

Sveučilište Jurja Dobrile u Puli  
Fakultet Informatike

DAVID MAGLICA

**Mobilna aplikacija za rezervaciju raspoloživih mesta u  
ugostiteljskim objektima**

Diplomski rad

Pula, Rujan, 2025. godine  
Sveučilište Jurja Dobrile u Puli  
Fakultet Informatike

DAVID MAGLICA

**Mobilna aplikacija za rezervaciju raspoloživih mesta u  
ugostiteljskim objektima**

Diplomski rad

**JMBAG: 0066306616, redoviti student**

**Studijski smjer: Informatika**

**Kolegij: Izrada informatičkih projekata**

**Znanstveno područje : Društvene znanosti**

**Znanstveno polje : Informacijske i komunikacijske znanosti**

**Znanstvena grana : Informacijski sustavi i informatologija**

**Mentor: doc. dr. sc. Nikola Tanković**

Pula, Rujan, 2025. godine

## Sadržaj

1	Uvod i motivacija .....	1
1.1	Analiza tržišta .....	1
2	Korištene tehnologije .....	3
2.1	Flutter.....	3
2.2	Kotlin .....	8
2.3	Spring Boot .....	9
2.4	H2 .....	10
3	Implementacija projekta .....	13
3.1	Implementacija korisnikog sučelja .....	13
3.2	Implementacija backenda .....	24
4	Korisničke upute .....	33
4.1	Korisničke upute za korištenje mobilne aplikacije.....	33
4.2	Korisničke upute za korištenje web aplikacije (administrativnog sučelja)....	42
5	Zaključak.....	52
6	Literatura .....	53
7	Popis slika .....	55
	Sažetak .....	56
	Abstract .....	57

# 1 Uvod i motivacija

Unatoč sveprisutnoj digitalizaciji, proces rezervacije stolova u ugostiteljskim objektima često se još uvijek temelji na tradicionalnim metodama, telefonskim pozivima, izravnim dolascima ili putem različitih platformi društvenih mreža. Ovakav nekoordiniran pristup stvara značajne operativne probleme kako za korisnike, tako i za vlasnike objekata. Korisnici se suočavaju s nepoznatim kapacitetom ugostiteljskog objekta i dugim vremenom čekanja, dok vlasnici neefikasno upravljaju raspoloživim kapacitetom, što direktno utječe na njihov prihod.

Temeljna motivacija za izradu ovog rada leži upravo u rješavanju ovih problema kroz razvoj jedinstvenog, integriranog sustava. Predloženo rješenje, pod nazivom *SeatSaver*, čini mobilna aplikacija namijenjena krajnjim korisnicima za jednostavno pretraživanje i rezervaciju te web aplikacija koja omogućuje vlasnicima objekata nadzor nad kapacitetom, rezervacijama i analitikom.

Cilj rada je detaljno opisati razvoj cjelovitog sustava, od odabira tehnologija, kroz implementaciju do testiranja konačnog rješenja. Rad će istaknuti kako sinergija navedenih modernih tehnologija omogućuje brz razvoj skalabilnog i korisnički prihvatljivog rješenja za tržište.

## 1.1 Analiza tržišta

Analiza tržišta predstavlja ispitivanje i razumijevanje čimbenika koji oblikuju određeni tržišni segment unutar neke industrije. Ovaj proces uključuje procjenu različitih varijabli koje utječu na uspjeh poslovanja, pružajući uvide koji usmjeravaju strateške odluke. Provođenje analize tržišta ključno je za održavanje konkurentnosti i informiranosti, kako za postojeće, tako i za nove tvrtke. Analiza tržišta omogućuje razumijevanje tko bi mogao biti zainteresiran za ponuđene proizvode ili usluge, što konkurenti nude te što potrošači cijene ili ne cijene [3]. Prije same izrade rješenja, proveden je pregled tržišta s fokusom na postojeće konkurenте, njihove ponude i funkcionalnosti, kako bi se identificirale mogućnosti za pozicioniranje novog proizvoda

te područja u kojima može dodati vrijednost i istaknuti se u odnosu na postojeća rješenja.

