administracija</p>

          <h1>Preporuke — pregled, uređivanje i brisanje</h1>

          <p className="muted">Ova strana je dostupna samo administratorima.</p>

        </div>

        <div className="admin-prep\_\_actions">

          <button className="btn btn--ghost" onClick={load}>Osveži</button>

        </div>

      </div>

      {/\* FILTERS \*/}

      <section className="admin-prep\_\_filters" aria-label="Filteri">

        <div className="filters-grid">

          <div className="filter">

            <label className="filter-label">Pretraga</label>

            <input

              value={q}

              onChange={e=>{ setQ(e.target.value); setPage(1); }}

              placeholder="Korisnik, sala, događaj, ID…"

            />

          </div>

          <div className="filter">

            <label className="filter-label">Datum od</label>

            <input type="date" value={dateFrom} onChange={e=>{ setDateFrom(e.target.value); setPage(1); }} />

          </div>

          <div className="filter">

            <label className="filter-label">Datum do</label>

            <input type="date" value={dateTo} onChange={e=>{ setDateTo(e.target.value); setPage(1); }} />

          </div>

          <div className="filter">

            <label className="filter-label">Sortiraj</label>

            <select value={sortBy} onChange={e=>setSortBy(e.target.value)}>

              <option value="datum">Datum</option>

              <option value="vreme\_od">Vreme od</option>

              <option value="korisnik">Korisnik</option>

              <option value="sala">Sala</option>

              <option value="dogadjaj">Događaj</option>

            </select>

          </div>

          <div className="filter">

            <label className="filter-label">Smer</label>

            <select value={sortDir} onChange={e=>setSortDir(e.target.value)}>

              <option value="asc">Rastuće</option>

              <option value="desc">Opadajuće</option>

            </select>

          </div>

        </div>

      </section>

      {msg && <div className="alert">{msg}</div>}

      {loading && <div className="alert">Učitavanje…</div>}

      {/\* TABLE \*/}

      <div className="card">

        <div className="table-wrap">

          <table className="tbl admin-prep\_\_table">

            <thead>

              <tr>

                <th>ID</th>

                <th className={`sortable ${sortBy==="korisnik"?`sorted ${sortDir}`:""}`} onClick={()=>toggleSort("korisnik")}>Korisnik</th>

                <th className={`sortable ${sortBy==="sala"?`sorted ${sortDir}`:""}`} onClick={()=>toggleSort("sala")}>Sala</th>

                <th className={`sortable ${sortBy==="dogadjaj"?`sorted ${sortDir}`:""}`} onClick={()=>toggleSort("dogadjaj")}>Događaj</th>

                <th className={`sortable ${sortBy==="datum"?`sorted ${sortDir}`:""}`} onClick={()=>toggleSort("datum")}>Datum</th>

                <th className={`sortable ${sortBy==="vreme\_od"?`sorted ${sortDir}`:""}`} onClick={()=>toggleSort("vreme\_od")}>Vreme</th>

                <th>Akcije</th>

              </tr>

            </thead>

            <tbody>

              {pageItems.length === 0 ? (

                <tr><td colSpan={7} className="center muted">Nema preporuka za prikaz.</td></tr>

              ) : pageItems.map(r => {

                const isEditing = editId === r.id;

                const userLabel = r.user\_name || `#${r.user\_id}`;

                const salaLabel = r.sala\_naziv || `sala\_id: ${r.sala\_id}`;

                return (

                  <tr key={r.id}>

                    <td className="muted">#{r.id}</td>

                    <td>{userLabel}</td>

                    <td>

                      {isEditing ? (

                        <input

                          className="w-120"

                          value={form.sala\_id}

                          onChange={e=>setForm(f=>({...f, sala\_id:e.target.value}))}

                          placeholder="sala\_id"

                        />

                      ) : salaLabel}

                    </td>

                    <td>

                      {isEditing ? (

                        <input

                          className="w-160"

                          value={form.tip\_dogadjaja}

                          onChange={e=>setForm(f=>({...f, tip\_dogadjaja:e.target.value}))}

                          placeholder="npr. sastanak"

                        />

                      ) : (r.tip\_dogadjaja || "—")}

                    </td>

                    <td>

                      {isEditing ? (

                        <input

                          type="date"

                          value={form.datum}

                          onChange={e=>setForm(f=>({...f, datum:e.target.value}))}

                        />

                      ) : (toISO(r.datum) || "—")}

                    </td>

                    <td>

                      {isEditing ? (

                        <div className="field-row">

                          <input

                            type="time"

                            value={form.vreme\_od}

                            onChange={e=>setForm(f=>({...f, vreme\_od:e.target.value}))}

                          />

                          <span>–</span>

                          <input

                            type="time"

                            value={form.vreme\_do}

                            onChange={e=>setForm(f=>({...f, vreme\_do:e.target.value}))}

                          />

                        </div>

                      ) : (

                        <span>{hhmm(r.vreme\_od)}–{hhmm(r.vreme\_do)}</span>

                      )}

                    </td>

                    <td className="right">

                      {!isEditing ? (

                        <>

                          <button className="btn btn--ghost" onClick={()=>startEdit(r)}>Uredi</button>

                          <button

                            className="btn btn--danger"

                            disabled={deletingId===r.id}

                            onClick={()=>remove(r.id)}

                          >

                            {deletingId===r.id ? "Brišem…" : "Obriši"}

                          </button>

                        </>

                      ) : (

                        <>

                          <button

                            className="btn btn--primary"

                            disabled={saving}

                            onClick={()=>saveEdit(r.id)}

                          >

                            {saving ? "Čuvam…" : "Sačuvaj"}

                          </button>

                          <button className="btn btn--ghost" onClick={cancelEdit}>Otkaži</button>

                        </>

                      )}

                    </td>

                  </tr>

                );

              })}

            </tbody>

          </table>

        </div>

      </div>

      {/\* PAGINATION \*/}

      <div className="pagination">

        <div className="page-size">

          <label>Prikaži:</label>

          <select value={perPage} onChange={e=>{setPerPage(+e.target.value); setPage(1);}}>

            {[5,10,20,50,100].map(n=><option key={n} value={n}>{n}/str</option>)}

          </select>

        </div>

        <div className="page-info"><span>Prikaz {total===0?0:start+1}–{end} od {total}</span></div>

        <div className="page-ctrls">

          <button className="btn btn--ghost" onClick={()=>setPage(1)} disabled={cur===1}>« Prva</button>

          <button className="btn btn--ghost" onClick={()=>setPage(p=>Math.max(1,p-1))} disabled={cur===1}>‹ Prethodna</button>

          <button className="btn btn--ghost" onClick={()=>setPage(p=>Math.min(totalPages,p+1))} disabled={cur===totalPages}>Sledeća ›</button>

          <button className="btn btn--ghost" onClick={()=>setPage(totalPages)} disabled={cur===totalPages}>Poslednja »</button>

        </div>

      </div>

    </main>

  );

}

##### AdminRezervacije

Ova React stranica je administrativni panel za pregled i upravljanje rezervacijama. Na vrhu drži stanje za podatke (rows) i metapodatke paginacije (pages: trenutna strana, stavki po strani, ukupan broj, poslednja strana), kao i indikatore učitavanja i poruke. Pored liste rezervacija, unapred učitava i katalog sala (/sale) kako bi mogla da prikazuje čitljiva imena sala umesto samo ID-eva; pomoćna funkcija salaName pretražuje taj lokalni cache da bi za dati sala\_id vratila naziv.