### 1.1.1 Slične aplikacije

- **Eat App** je sustav temeljen u oblaku (eng. *cloud-based*) za upravljanje rezervacijama u restoranima, koji integrira upravljanje stolovima, bazom gostiju, analitiku, integracije s POS sustavom i društvenim mrežama [1].
- **Resy** je platforma za upravljanje rezervacijama restorana, komunikaciju s gostima i listom čekanja. Omogućuje fleksibilne rezervacije, upravljanje događajima, napredne analitike i integraciju s POS sustavom [1].
- **OpenTable** je jedna od najpoznatijih platformi za upravljanje rezervacijama, s funkcijama poput upravljanja listom čekanja, inventarom, marketingom i odnosa s gostima [1].
- **SevenRooms** je platforma za rezervacije i iskustvo gosta, koja optimizira rad restorana. Nudi upravljanje stolovima, rezervacijama, kreiranje baze gostiju, personalizirani marketing, integraciju s programom vjernosti (eng. *loyalty program*) te vezu s društvenim mrežama i POS sustavom [1].
- **Google Maps rezervacije** omogućuje korisnicima da rezerviraju stol izravno putem Google pretraživanja ili Google Maps, bez potrebe za napuštanjem Google sučelja. Rezervacije se automatski sinkroniziraju s restoranskim sustavom za upravljanje rezervacijama, ako ga isti ima [2].

Sve navedene platforme imaju za cilj pojednostavnići proces rezervacije, poboljšati iskustvo gostiju i omogućiti ugostiteljskim objektima bolju kontrolu nad kapacitetom i podacima korisnika. Prednosti uključuju integraciju s POS sustavima, upravljanje stolovima, analitiku, CRM funkcionalnosti i povezivanje s društvenim mrežama, što omogućuje učinkovito vođenje objekta i kvalitetniju interakciju s gostima.

Nedostatak Google Maps rezervacija je potreba za aktivnim i optimiziranim Google Business Profileom, što uključuje ažurirane podatke o restoranu, radno vrijeme, slike i opise kako bi gumb za rezervaciju bio vidljiv i funkcionalan. Nedostatci ostalih platformi su komercijalni karakter i veća kompleksnost sustava, što može predstavljati dodatni trošak za ugostiteljske objekte i zahtijeva određenu obuku i tehničko znanje za optimalno korištenje.

## 2 Korištene tehnologije

### 2.1 Flutter

Flutter je skup alata za korisničko sučelje otvorenog koda (eng. *open-source UI toolkit*), odnosno programski okvir i razvojni komplet (eng. *SDK*), koji je razvio Google s ciljem izrade nativno kompiliranih aplikacija za mobilne uređaje, web i *desktop*. Ova tehnologija omogućuje programerima da značajno skrate vrijeme razvoja jer isti kod može biti korišten na više platformi bez potrebe za pisanjem zasebnih aplikacija. Jedna od ključnih značajki Fluttera je *hot reload*, koji omogućuje prikaz promjena u kodu bez gubitka stanja aplikacije, čime se ubrzava proces testiranja.

Flutter aplikacije temelje se na *widgetima*, koji predstavljaju osnovne elemente korisničkog sučelja i omogućuju visoki stupanj prilagodljivosti i modularnosti. Budući da Flutter kompilira aplikacije direktno u strojni kod, performanse su vrlo blizu nativnim rješenjima, što ga čini pogodnim za složene i zahtjevne projekte [4].

Pošto je Google razvio Flutter, omogućena je besprijkorna integracija Googleovih usluga, omogućujući jednostavno korištenje moćnih alata za razvoj, unovčavanje i rast aplikacija. Među glavnim integracijama nalaze se:

- Google AI: omogućuje integraciju generativne umjetne inteligencije poput tekstualne generacije, sažimanja i *chat* funkcionalnosti.
- Firebase: pruža gotove usluge poput baza podataka, autentifikacije i pohrane iste, čime značajno smanjuje potrebu za složenim *backend* razvojem.
- Google Ads: omogućuje jednostavno unovčavanje aplikacija kroz integraciju oglasa u različitim formatima.
- Google Play: centralizira upravljanje prihodima kroz pretplate, digitalne proizvode i premium funkcionalnosti.
- Google Pay i Google Wallet: omogućuju sigurno i jednostavno plaćanje unutar aplikacije, čime se poboljšava iskustvo korisnika.
- Google Maps: podržava geolokacijske funkcionalnosti poput interaktivnih karata, navigacije i drugih prostornih usluga.

## 2.1.1 Životni ciklus Widgeta

Flutter *widgeti* prolaze kroz osam faza tijekom svog postojanja u stablu *widgeta* (eng. *widget tree*) (struktura u kojoj se *widgeti* grade jedan unutar drugog). Poznavanje tih faza pomaže u optimizaciji performansi, boljem upravljanju resursima i izbjegavanju grešaka. Postoje dva osnovna tipa *widgeta* *StatelessWidget* i *StatefulWidget*. *Stateless widgeti* su nepromjenjivi dok njihov roditelj ne uzrokuje novu izgradnju, tj. *rebuild*, dok *Stateful widgeti* sadrže vlastito stanje (eng. *state*) kojeg mogu mijenjati neovisno o roditelju i zadržavaju to stanje tokom *rebuilda* roditelja ako ga roditelj ne promijeni (Bhatt, 2025).

### Primjer *StatelessWidgeta*

```
class My StatelessWidget extends StatelessWidget {  
    final String title;  
  
    const My StatelessWidget({Key? key, required this.title}) : super(key:  
key);  
  
    @override  
    Widget build(BuildContext context) {  
        return Text(title);  
    }  
} (Bhatt, 2025)
```

### Primjer *StatefulWidgeta*

```
class My StatefulWidget extends StatefulWidget {  
    @override  
    _My StatefulWidget createState() => _My StatefulWidget();  
}  
  
class _My StatefulWidget extends State<My StatefulWidget> {  
    @override  
    Widget build(BuildContext context) {  
        return Container();  
    }  
} (Bhatt, 2025)
```

Metode u životnom ciklusu  *StatefulWidgeta*:

- *createState()* - poziva se pri prvom ubacivanju *widgeta* u *widget tree*. Svrha mu je stvaranje *State* objekta koji će biti asociran s tim *widgetom*.

```
class CounterWidget extends StatefulWidget {  
    @override  
    _CounterWidgetState createState() {  
        print('createState() called');  
        return _CounterWidgetState();  
    }  
} (Bhatt, 2025)
```

- `initState()` - poziva se čim je `State` objekt stvoren, prije prvog `builda`. Svrha mu je inicijalizacija podataka koji ovise o specifičnom kontekstu ili `widget` konfiguraciji.

```
class _CounterWidgetState extends State<CounterWidget> {
  late AnimationController _controller;
  int _counter = 0;

  @override
  void initState() {
    super.initState();
    print('initState() called');

    // Initialize controllers
    _controller = AnimationController(
      duration: Duration(seconds: 1),
      vsync: this,
    );

    // Set up listeners
    _setupListeners();

    // Load initial data
    _loadInitialData();
  }

  void _setupListeners() {
    // Set up any listeners here
  }

  void _loadInitialData() {
    // Load data from API, database, etc.
  }
} (Bhatt, 2025)
```

- `didChangeDependencies()` - poziva se odmah nakon `initState()` metode i svaki put kada se ovisnosti (eng. *dependencies*) tog `widgeta` promijene. Svrha mu je obraditi promjene na kojima se `widget` oslanja.

```
@override
void didChangeDependencies() {
  super.didChangeDependencies();
  print('didChangeDependencies() called');

  // Safe to access inherited widgets here
  final theme = Theme.of(context);
  final mediaQuery = MediaQuery.of(context);
  // Update configuration based on dependencies
  _updateTheme(theme);
}

void _updateTheme(ThemeData theme) {
  // Update widget appearance based on theme
} (Bhatt, 2025)
```

- `build()` - poziv a se nakon `didChangeDependencies()`, svaki put kada se pozove `setState()` i kada roditelj *rebuilda* *widget* s novom konfiguracijom. Svrha mu je prikazati korisničko sučelje *widgeta*.

```

@Override
Widget build(BuildContext context) {
  print('build() called');

  return Scaffold(
    appBar: AppBar(
      title: Text('Counter: ${_counter}'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text('Count: ${_counter}'),
          ElevatedButton(
            onPressed: _incrementCounter,
            child: Text('Increment'),
          ),
        ],
      ),
    ),
  );
}

void _incrementCounter() {
  setState(() {
    _counter++;
  });
} (Bhatt, 2025)

```

- `didUpdateWidget(oldWidget)` - poziva se kada se roditeljski *widget* promijeni i treba ažurirati trenutačni *widget*. Svrha mu je usporediti staru i novu *widget* konfiguraciju i ažurirati *State* u skladu s tim.

```

@Override
void didUpdateWidget(CounterWidget oldWidget) {
  super.didUpdateWidget(oldWidget);
  print('didUpdateWidget() called');

  // Compare old and new widget properties
  if (widget.initialValue != oldWidget.initialValue) {
    setState(() {
      _counter = widget.initialValue;
    });
  }

  if (widget.animationDuration != oldWidget.animationDuration) {
    _controller.duration = widget.animationDuration;
  }
} (Bhatt, 2025)

```

- `setState()` - poziva se kada je potrebno ažurirati *widget*ov *State* i pokrenuti *rebuild* istog. Svrha mu je obavijestiti okvir (eng. *framework*) da se interno stanje promijenilo.

```
void _updateCounter() {
    setState(() {
        // Update state variables here
        _counter++;

        // Any code here will run before build()
        _validateCounter();
    });
}

void _validateCounter() {
    if (_counter > 100) {
        _showWarning();
    }
} (Bhatt, 2025)
```