Filtriranje je rešeno posednim stanjima: tekstualna pretraga po tipu događaja (q), filter sala (fSala), status rezervacije (fStatus) i datum (fDate). Funkcija toQuery iz tih vrednosti generiše query string, a load poziva backend rutu za paginirani spisak — GET /rezervacije/paginacija?page=…&per\_page=…&… — i popunjava tabelu i paginacione podatke na osnovu standardnog Laravel paginator odgovora. Svaka promena filtera automatski trigeruje refetch (kroz useEffect), a dodatno postoje i kontrole za ručno osvežavanje trenutne strane.

Tabelarni prikaz prikazuje osnovne kolone (ID, sala, datum, vreme, tip događaja, status) uz inline kontrolu statusa i checkboxe za masovni izbor. Pored naziva statusa renderuje se i select koji, na promenu, pokreće changeStatus: optimistički menja status u UI-ju i šalje PUT /rezervacije/{id}; u slučaju greške vrši rollback na prethodno stanje. Brisanje jedne rezervacije (removeRow) radi sa potvrdom, optimističkim uklanjanjem reda i server pozivom DELETE /rezervacije/{id}; ako back-end odbije, vraća prethodnu listu.

Za rad sa više stavki odjednom, tu je „bulk” akcija za promenu statusa. Izbor je preko checkboxa (pojedinačni i „select all”), a bulkUpdate optimistički ažurira sve označene redove, potom paralelno šalje niz PUT zahteva i, ako nešto pođe po zlu, vraća prethodno stanje i obaveštava korisnika. Na taj način UI ostaje responsivan, a korisnik odmah vidi efekte svoje akcije, uz sigurnosnu mrežu u slučaju neuspeha servera.

Izvoz u CSV je obezbeđen dugmetom „Izvezi CSV”. Funkcija exportCsv konstruiše iste filtere u query stringu i poziva GET /rezervacije/export/csv sa responseType: "blob". Zatim iz zaglavlja pokušava da izvuče smislen naziv fajla, formira Blob, kreira privremeni a link i programatski pokreće preuzimanje — standardni obrazac za „file download” u browseru bez dodatnih biblioteka.

Paginacija je server-side: kontrole na dnu (prva/poslednja, nazad/napred i izbor „po strani”) zapravo pozivaju load sa traženom stranom ili novim per\_page. Komponenta vodi računa da resetuje selekciju (selected) posle svakog učitavanja, kako bi se izbegle zabune sa starim izborima. Uređivanje pojedinačne rezervacije je rešeno modal formom EditModal (otvara je „Uredi”), koja omogućava izmenu sale, datuma, vremena, tipa događaja i statusa, pa poziva PUT /rezervacije/{id} i po uspehu osveži tekuću stranu.

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

import api from "../api/client";

import "./admin-reservations.css";

/\* helperi \*/

const fmtTime = (t) => (t || "").slice(0,5);

const toQuery = (obj) =>

  Object.entries(obj)

    .filter(([,v]) => v !== undefined && v !== null && v !== "" && v !== "all")

    .map(([k,v]) => `${encodeURIComponent(k)}=${encodeURIComponent(v)}`)

    .join("&");

const STATUS\_OPTS = [

  { value: "pending",  label: "Na čekanju" },

  { value: "approved", label: "Odobrena" },

  { value: "rejected", label: "Odbijena" },

  { value: "cancelled", label: "Otkazana" },

];

export default function AdminReservations() {

  /\* data state \*/

  const [rows, setRows] = useState([]);

  const [pages, setPages] = useState({ page: 1, per\_page: 10, total: 0, last\_page: 1 });

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

  const [msg, setMsg] = useState("");

  /\* lists for filters \*/

  const [sale, setSale] = useState([]);

  /\* filters \*/

  const [q, setQ] = useState("");

  const [fSala, setFSala] = useState("all");

  const [fStatus, setFStatus] = useState("all");

  const [fDate, setFDate] = useState("");

  /\* selection for bulk actions \*/

  const [selected, setSelected] = useState(new Set());

  /\* edit modal \*/

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

  const loadSale = useCallback(async ()=>{

    try {

      const sRes = await api.get("/sale");

      setSale(sRes.data?.data || sRes.data || []);

    } catch {}

  },[]);

  useEffect(()=>{ loadSale(); }, [loadSale]);

const load = useCallback(async (page = pages.page, per\_page = pages.per\_page) => {

  setLoading(true); setMsg("");

  const query = {

    page, per\_page,

    tip\_dogadjaja: q || undefined,

    sala\_id: fSala !== "all" ? fSala : undefined,

    status: fStatus !== "all" ? fStatus : undefined,

    datum: fDate || undefined,

  };

  const url = "/rezervacije/paginacija?" + toQuery(query);

  try {

    const res = await api.get(url);

    const data = res.data;                 // <- ceo paginator objekat

    const rowsArr = Array.isArray(data.data) ? data.data : []; // <- uvek niz

    setRows(rowsArr);

    setPages({

      page: Number(data.current\_page ?? page),

      per\_page: Number(data.per\_page ?? per\_page), // backend vraća string "10"

      total: Number(data.total ?? rowsArr.length),

      last\_page: Number(data.last\_page ?? 1),

    });

  } catch (e) {

    setMsg("Neuspešno učitavanje rezervacija.");

    setRows([]);

  } finally {

    setLoading(false);

    setSelected(new Set());

  }

}, [q, fSala, fStatus, fDate, pages.page, pages.per\_page]);

  useEffect(()=>{ load(1, pages.per\_page); }, [q, fSala, fStatus, fDate]); // refetch na promenu filtera

  const refreshSamePage = () => load(pages.page, pages.per\_page);

  /\* actions \*/

  const changeStatus = async (row, status) => {

    const prev = rows;

    setRows(r => r.map(x => x.id === row.id ? { ...x, status } : x));

    try {

      await api.put(`/rezervacije/${row.id}`, { ...row, status });