- `deactivate()` - pozove se kada se *widget* privremeno ukloni iz *widget* stabla. Svrha mu je počistiti prije potencijalnog odlaganja (eng. *disposal*), ali moguće je da se *widget* ponovo ubaci.

```
@override
void deactivate() {
    print('deactivate() called');
    super.deactivate();

    // Pause timers, animations
    _controller.stop();

    // Don't dispose resources here - widget might be reinserted
} (Bhatt, 2025)
```

- `dispose()` - poziva se kada je *widget* trajno izbačen iz *widget* stabla. Svrha mu je oslobađanje resursa kako bi se spriječilo curenje memorije (eng. *memory leak*)

```
@override
void dispose() {
    print('dispose() called');
    // Dispose controllers
    _controller.dispose();
    // Cancel subscriptions
    _subscription?.cancel();
    // Close streams
    _streamController.close();
    // Remove listeners
    _removeListeners();
```

```
super.dispose();  
} (Bhatt, 2025)
```

## 2.1.2 Usporedba s React Native

Flutter i React Native često se uspoređuju kada je u pitanju višeplatformski razvoj mobilnih aplikacija, jer oba omogućavaju da se jedan kod koristi i za iOS i za Android, čime se štede i vrijeme i resursi (Łojniewski, 2025).

Sličnosti:

- Oba omogućuju izradu aplikacija za više platformi (iOS, Android) iz jednog koda, što smanjuje trošak i vrijeme razvoja (Łojniewski, 2025).
- Oba okvira koriste pristupe koji omogućavaju ponovno korištenje komponenti (*widgeti* u Flutteru, komponente u React Nativeu), što doprinosi strukturiranom i održivom kodu (Łojniewski, 2025).
- Oba imaju jaku zajednicu developera, česte nadogradnje i poboljšanja, što osigurava da oba alata ostaju relevantna i prilagođena novim zahtjevima tržišta (Łojniewski, 2025).

Razlike:

- Flutter kompajlira aplikacije direktno u nativni strojni kod, što daje bolje performanse, posebno kod grafički zahtjevnih aplikacija, dok React Native je tek nedavno dodao *New Architecture* koji zamjenjuje asinkroni most između JavaScripta i izvornih dretvi (eng. *native threads*), a rezultat toga je značajno poboljšana responzivnost aplikacije (Łojniewski, 2025).
- Flutter koristi Dart, manje rasprostranjeni jezik, dok React Native koristi JavaScript, koji je široko poznat i popularan (Łojniewski, 2025).
- Flutter omogućuje visoku razinu kontrole nad izgledom zahvaljujući *widgetima*, što omogućuje dizajn koji je vrlo prilagođen, konzistentan i često neovisniji o nativnim UI komponentama platformi. React Native koristi nativne komponente i ima poboljšanja u *layoutu* i stiliranju, ali ponekad može biti ograničen u dizajnu koji želi odstupati od zadane platforme (Łojniewski, 2025).

## 2.2 Kotlin

Kotlin je moderan, statički tipiziran (eng. *statically typed*) programski jezik kojeg je razvio JetBrains, a službeno je podržan od Googlea za razvoj Android aplikacija. Namijenjen je izradi pouzdanih, sigurnih i čitljivih aplikacija uz smanjivanje količine potrebnog koda u odnosu na Javu. Jedna od glavnih značajki Kotlina je

interoperabilnost s Javom, što znači da se Kotlin kod može kombinirati s postojećim Java projektima bez prepreka.

Kotlin pruža napredne funkcionalnosti poput sigurnosti od *null* vrijednosti (eng. *null safety*), funkcija višeg reda, proširenih funkcija i korutina za jednostavno upravljanje konkurentnošću. Sve te značajke omogućuju pisanje kraćeg, čitljivijeg koda koji je ujedno i manje podložan greškama.

Osim za Android razvoj, Kotlin se koristi i u razvoju web aplikacija, *server-side* sustava i multiplatformskih projekata [7].

### 2.2.1 Usporedba s Javom

Kotlin je dizajniran tako da u potpunosti interoperira s Javom, što znači da se Kotlin kod može koristiti unutar postojećih Java projekata i obrnuto. Iako oba jezika dijele JVM (*Java Virtual Machine*) kao zajedničku platformu, postoje značajne razlike u sintaksi i funkcionalnostima (Fadatare, 2025).

Sličnosti:

- Oba jezika se izvršavaju na JVM-u, što omogućuje platformsku neovisnost i slične performanse [8].
- Oboje se mogu koristiti za razvoj *backenda* aplikacija, web API-ja i mobilne aplikacije [8].
- Oba jezika podržavaju objektno-orientirano programiranje i imaju pristup velikim bibliotekama i okvirima za razvoj [8].

Razlike:

- Kotlin je kraći, izražajniji i ima manje *boilerplate* koda, dok je Java opširnija [8].
- Kotlin ima ugrađen *null safety* što smanjuje greške tokom izvođenja, dok u Javi toga nema [8].
- Kotlin koristi korutine za jednostavno i učinkovito upravljanje konkurentnošću dok Java koristi dretve (eng. *threads*) koje mogu biti kompleksnije [8].