    } catch {

      setRows(prev); // rollback

    }

  };

  const removeRow = async (row) => {

    if (!window.confirm("Obrisati rezervaciju #" + row.id + "?")) return;

    const prev = rows;

    setRows(r => r.filter(x => x.id !== row.id));

    try {

      await api.delete(`/rezervacije/${row.id}`);

      refreshSamePage();

    } catch {

      setRows(prev);

      alert("Brisanje nije uspelo.");

    }

  };

  const bulkUpdate = async (status) => {

    if (selected.size === 0) return;

    if (!window.confirm(`Promeni status za ${selected.size} rezervacija?`)) return;

    // optimistički

    const ids = Array.from(selected);

    const prev = rows;

    setRows(r => r.map(x => ids.includes(x.id) ? { ...x, status } : x));

    try {

      await Promise.all(ids.map(id => {

        const row = rows.find(x => x.id === id);

        return api.put(`/rezervacije/${id}`, { ...row, status });

      }));

      setSelected(new Set());

    } catch {

      setRows(prev);

      alert("Bulk izmena nije uspela.");

    }

  };

    const exportCsv = async () => {

    const query = {

        tip\_dogadjaja: q || undefined,

        sala\_id: fSala !== "all" ? fSala : undefined,

        status: fStatus !== "all" ? fStatus : undefined,

        datum: fDate || undefined,

    };

    const url = "/rezervacije/export/csv?" + toQuery(query);

    try {

        const res = await api.get(url, { responseType: "blob" });

        // napravi Blob i smisleno ime fajla iz header-a (ako ga backend šalje)

        const blob = new Blob([res.data], { type: "text/csv;charset=utf-8;" });

        let filename = "rezervacije.csv";

        const cd = res.headers["content-disposition"];

        if (cd) {

        const m = cd.match(/filename\\*?=(?:UTF-8'')?["']?([^"';]+)["']?/i);

        if (m && m[1]) filename = decodeURIComponent(m[1]);

        }

        // triggeruj download

        const link = document.createElement("a");

        const objUrl = window.URL.createObjectURL(blob);

        link.href = objUrl;

        link.download = filename;

        document.body.appendChild(link);

        link.click();

        link.remove();

        window.URL.revokeObjectURL(objUrl);

    } catch (e) {

        console.error(e);

        alert("Izvoz nije uspeo.");

    }

    };

  const toggleOne = (id) => {

    setSelected(s => {

      const n = new Set(s);

      if (n.has(id)) n.delete(id); else n.add(id);

      return n;

    });

  };

  const toggleAll = (checked) => {

    if (!checked) { setSelected(new Set()); return; }

    setSelected(new Set(rows.map(r => r.id)));

  };

  const pageTo = (p) => {

    const clamp = Math.max(1, Math.min(pages.last\_page, p));

    load(clamp, pages.per\_page);

  };

  const perPageChange = (n) => {

    load(1, Number(n));

  };

  /\* UI helpers \*/

  const salaName = useCallback((id) => {

    const s = sale.find(x => String(x.id) === String(id));

    return s?.naziv || `#${id}`;

  }, [sale]);

  return (

    <main className="admin-res">

      <div className="admin-head">

        <div>

          <p className="eyebrow">Administracija</p>

          <h1>Rezervacije — pregled i upravljanje</h1>

          <p className="muted">Pretraga, filteri, uređivanje, odobravanje, brisanje i izvoz u CSV.</p>

        </div>

        <div className="head-actions">

          <button className="btn" onClick={exportCsv}>Izvezi CSV</button>

          <button className="btn btn--ghost" onClick={refreshSamePage}>Osveži</button>

        </div>

      </div>

      {/\* FILTERI \*/}

      <section className="filters">

        <div className="filters-grid">

          <div className="filter">

            <label>Pretraga po tipu događaja</label>

            <input value={q} onChange={(e)=>setQ(e.target.value)} placeholder="npr. konferencija" />

          </div>

          <div className="filter">

            <label>Sala</label>

            <select value={fSala} onChange={(e)=>setFSala(e.target.value)}>

              <option value="all">Sve sale</option>

              {sale.map(s => <option key={s.id} value={s.id}>{s.naziv}</option>)}

            </select>

          </div>

          <div className="filter">

            <label>Status</label>

            <select value={fStatus} onChange={(e)=>setFStatus(e.target.value)}>

              <option value="all">Svi statusi</option>

              {STATUS\_OPTS.map(o => <option key={o.value} value={o.value}>{o.label}</option>)}

            </select>

          </div>

          <div className="filter">

            <label>Datum</label>

            <input type="date" value={fDate} onChange={(e)=>setFDate(e.target.value)} />

          </div>

          <div className="filter filter-actions">

            <div className="bulk">

              <span>Bulk status:</span>

              <select onChange={(e)=> e.target.value && bulkUpdate(e.target.value)} defaultValue="">

                <option value="" disabled>— izaberi —</option>

                {STATUS\_OPTS.map(o => <option key={o.value} value={o.value}>{o.label}</option>)}

              </select>

            </div>

          </div>

        </div>

      </section>

      {msg && <div className="alert">{msg}</div>}

      {loading && <div className="alert">Učitavanje…</div>}

      {/\* TABELA \*/}

      <div className="card table-wrap">

        <table className="tbl">

          <thead>

            <tr>

              <th className="center" style={{width:36}}>

                <input

                  type="checkbox"

                  checked={selected.size === rows.length && rows.length>0}

                  onChange={(e)=>toggleAll(e.target.checked)}

                />

              </th>

              <th>ID</th>

              <th>Sala</th>

              <th>Datum</th>

              <th>Vreme</th>

              <th>Tip događaja</th>

              <th>Status</th>

              <th className="right">Akcije</th>

            </tr>

          </thead>

          <tbody>

            {rows.length === 0 && (

              <tr>

                <td colSpan="8" className="center muted">Nema rezultata.</td>

              </tr>

            )}

            {rows.map(row => (

              <tr key={row.id}>

                <td className="center">

                  <input

                    type="checkbox"

                    checked={selected.has(row.id)}

                    onChange={()=>toggleOne(row.id)}

                  />

                </td>

                <td>{row.id}</td>

                <td title={`Sala ID: ${row.sala\_id}`}>{salaName(row.sala\_id)}</td>

                <td>{row.datum}</td>

                <td>{fmtTime(row.vreme\_od)}–{fmtTime(row.vreme\_do)}</td>

                <td>{row.tip\_dogadjaja}</td>

                <td>

                  <span className={`badge ${row.status}`}>

                    {STATUS\_OPTS.find(o=>o.value===row.status)?.label || row.status}

                  </span>

                  <select

                    className="inline-status"

                    value={row.status}

                    onChange={(e)=>changeStatus(row, e.target.value)}

                  >

                    {STATUS\_OPTS.map(o => <option key={o.value} value={o.value}>{o.label}</option>)}

                  </select>

                </td>

                <td className="right actions">

                  <button className="btn btn--ghost" onClick={()=>setEditing(row)}>Uredi</button>

                  <button className="btn btn--danger" onClick={()=>removeRow(row)}>Obriši</button>

                </td>

              </tr>

            ))}

          </tbody>

        </table>

      </div>

      {/\* PAGINACIJA \*/}

      <div className="pagination">

        <div className="page-size">

          <span>Po strani:</span>

          <select value={pages.per\_page} onChange={(e)=>perPageChange(e.target.value)}>

            {[10,20,30,50].map(n => <option key={n} value={n}>{n}</option>)}

          </select>

        </div>

        <div className="page-ctrls">

          <button className="btn" onClick={()=>pageTo(1)} disabled={pages.page<=1}>« Prva</button>

          <button className="btn" onClick={()=>pageTo(pages.page-1)} disabled={pages.page<=1}>‹ Nazad</button>

          <span className="page-info">Strana {pages.page} / {pages.last\_page}</span>

          <button className="btn" onClick={()=>pageTo(pages.page+1)} disabled={pages.page>=pages.last\_page}>Napred ›</button>

          <button className="btn" onClick={()=>pageTo(pages.last\_page)} disabled={pages.page>=pages.last\_page}>Poslednja »</button>

        </div>

      </div>

      {/\* MODAL: Edit rezervacije \*/}

      {editing && (

        <EditModal

          initial={editing}

          sale={sale}

          onClose={()=>setEditing(null)}

          onSave={async (data)=>{

            try {

              await api.put(`/rezervacije/${editing.id}`, data);

              setEditing(null);

              refreshSamePage();

            } catch {

              alert("Čuvanje nije uspelo.");

            }

          }}

        />

      )}

    </main>

  );

}

/\* ---------- Edit modal ---------- \*/

function EditModal({ initial, sale, onClose, onSave }) {

  const [form, setForm] = useState({

    sala\_id: initial.sala\_id,

    datum: initial.datum,

    vreme\_od: initial.vreme\_od?.slice(0,5) || "",

    vreme\_do: initial.vreme\_do?.slice(0,5) || "",

    tip\_dogadjaja: initial.tip\_dogadjaja || "",

    status: initial.status || "pending",

  });

  const [saving, setSaving] = useState(false);

  const submit = async (e) => {

    e.preventDefault();

    setSaving(true);

    try {

      await onSave({

        ...form,

        vreme\_od: form.vreme\_od,

        vreme\_do: form.vreme\_do,

      });

    } finally {

      setSaving(false);

    }

  };

  return (

    <div className="modal" role="dialog" aria-modal="true">

      <div className="modal-card">

        <h3>Uredi rezervaciju #{initial.id}</h3>

        <form onSubmit={submit} className="edit-form">

          <div className="form-row">

            <label>Sala</label>

            <select

              value={form.sala\_id}

              onChange={(e)=>setForm(f=>({...f, sala\_id: e.target.value}))}

              required

            >

              {sale.map(s => <option key={s.id} value={s.id}>{s.naziv}</option>)}

            </select>

          </div>

          <div className="form-row">

            <label>Datum</label>

            <input type="date" value={form.datum} onChange={(e)=>setForm(f=>({...f, datum: e.target.value}))} required />

          </div>

          <div className="form-row two">

            <div>

              <label>Vreme od</label>

              <input type="time" value={form.vreme\_od} onChange={(e)=>setForm(f=>({...f, vreme\_od: e.target.value}))} required />

            </div>

            <div>

              <label>Vreme do</label>

              <input type="time" value={form.vreme\_do} onChange={(e)=>setForm(f=>({...f, vreme\_do: e.target.value}))} required />

            </div>

          </div>

          <div className="form-row">

            <label>Tip događaja</label>

            <input value={form.tip\_dogadjaja} onChange={(e)=>setForm(f=>({...f, tip\_dogadjaja: e.target.value}))} required />

          </div>

          <div className="form-row">

            <label>Status</label>

            <select value={form.status} onChange={(e)=>setForm(f=>({...f, status: e.target.value}))}>

              {STATUS\_OPTS.map(o => <option key={o.value} value={o.value}>{o.label}</option>)}

            </select>

          </div>

          <div className="form-actions">

            <button type="button" className="btn btn--ghost" onClick={onClose}>Otkaži</button>

            <button type="submit" className="btn" disabled={saving}>{saving ? "Čuvam…" : "Sačuvaj"}</button>

          </div>

        </form>

      </div>

    </div>

  );

}

#### React Rutiranje

Ovaj fajl postavlja klijentsko rutiranje u React aplikaciji pomoću biblioteke react-router-dom. Na samom vrhu se koristi BrowserRouter, koji „presreće” promene URL-a (HTML5 History API) i, umesto punog osvežavanja stranice, prikazuje pravu React komponentu za zadatu putanju. To aplikaciji daje ponašanje SPA (Single-Page Application): kretanje kroz stranice je brzo, bez ponovnog učitavanja.

Unutar BrowserRouter-a nalazi se AuthProvider, kontekst koji drži informacije o ulogovanom korisniku i ulozi (admin, menadžer, korisnik). Pošto obuhvata ceo deo sa rutama, sve komponente u nastavku mogu da koriste taj kontekst (npr. ProtectedRoute), a Navbar i Breadcrumbs su postavljeni iznad <Routes> kako bi se uvek renderovali, bez obzira na aktivnu rutu.

Blok <Routes> sadrži pojedinačne <Route> definicije, koje mapiraju URL putanje na odgovarajuće React stranice. „Javne” rute (početna, login, registracija, katalog, mapa) dostupne su svima. Za „zaštićene” rute koristi se komponenta ProtectedRoute, koja proverava prava pristupa:

* requireAdminOrManager dozvoljava ulaz adminu i menadžeru (npr. /sale).
* requireAdmin dozvoljava samo adminu (npr. dashboard /admin, administracija rezervacija i preporuka).

Takav raspored implementira kontrolu pristupa na nivou klijenta: ako korisnik nema odgovarajuću ulogu ili nije ulogovan, ProtectedRoute ga može preusmeriti na login ili pokazati poruku o zabrani pristupa. Pored toga, login i registracija primaju redirectTo="/sale", što znači da će, nakon uspešne autentikacije, korisnik biti automatski preusmeren na administraciju sala.