## 2.3 Spring Boot

Spring Boot je Java okvir otvorenog koda koji pojednostavljuje razvoj samostalnih produkcijski spremnih aplikacija temeljenih na Springu. Dizajniran je kako bi eliminirao potrebu za složenim konfiguracijama i omogućio brzi razvoj aplikacija. Neke od njegovih ključnih značajki su:

- Mogućnost ugradnje Tomcat, Jetty ili Undertow (nije potrebno implementirati WAR datoteke) [9].
- Automatsko konfiguriranje Spring i drugih biblioteka [9].
- Pruža značajke kao što su metrike, provjere ispravnosti (eng. *health checks*) i eksternalizirane konfiguracije [9].

Sve to omogućava brži razvoj zahvaljujući smanjenju *boilerplate* koda, jednostavnu integraciju s drugim Spring projektima, kao što su Spring Data, Spring Security i Spring Cloud, a za početak rada sa Spring Bootom, preporučuje se korištenje Spring Initializr, alat koji brzo i jednostavno, u par klikova, generira početnu strukturu projekta [9].

## 2.4 H2

H2 je lagana i brza, relacijska baza podataka otvorenog koda napisana u Javi. Može se izvoditi u dva načina: u memoriji (eng. *in-memory*) i ugrađenom načinu (eng. *embedded*). *In-memory* način je koristan za testiranje i razvoj jer omogućuje stvaranje privremene baze podataka koja se automatski uništava kada se aplikacija zaustavi. Ugrađeni način rada koristi se za aplikacije kojima je potrebna mala, samostalna baza podataka [10].

Neke od značajki H2 baze podataka su:

- Vrlo brz, otvorenog koda, JDBC API
- Ugrađeni i poslužiteljski načini rada; baze podataka na disku ili u memoriji.
- Podrška za transakcije, konkurentnost više verzija
- Konzolna aplikacija u pregledniku
- Šifrirane baze podataka
- Pretraživanje
- Čista Java s malim otiskom: veličina JAR datoteke oko 2,5 MB
- ODBC upravljački program

Primjer konfiguracije u `application.properties` datoteci, ako je `spring.h2.console.enabled` postavljen na `true`, konzola je dostupna na `http://localhost:<port>/h2-console`, gdje *port* označava *port* na kojem je pokrenut *backend*, i može joj se pristupiti s upisanim korisničkim imenom i lozinkom [10].

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:dcapp
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
```

```
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect [10]
```

## 2.4.1 Usporedba s MySQL

MySQL je vrlo skalabilna baza podataka, pogodna za velike količine podataka i kompleksne upite. Neke od ključnih značajki MySQLa su:

- Visoka skalabilnost - MySQL pruža izvrsne performanse na ogromnim količinama podataka i složenim upitima. Na primjer, njegova arhitektura omogućuje horizontalno skaliranje putem *shardinga* i replikacije, čime se osiguravaju izvrsne performanse za aplikacije s velikim prometom [11].
- Robustan sigurnosni okvir - ima sveobuhvatan sigurnosni okvir koji pokriva autentifikaciju korisnika s raznim algoritmima za *hashiranje* lozinki, kontrole pristupa putem uloga i privilegija, kao i mogućnost šifriranja podataka [11].
- Opsežan skup značajki - mogućnost korištenja pohranjenih procedura (eng. *stored procedures*), okidača (eng. *triggers*), pogleda (eng. *views*) i funkcija su par naprednih značajki koje MySQL nudi [11].

Na slici 1 su prikazane razlike između MySQL-a i H2 baze podataka po pitanjima skalabilnosti, performansi, sigurnosti, lakoće korištenja, dostupnih funkcionalnosti, idealnog zahtjeva korištenja i *deploymenta*.

Feature	MySQL	H2
Deployment	Server-based (separate process)	Flexible (in-memory, embedded, server mode)
Scalability	Highly scalable (horizontal scaling)	Limited scalability for large datasets
Performance	Excellent for complex queries and high traffic	Excellent in-memory performance
Security	Robust security features (authentication, access control, encryption)	Basic security features (authentication, access control)
Ease of Use	User-friendly with extensive documentation and community support	Simple setup, ideal for development and testing
Feature Set	Comprehensive (stored procedures, triggers, views, functions)	Streamlined (focuses on core functionalities)
Ideal Use Cases	Large applications, high traffic, complex data	Development, testing, embedded applications, small datasets

Slika 1 Razlike MySQL i H2 [11]

## 2.4.2 Rad s bazama podataka u Javi: JPA i JDBC

JDBC (*Java Database Connectivity*) je programsko sučelje (*API*) koji pomaže Java aplikacijama da komuniciraju s bazama podataka, omogućuje im povezivanje s bazom, pokretanju upita, dohvaćanje i manipuliranje podacima. Zahvaljujući JDBC-u, Java aplikacije mogu jednostavno raditi s različitim relacijskim bazama podataka poput MySQL, Oraclea, PostgreSQL-a i drugih [12].

JPA (*Java Persistence API*) je Java specifikacija koja olakšava rad s relacijskim bazama podataka. Omogućuje mapiranje Java klasa (označenih s `@Entity` anotacijom) na tablice baze podataka i upravljanje podacima pomoću jednostavnih API-ja umjesto pisanja složenog SQL-a [13].

Glavne razlike:

- JDBC omogućuje pisanje SQL naredbi za čitanje podataka iz baze podataka, a JPA, omogućuje programerima izradu *database-driven* Java programa koristeći objektno orijentiranu semantiku. JPA anotacije opisuju kako se određena Java klasa i njezine varijable mapiraju na određenu tablicu i njezine stupce [14].