Tu su i rute za reset lozinke: /lozinka/posalji za stranicu slanja linka i /lozinka/reset za formu promene lozinke (otvara se iz email linka). Na kraju, fallback ruta sa path="\*" koristi <Navigate to="/" replace /> da sve nepoznate putanje vrati na početnu – elegantan način da se izbegnu „mrtvi linkovi”.

import "./App.css";

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

import LandingPage from "./pages/LandingPage";

import LoginPage from "./pages/LoginPage";

import RegisterPage from "./pages/RegisterPage";

import Navbar from "./components/Navbar";

import SalesAdminPage from "./pages/SalesAdminPage";

import { AuthProvider } from "./context/AuthContext";

import ProtectedRoute from "./components/ProtectedRoute";

import Breadcrumbs from "./components/Breadcrumbs";

import SaleCatalog from "./pages/SaleCatalog";

import FloorPlan from "./pages/FloorPlan";

import AdminDashboardPage from "./pages/AdminDashboardPage";

import AdminReservations from "./pages/AdminReservations";

import AdminPreporuke from "./pages/AdminPreporuke";

import ForgotPasswordPage from "./pages/ForgotPasswordPage";

import ResetPasswordPage from "./pages/ResetPasswordPage";

function App() {

  return (

    <BrowserRouter>

      <AuthProvider>

        <Navbar />

        <Breadcrumbs />

        <Routes>

          {/\* Public \*/}

          <Route path="/" element={<LandingPage />} />

          <Route path="/login" element={<LoginPage redirectTo="/sale" />} />

          <Route path="/registracija" element={<RegisterPage redirectTo="/sale" />} />

          <Route path="/katalog" element={<SaleCatalog />} />

          <Route path="/floor-plan" element={<FloorPlan />} />

          {/\* Admin/Manager \*/}

          <Route

            path="/sale"

            element={

              <ProtectedRoute requireAdminOrManager>

                <SalesAdminPage />

              </ProtectedRoute>

            }

          />

          {/\* Admin-only \*/}

          <Route

            path="/admin"

            element={

              <ProtectedRoute requireAdmin>

                <AdminDashboardPage />

              </ProtectedRoute>

            }

          />

          <Route

            path="/admin/rezervacije"

            element={

              <ProtectedRoute requireAdmin>

                <AdminReservations />

              </ProtectedRoute>

            }

          />

          <Route

            path="/admin/preporuke"

            element={

              <ProtectedRoute requireAdmin>

                <AdminPreporuke />

              </ProtectedRoute>

            }

          />

          <Route path="/lozinka/posalji" element={<ForgotPasswordPage />} />

          <Route path="/lozinka/reset"  element={<ResetPasswordPage />} />

          {/\* Fallback \*/}

          <Route path="/app" element={<LandingPage />} />

          <Route path="\*" element={<Navigate to="/" replace />} />

        </Routes>

      </AuthProvider>

    </BrowserRouter>

  );

}

export default App;

#### Laravel i React

Ovaj fajl predstavlja centralnu tačku veze između React klijenta i Laravel API-ja. Kreira se jedan Axios klijent kojem se zadaje osnovni URL ka backendu, pa svaki poziv iz frontenda ide na isti API koren bez ponavljanja pune adrese. Tako, na primer, api.get("/sale") automatski gađa rutu http://localhost:8000/api/sale koju u Laravelu definišeš u api.php, a odgovor stiže kao JSON koji komponente dalje prikazuju.

Pre slanja svakog zahteva radi se ubacivanje Bearer tokena u HTTP zaglavlje. Token se čuva u localStorage nakon prijave, a interceptor ga umeće u Authorization zaglavlje tako da backend odmah zna ko je korisnik i može da primeni autentikaciju i autorizaciju preko Laravel middleware-a. Na taj način ne moraš po komponentama ručno da dodaješ kredencijale; sve ide kroz jedan konfigurisan klijent.

Presreću se i odgovori sa servera kako bi se uredno rukovalo greškama sesije. Ako Laravel vrati status 401 (istekao ili nevažeći token), klijent briše token i podatke o korisniku iz localStorage, čime frontend prelazi u „odjavljen” način rada i zaštićene rute prestaju da budu dostupne dok se ponovo ne prijaviš. Time se izbegava „zaglavljena” sesija i UI ostaje u dosluhu sa stvarnim stanjem autentikacije na serveru.

Ovakva postavka praktično „lepi” React i Laravel: komponente pozivaju api.\*, Axios automatski dodaje autentikaciju, Laravel rute iz api.php obrađuju zahtev u kontrolerima i vraćaju JSON, a frontend renderuje rezultat ili reaguje na grešku. Po želji, baseURL se može izvući u .env promenljivu radi lakšeg prebacivanja između lokalnog i produkcionog okruženja, ali suština ostaje ista - jedan konfigurisani klijent brine o adresama, tokenima i globalnom rukovanju sesijom.

import axios from "axios";

const api = axios.create({

  baseURL:  "http://localhost:8000/api",

  withCredentials: false,

});

// U svaki request ubacujemo Bearer token iz localStorage

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

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

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

  return config;

});

// Ako dobijemo 401, čistimo keš

api.interceptors.response.use(

  (res) => res,

  (err) => {

    if (err?.response?.status === 401) {

      localStorage.removeItem("token");

      localStorage.removeItem("user");

    }

    return Promise.reject(err);

  }

);

export default api;

### Projektovanje strukture softverskog sistema

U okviru projektovanja strukture softverskog sistema definiše se raspored i organizacija svih glavnih komponenti aplikacije, kao što su frontend, backend, baza podataka i način njihove međusobne komunikacije. Cilj ove faze je da se obezbedi jasan i modularan raspored sistema koji omogućava laku održivost, testiranje i nadogradnju.

#### Migracije

Migracije u Laravelu su mehanizam za verzionisanje šeme baze podataka: svaka migracija je PHP klasa (sa metodama up() i down()) koja opisuje koje tabele, kolone, indekse i spoljne ključeve treba kreirati, izmeniti ili obrisati. Umesto ručnog pisanja SQL-a, koristi se fluentni Schema Builder, pa se promenama upravlja kroz repozitorijum koda – tim može dosledno primenjivati strukturalne izmene na svim okruženjima komandom php artisan migrate, a po potrebi ih vraćati migrate:rollback. Migracije obezbeđuju trag promena, olakšavaju saradnju i automatizuju pripremu baze (često zajedno sa „seed” klasama za inicijalne podatke), što čini razvoj i deploy pouzdanijim i ponovljivim.