```
@Entity  
@Table(name = "employee")  
public class Employee implements Serializable {  
    @Column(name = "employee_name")  
    private String employeeName;  
}
```

- JDBC ovisi o bazi podataka, što znači da se za različite baze podataka moraju pisati različite skripte dok je JPA agnostičan u odnosu na bazu podataka, što znači da se isti kod može koristiti u raznim bazama s malo (ili bez) izmjena [14].
- U JDBC-u se upravljanja transakcijama obavlja eksplisitno korištenjem `commit` i `rollback`. S druge strane, upravljanje transakcijama je implicitno osigurano u JPA [14].

### 3 Implementacija projekta

Projekt je podijeljen u dvije cjeline, korisničko sučelje i *backend*. Oba projekta imaju zasebni repozitorij na GitHubu. Korisničko sučelje je dostupno na: <https://github.com/DavidMaglica/seat-saver-frontend>, a *backend* je dostupan na: <https://github.com/DavidMaglica/seat-saver-backend>.

#### 3.1 Implementacija korisnikog sučelja

Korisničko sučelje je pisano u Flutteru jer omogućuje razvoj jedinstvene kodne baze za više platformi, što značajno skraćuje vrijeme razvoja i olakšava održavanje projekta.

Za potrebe ovog projekta izrađena su dva odvojena korisnička sučelja: mobilna aplikacija, namijenjena krajnjim korisnicima, i web aplikacija u obliku administrativnog sučelja, namijenjena voditelju ili osoblju ugostiteljskog objekta, za upravljanje rezervacijama i objektima. Oba sučelja dijele većim djelom zajedničku logiku i stilove zahvaljujući modularnoj arhitekturi Fluttera, ali su funkcionalno razdvojena kako bi zadovoljila specifične potrebe svojih korisnika.

Osim osnovnog Flutter SDK-a, korištene su i različite biblioteke za proširenje funkcionalnosti i olakšavanje razvoja. *Provider* biblioteka je korištena za upravljanje stanjem aplikacije, *dio* za komunikaciju s *backendom* poput REST API-ja i *intl* za rad s vremenskim i lokalizacijskim podacima.

##### 3.1.1 Struktura korisnikog sučelja

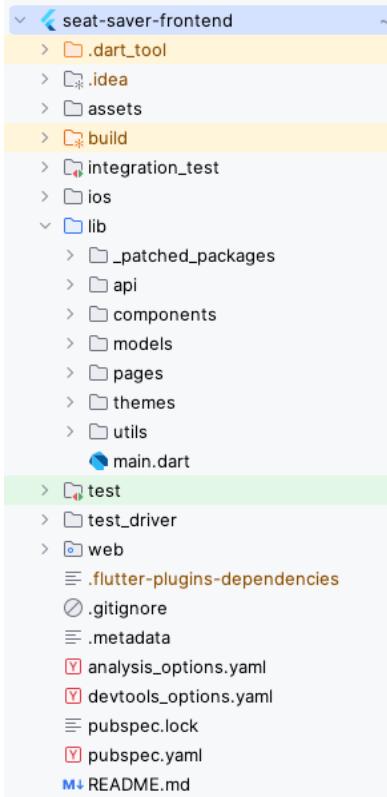
Projekt je strukturiran s posebnom pažnjom na modularnost samog projekta. Takva struktura omogućuje jasnu podjelu odgovornosti, jednostavnije održavanje i lakšu nadogradnju funkcionalnosti u budućnosti.

Unutar glavnog direktorija nalazi se mapa `lib`, koja čini središnji dio aplikacije i sadrži sav izvorni kod pisan u Dartu. Direktorij je dodatno podijeljen u 8 cjelina:

- `_patched_packages` - sadrži lokalne biblioteke koje su stvarale probleme s ostalim ovisnostima (eng. *dependencies*).
- `api` - sadrži implementaciju komunikacije s *backendom* i definiranje servisnih klasa za slanje HTTP zahtjeva, API rute i modele .
- `components` - sadrži *widgete* koji se koriste na više mesta unutar aplikacije.

- `models` - predstavlja *view model* sloj aplikacije. Njegova je uloga komunikacija između *model* (API) sloja i *view* sloja (sloja stranica).
- `pages` - sadrži logički odvojene stranice aplikacije koje predstavljaju korisničko sučelje.
- `themes` - sadrži konfiguracije izgleda aplikacije, uključujući boje, tipografiju i stilove.
- `utils` - sadrži razne pomoćne klase i funkcije koje olakšavaju implementaciju ponavljajućih zadataka.
- `main.dart` - ulazna točka aplikacije unutar koje se inicijalizira cijeli sustav i definira osnovna navigacija.

Osim direktorija `lib`, važan dio strukture čine i direktoriji `assets` (pohrana slika, fontova i drugih statičkih resursa), `test` (*unit* i *widget* testovi) i `integration_test` (E2E, tj. integracijski testovi). Konfiguracija ovisnosti (eng. *dependencies*) i dodatnih resursa nalazi se u datoteci `pubspec.yaml`. Struktura projekta je prikazana na slici 2.



Slika 2 Struktura korisničkog sučelja

### **3.1.2 Implementacija mobilne aplikacije**

Mobilna aplikacija je zamišljena za isključivo korištenje od strane krajnjih korisnika, odnosno ljudima koji žele rezervirati mjesto u nekom ugostiteljskom objektu. Za samu implementaciju mobilne aplikacije korišteni su razni paketi, kako za vizualnu implementaciju nekih funkcionalnosti, tako i za implementaciju logičkih funkcionalnosti. Neki od tih paketa su: paket *google\_maps\_flutter* koji se koristi za prikaz Google mapa, paket *google\_sign\_in* za implementaciju autentifikacije pomoću Google računa. Osim navedenih, korišten je i veći broj drugih paketa koji su doprinijeli cijelokupnoj funkcionalnosti aplikacije.