Migracija pod nazivom **CreateUsersTable** kreira tabelu users koja sadrži sledeće kolone: id za jedinstvenu identifikaciju korisnika, name za čuvanje imena korisnika, email kao jedinstveno polje za čuvanje email adrese, email\_verified\_at za evidentiranje datuma verifikacije emaila (koje može biti prazno), password za čuvanje korisničke šifre, remember\_token za upravljanje tokenima sesije, kao i standardne kolone created\_at i updated\_at za evidenciju vremena kreiranja i ažuriranja zapisa.

Migracija pod nazivom **CreateSalasTable** kreira tabelu salas sa sledećim kolonama: id za jedinstvenu identifikaciju svake sale, naziv za naziv sale, tip za definisanje vrste sale (npr. učionica ili konferencijska sala), kapacitet za broj osoba koje sala može primiti, opis kao opcionalno polje za dodatne informacije o sali, status za označavanje dostupnosti sale (podrazumevano true znači da je sala dostupna) i standardne kolone created\_at i updated\_at za praćenje vremena kreiranja i poslednjeg ažuriranja zapisa.

Migracija pod nazivom **CreateRezervacijasTable** kreira tabelu rezervacijas sa sledećim kolonama: id za jedinstvenu identifikaciju svake rezervacije, sala\_id kao spoljni ključ koji se povezuje sa tabelom salas i briše se kaskadno pri brisanju sale, user\_id kao spoljni ključ povezan sa tabelom users koji se takođe briše kaskadno pri brisanju korisnika, datum za datum rezervacije, vreme\_od i vreme\_do za vremenski raspon rezervacije, tip\_dogadjaja za vrstu događaja (npr. konferencija), status za trenutno stanje rezervacije sa podrazumevanom vrednošću na čekanju, i standardne kolone created\_at i updated\_at za praćenje vremena kreiranja i poslednjeg ažuriranja zapisa.

Migracija pod nazivom **CreatePreporukasTable** kreira tabelu preporukas sa sledećim kolonama: id za jedinstvenu identifikaciju svake preporuke, sala\_id kao spoljni ključ povezan sa tabelom salas koji se briše kaskadno pri brisanju sale, user\_id kao spoljni ključ povezan sa tabelom users koji se takođe briše kaskadno pri brisanju korisnika, datum za datum preporuke, vreme\_od i vreme\_do za vremenski interval za koji se preporuka odnosi, tip\_dogadjaja za vrstu događaja za koji je sala preporučena, i standardne kolone created\_at i updated\_at za praćenje vremena kreiranja i poslednjeg ažuriranja zapisa.

Migracija pod nazivom **AddLokacijaToSalasTable** dodaje novu kolonu lokacija u tabelu salas. Ova kolona je tipa string, može biti prazna (nullable) i smeštena je odmah nakon kolone kapacitet. Kolona služi za čuvanje informacije o lokaciji sale.

Migracija pod nazivom **AddIndexToNazivInSalasTable** dodaje indeks na kolonu naziv u tabeli salas. Ova optimizacija omogućava brže pretraživanje i filtriranje podataka po nazivu sale.

Migracija pod nazivom **DropLokacijaFromSalasTable** uklanja kolonu lokacija iz tabele salas, čime se trajno brišu podaci o lokaciji sala koji su prethodno bili sadržani u toj koloni.

Migracija pod nazivom **AddManagerIdToSalasTable** dodaje kolonu manager\_id u tabelu salas. Ova kolona je tipa unsignedBigInteger, može biti prazna (nullable), i smeštena je odmah nakon kolone status. Uspostavlja se spoljni ključ koji povezuje manager\_id sa kolonom id u tabeli users, pri čemu se vrednost postavlja na null u slučaju brisanja povezanog korisnika. Ova kolona omogućava dodeljivanje menadžera svakoj sali.

Migracija pod nazivom **DropForeignKeyFromSalasTable** uklanja strani ključ sa kolone manager\_id u tabeli salas. Ovom promenom se ukida veza između kolone manager\_id u tabeli salas i tabele users, dok sama kolona manager\_id ostaje netaknuta.

Migracija pod nazivom **AddFilePathToSalasTable** dodaje novu kolonu file\_path u tabelu salas. Ova kolona je tipa string, može biti prazna (nullable), i smeštena je odmah nakon kolone status. Kolona file\_path služi za čuvanje putanje do fajla povezanog sa salom, poput slike ili dokumentacije.

#### Modeli

Model **Sala** predstavlja entitet koji opisuje sale, uključujući njihove osnovne karakteristike kao što su naziv, tip, kapacitet, opis, status, putanja do fajla i cena. Polja su definisana u okviru atributa fillable, što omogućava masovno popunjavanje ovih vrednosti tokom kreiranja ili ažuriranja zapisa. Svaka sala može imati više rezervacija, što je predstavljeno relacijom "jedan prema više" sa modelom Rezervacija. Ovo znači da jedna sala može biti povezana sa više različitih termina rezervacija, omogućavajući fleksibilno upravljanje raspoloživošću.

Pored rezervacija, model takođe ima relaciju sa preporukama kroz metodu preporuke, koja takođe uspostavlja vezu "jedan prema više". Ovo omogućava da sala bude povezana sa više preporuka, što je korisno za implementaciju sistema koji podržava recenzije ili predloge za korisnike. Na ovaj način, model Sala funkcioniše kao centralni entitet koji povezuje druge delove aplikacije, omogućavajući detaljno upravljanje i praćenje kako rezervacija, tako i preporuka.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Sala extends Model

{

    use HasFactory;

    protected $fillable = [

        'naziv','tip','kapacitet','opis','status','file\_path','cena',

        'floor','layout\_x','layout\_y','layout\_w','layout\_h',

    ];

    protected $casts = [

        'status'    => 'boolean',

        'kapacitet' => 'integer',

        'floor'     => 'integer',

        'layout\_x'  => 'integer',

        'layout\_y'  => 'integer',

        'layout\_w'  => 'integer',

        'layout\_h'  => 'integer',

        'cena'      => 'decimal:2',

    ];

    public function rezervacije()

    {

        return $this->hasMany(Rezervacija::class);

    }

    public function preporuke()

    {

        return $this->hasMany(Preporuka::class);

    }

}

Model **Rezervacija** predstavlja entitet koji služi za praćenje i upravljanje rezervacijama sala. Ključni atributi uključuju identifikaciju sale putem sala\_id, korisnika koji je napravio rezervaciju kroz user\_id, kao i informacije o datumu i vremenskim intervalima rezervacije, tip događaja i trenutni status rezervacije. Ovi atributi omogućavaju fleksibilno evidentiranje rezervacija sa svim relevantnim detaljima.