Sukladno namjeni, mobilna aplikacija se sastoji od osam glavnih stranica koje zajedno čine cjelinu korisničkog iskustva. Početna stranica služi kao ulaz u aplikaciju i omogućuje njeno korištenje i bez prijavljenog računa, ali tada su funkcionalnosti ograničene, jer korisnik ne može napraviti rezervaciju. Za punu funkcionalnost aplikacije, predviđena je stranica za autentifikaciju koja se sastoji od *tabova* za registraciju za nove korisnike i prijavu za postojeće korisnike. Nakon prijave korisniku postaje dostupna stranica računa koja nudi mogućnosti uređivanja osobnih podataka (email adresa, korisničko ime i lozinka), kao i pregled povijesti rezervacija, pristup korisničkoj podršci, pregled uvjeta korištenja i postavke obavijesti i dopuštenja aplikaciji.

Na glavnoj stranici (eng. *homepage*) aplikacije prikazani su ugostiteljski objekti kroz četiri različite kategorije: novi, obližnji, popularni i preporučeni (*new, nearby, popular, suggested*), prema tim kategorijama postoje i zasebne stranice gdje se mogu vidjeti svi objekti filtrirani po kategoriji. Pretraga ugostiteljskih objekata je omogućena kroz zasebnu stranicu koji omogućuje filtriranje objekata prema imenu, lokaciji ili tipu, čime se korisniku olakšava pronađak željenog mjesta. Uz to, aplikacija nudi i stranicu s kartom po kojoj se korisnik može kretati i vizualno pregledati gdje se pojedini objekt nalazi. Kada korisnik odabere željeni objekt, otvara se stranica s detaljima tog objekta, kao što su naziv objekta, lokacija, radno vrijeme, prosječna ocjena i fotografije objekta i menija objekta, moguće je i pristupiti svim recenzijama objekta na drugoj stranici, gdje korisnik može i sam pustiti recenziju. Na istoj stranici moguće je i stvoriti rezervaciju.

Tako aplikacija pokriva cijeli proces, od prvog ulaska i pregleda ponude, preko pretrage i odabira pa sve do rezervacije i ostavljanja recenzije nakon korištenja usluge.

### **3.1.3 Implementacija web aplikacije (administrativnog sučelja)**

Web aplikacija je zamišljena za isključivo korištenje vlasnika ugostiteljskog objekta ili njegovih zaposlenika. Kao i za implementaciju mobilne, korišteni su razni paketi za olakšavanje implementacije, poput *fl\_chart* paketa za prikaz grafova vezanih za zauzetost objekta, *toastification* paketa za prikazivanje *toaster* notifikacija.

Web aplikacija se sastoji od devet glavnih stranica koje čine cjelinu administrativnog iskustva. Početna služi kao ulaz u aplikaciju, ali za razliku od mobilne aplikacije, web aplikacija se ne može koristiti ako korisnik nema korisnički račun. S početne stranice korisnik se preusmjerava na stranicu za autentifikaciju, koja se, kao i u mobilnoj verziji, sastoji od *tabova* za registraciju novih i prijavu postojećih korisnika.

Nakon prijave, korisniku postaje dostupna glavna stranica (eng. *homepage*) koja prikazuje razne informacije o poslovanju objekata pod njegovim vlasništvom. Korisnik može vidjeti statistike o rezervacijama u posljednjih i sljedećih 30 dana, ukupni broj zaprimljenih rezervacija i recenzija, prosječnu ocjenu svih objekata i indikator zauzetosti. Osim toga, prikazana je i tablica sa svim objektima, kao i podaci o najbolje i najslabije ocijenjenim objektima. Za detaljni uvid u recenzije, web aplikacija nudi zasebnu stranicu gdje se u obliku kartica prikazuju svi objekti s pripadajućim brojem recenzija i prosječnom ocjenom. Rezervacije se prate putem posebne stranice na kojoj se nalazi tablica svih rezervacija. Kako bi korisnik imao pregled dinamike rezervacija, stranica rezervacijskih grafova prikazuje vizualizacije broja rezervacija po danu ili tjednu. Uz to, za svaki objekt postoji zasebna stranica koja se sastoji od tri *taba*, prvi sa svim podacima objekta, drugi s prikazom svih recenzija i treći sa svim slikama objekta i menija. Konačno, aplikacija uključuje i stranicu s mozaičkim pregledom svih objekata pod vlasništvom korisnika, omogućujući brzu orientaciju i jednostavan odabir objekta za daljnje upravljanje.

Time web aplikacija vlasnicima objekata i njihovim zaposlenicima pruža sve potrebne alate za pregled, analizu i upravljanje objektima, rezervacijama i recenzijama, objedinjeno unutar jednog sustava.

### **3.1.4 Komunikacija s backendom**

Za komunikaciju s *backendom* koristi se *dio* paket, koji omogućava slanje HTTP zahtjeva i obradu odgovora, i nudi napredne mogućnosti izvan osnovnog slanja GET ili POST zahtjeva, i veoma je lagan za postaviti.