Relacija sala povezuje svaku rezervaciju sa određenom salom, čineći je delom relacije "više prema jednom" sa modelom Sala. Na taj način, svaka rezervacija pripada jednoj specifičnoj sali, omogućavajući povezivanje podataka o rezervaciji sa atributima sale, poput kapaciteta ili cene. Relacija user povezuje rezervaciju sa korisnikom koji je kreirao rezervaciju, uspostavljajući vezu "više prema jednom" sa modelom User. Ovo omogućava praćenje korisničkog angažmana i povezivanje rezervacija sa specifičnim profilima korisnika. Model Rezervacija efikasno povezuje korisnike i sale, pružajući centralizovano upravljanje podacima o rezervacijama u aplikaciji.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Rezervacija extends Model

{

    use HasFactory;

    protected $fillable = ['sala\_id', 'user\_id', 'datum', 'vreme\_od', 'vreme\_do', 'tip\_dogadjaja', 'status'];

    public function sala()

    {

        return $this->belongsTo(Sala::class);

    }

    public function user()

    {

        return $this->belongsTo(User::class);

    }

}

Model **Preporuka** koristi se za evidentiranje predloga ili preporuka sala za određene korisnike i događaje. Ključni atributi uključuju identifikaciju sale putem sala\_id, korisnika kome je preporuka namenjena kroz user\_id, kao i informacije o datumu i vremenskom intervalu koji su predloženi za korišćenje sale. Dodatno, polje tip\_dogadjaja omogućava specifikaciju vrste događaja za koji je preporuka relevantna, pružajući kontekst i personalizaciju za korisnika.

Relacija sala povezuje svaku preporuku sa određenom salom, uspostavljajući vezu "više prema jednom" sa modelom Sala. Ovo omogućava da se svaka preporuka odnosi na konkretan prostor, čineći preporuke specifičnim i relevantnim za korisnika. Relacija user povezuje preporuku sa korisnikom kome je ona namenjena, uspostavljajući vezu "više prema jednom" sa modelom User. Ovo omogućava praćenje personalizovanih predloga za svakog korisnika. Model Preporuka efikasno povezuje korisnike, sale i kontekstualne informacije, omogućavajući aplikaciji da generiše personalizovane i korisne preporuke za različite događaje.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Preporuka extends Model

{

    use HasFactory;

    protected $fillable = ['sala\_id', 'user\_id', 'datum', 'vreme\_od', 'vreme\_do', 'tip\_dogadjaja'];

    public function sala()

    {

        return $this->belongsTo(Sala::class);

    }

    public function user()

    {

        return $this->belongsTo(User::class);

    }

}

Model **User** predstavlja osnovni entitet za korisnike sistema i koristi se za autentifikaciju i upravljanje podacima o korisnicima. Atributi koji su definisani u fillable omogućavaju masovno popunjavanje, uključujući name, email i password, što predstavlja osnovne podatke korisnika. Atributi definisani u hidden, poput password i remember\_token, osiguravaju da se osetljivi podaci ne prikazuju u JSON odgovorima ili drugim izlazima. Dodatno, atribut email\_verified\_at koristi se za evidenciju verifikacije email adrese i automatski se konvertuje u instancu datetime zahvaljujući casts svojstvu.

<?php

namespace App\Models;

use App\Notifications\ResetPasswordCustom;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

    use HasFactory, Notifiable, HasApiTokens;

    /\*\*

     \* The attributes that are mass assignable.

     \*

     \* @var array

     \*/

    protected $fillable = [

        'name',

        'email',

        'password',

        'role'

    ];

    /\*\*

     \* The attributes that should be hidden for arrays.

     \*

     \* @var array

     \*/

    protected $hidden = [

        'password',

        'remember\_token',

    ];

    /\*\*

     \* The attributes that should be cast to native types.

     \*

     \* @var array

     \*/

    protected $casts = [

        'email\_verified\_at' => 'datetime',

    ];

    public function sendPasswordResetNotification($token)

    {

        $frontend = config('app.frontend\_url', env('FRONTEND\_URL', 'http://localhost:3000'));

        $url = $frontend . '/lozinka/reset?token=' . $token . '&email=' . urlencode($this->email);

        $this->notify(new ResetPasswordCustom($url));

    }

}

### Projektovanje skladišta podataka

U ovoj etapi rada, polazeći od konceptualnog modela (UML dijagrama klasa), izvedeno je projektovanje relacionog modela baze. Svaka klasa iz UML-a preslikana je u tabelu, njeni atributi u kolone, a veze među klasama u spoljne ključeve koji obezbeđuju povezivanje podataka.

Relaciona šema je realizovana kroz Laravel migracije, što omogućava da se struktura baze kreira i menja programskim putem. U bazi su definisane sve potrebne tabele (entiteti), dok su odnosi među njima implementirani stranim ključevima radi očuvanja referencijalnog integriteta. Ovakav pristup obezbeđuje konzistentnost podataka i olakšava kasnije izvođenje upita i transakcionu obradu unutar aplikacije.

User(id, name, email, password, role, email\_verified\_at, remember\_token, created\_at, updated\_at)  
Sala(id, naziv, tip, kapacitet, opis, status, file\_path, cena, floor, layout\_x, layout\_y, layout\_w, layout\_h, created\_at, updated\_at)  
Rezervacija(id, sala\_id, user\_id, datum, vreme\_od, vreme\_do, tip\_dogadjaja, status, created\_at, updated\_at)  
Preporuka(id, sala\_id, user\_id, datum, vreme\_od, vreme\_do, tip\_dogadjaja, created\_at, updated\_at)

## Faza implementacije

U Laravel delu projekta jezgro koda je u folderu **app**. Unutar **app/Http/Controllers** nalaze se svi kontroleri (npr. SalaController, RezervacijaController, PreporukaController, UserController, PasswordController, AdminDashboardController) koji primaju HTTP zahteve, pozivaju modele/servise i vraćaju JSON odgovore. Modeli su u **app/Models** (Sala, Rezervacija, Preporuka, User) i predstavljaju tabelarne entitete, sa definisanim relacijama i cast-ovima. U **app/Http/Middleware** bi se nalazili slojevi za proveru pristupa, autentikaciju ili uloge (ako su prilagođeni). Laravel-ova konfiguracija je u **config**, dok je sistemski „bootstrapping” u **bootstrap** i **app/Providers** (registracija servisa, event-ova, itd.).

Baza podataka se razvija kroz **database/migrations** (evolucija šeme), **database/seeders** (početni ili demo podaci) i opcione **factories** (generisanje test podataka). Rute API-ja su definisane u **routes/api.php** i direktno mapiraju putanje na kontrolere, uz middleware slojeve (npr. auth:sanctum, provera uloga). U **app/Notifications** je custom notifikacija za reset lozinke, a **resources** može sadržati resource klase (npr. SalaResource, RezervacijaResource) za oblikovanje JSON odgovora. Ovakav raspored jasno odvaja odgovornosti: rute → kontroleri → modeli/servisi → resursi, a migracije/seed-ovi kontrolišu stanje baze preko koda.

Svi front-end fajlovi su u **src**. U **api/client.js** je centralizovan Axios klijent: tu se zadaje baseURL ka Laravel API-ju i interceptori koji automatski dodaju Bearer token i reaguju na 401. UI sloj je podeljen na **components** i **ui**: u **ui** žive bazični, višekratni građevni elementi (npr. Button, Input, Alert, stilovi kao navbar.css, floor-map.css, sale.css), dok u **components** držiš složenije komponente poput Navbar, ProtectedRoute, ReservationForm, SaleForm, FloorMap koje kombinuju više UI elemenata i enkapsuliraju poslovnu logiku ekrana.

Kontekst autentikacije je u **context/AuthContext.jsx** (čuva korisnika/token i nudi hook za uloge), dok su pomoćni React hook-ovi u **hooks** (npr. useSalesData za učitavanje i ponovno učitavanje lista sala). Stranice (route-level ekrani) su u **pages**: LandingPage, LoginPage, RegisterPage, SaleCatalog, FloorPlan, administratorske stranice AdminDashboardPage, AdminReservations, AdminPreporuke, kao i forme za „zaboravljenu” i „reset” lozinku. Svaka stranica obično ima svoj prateći .css fajl. Dodatne pomoćne funkcije ili integracije su u **services** (npr. external.js). Ulazna tačka aplikacije je App.jsx (definiše React Router rute, AuthProvider, globalne komponente poput Navbar i Breadcrumbs), a globalni stilovi su u App.css

Na backendu se poštuje standardna Laravel konvencija (controllers/models/routes/migrations), što olakšava skaliranje, testiranje i onboard novih članova. Na frontendu je razdvajanje **pages → components → ui** namerno: stranice predstavljaju „scene” povezane rutama, komponente enkapsuliraju kompleksne delove funkcionalnosti, a UI folder drži najmanje, neutralne, višekratne gradivne blokove i stilove. **api/client.js** centralizuje HTTP sloj (token, greške), **context** centralizuje stanje autentikacije, a **hooks** čuva ponovnu upotrebljivu logiku učitavanja i transformacije podataka. Ovakva organizacija omogućava da se backend i frontend razvijaju nezavisno, uz jasne granice između prikaza, logike i pristupa podacima.

## Faza testiranja

U ovoj fazi sproveli smo **ručno testiranje** celog sistema, fokusirajući se na funkcionalnu ispravnost, upotrebljivost i stabilnost aplikacije. Testovi su obuhvatili frontend (React) i backend (Laravel) sloj, kao i njihovu integraciju. Scenariji su izvođeni kroz tipične tokove korisnika i administratora, bez automatizovanih testova, ali sa pažljivo definisanim listama provera (checklistama) po modulu.

Testirali smo ključne funkcionalnosti: registraciju, prijavu, odjavu i reset lozinke; pregled kataloga sala, pretragu i filtriranje; kreiranje rezervacije i validacije termina; rad sa preporukama; administraciju sala (kreiranje/izmena/deaktivacija, upload fajla, uređivanje rasporeda na mapi); pregled i odobravanje/odbijanje rezervacija; kao i administratorski dashboard (KPI, grafikoni). Posebna pažnja posvećena je pravima pristupa (uloge: korisnik/menadžer/administrator) i zaštiti ruta.

Na nivou robusnosti, proverili smo ponašanje sistema kod pogrešnih unosa (obavezna polja, format datuma/vremena, kolizija termina), nepostojećih resursa, kao i slučajeva bez podataka. Validacije i poruke o greškama prikazuju se očekivano, a optimistička ažuriranja u administraciji (npr. promena statusa) korektno rade uz rollback u slučaju neuspeha servera. Proverili smo i keširanje/listanje (paginacija), preuzimanje CSV izveštaja, kao i vizuelnu dostupnost ključnih informacija u tabelama i na mapi.

Kompatibilnost je proverena u modernim pregledačima (Chrome/Edge/Firefox, aktuelne verzije) i na različitim rezolucijama ekrana. Interfejs ostaje funkcionalan i čitljiv, a osnovne operacije se izvršavaju bez vidljivih zastoja. Integraciona komunikacija između React klijenta i Laravel API-ja protiče stabilno (JWT/Sanctum token, zaglavlja, statusni kodovi).

Na osnovu svih ručnih provera, **aplikacija funkcioniše ispravno** i pokriva predviđene slučajeve korišćenja. Nismo uočili kritične greške koje bi onemogućile rad sistema. Preporuka za naredne iteracije je uvođenje automatizovanih testova (unit/integracionih/E2E), kao i dodatnih „edge case” scenarija (veći obim podataka, opterećenje, negativni testovi), kako bismo dugoročno održali kvalitet i olakšali buduće izmene.

# ZAKLJUČAK

Realizacijom aplikacije demonstrirano je kako se moderni frontend (React) i backend (Laravel) mogu spojiti u celinu koja korisnicima nudi jednostavan katalog sala, napredne filtere, vizuelnu mapu spratova i brzu rezervaciju, dok administratorima pruža potpun nadzor nad resursima, metrikama i preporukama. Iterativan razvoj i dosledna upotreba migracija, Eloquent modela, resursnih kontrolera i jasno definisanih API ruta doveli su do stabilnog i preglednog rešenja. Ručnim testiranjem ključnih scenarija (registracija, prijava, CRUD nad salama i rezervacijama, izmene layout-a, izvoz, grafikoni) potvrđena je ispravnost funkcionalnosti i spremnost sistema za dalji rad i buduća proširenja (npr. dodatne role, složenije preporuke, integracije sa spoljim servisima). Aplikacija je, zahvaljujući modularnoj organizaciji i dobrim praksama, spremna za održavanje i razvoj novih mogućnosti.

# LITERATURA

* Severance, C. (2012). JavaScript: Designing a Language in 10 Days. Computer. <https://doi.org/10.1109/MC.2012.194>
* Welling, L., & Thomson, L. (2016). PHP and MySQL Web Development (5th ed.). Addison-Wesley.
* Otwell, T. (2023). Laravel Documentation. Laravel LLC. <https://laravel.com/docs>
* React Team. (2023). React – A JavaScript library for building user interfaces. Meta Platforms. https://reactjs.org
* Mozilla Developer Network. (2023). JavaScript Guide. Mozilla Foundation. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
* Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.). Pearson Education.