```

Dio setupDio() {
    return Dio(
        BaseOptions(
            baseUrl: 'http://<backend-address>:<backend-port>',
            connectTimeout: const Duration(seconds: 5),
            receiveTimeout: const Duration(seconds: 3),
        ),
    );
}

```

Komunikacija prema *backendu* je organizirana kroz pet zasebnih klasa: AccountApi, GeolocationApi, ReservationsApi, SupportApi i VenuesApi. Svaka od ovih klasa je zadužena za komunikaciju s pripadajućim REST kontrolerom na *backendu*. Dodatno, koristi se i *logger* paket, koji omogućuje detaljno logiranje grešaka i ostalih važnih događaja tokom komunikacije s *backendom* kako bi se olakšalo praćenje toka izvršavanja i dijagnosticiranje problema u sustavu.

Za razmjenu multimedijskih sadržaja, poput slanja i primanja slika, koristi se jedna značajka *dio* paketa, koja omogućuje rad s *multipart/form-data* datotekama te njihovo uključivanje kao parametar unutar API poziva.

Odgovori koje vraća *backend* razlikuju se ovisno o pozvanoj metodi, pa korisničko sučelje mora moći obraditi različite formate podataka. Kako bi se to postiglo, definirane su dodatne klase koje predstavljaju modele povratnih vrijednosti, kako bi se standardizirala komunikacija između *backenda* i korisničkog sučelja.

Prva od tih klasa je klasa BasicResponse, koja sadrži bool success, koji prenosa informaciju o uspješnosti provođenja neke metode na *backendu*, zatim String message, koji se sastoji od nekakve poruke s *backenda* koja može ili označiti točno što je to pošlo po zlu ili kratku poruku uspjeha te za kraj neki opcionalni generički tip koji se vraća kao dio odgovora, ako metoda osim statusa i poruke vraća dodatni sadržaj.

```

class BasicResponse<T> {
    final bool success;
    final String message;
    final T? data;

    BasicResponse({required this.success, required this.message,
    this.data});

    factory BasicResponse.fromJson(
        Map<String, dynamic> json,
        T Function(Map<String, dynamic>) fromJsonT,
    ) {
        return BasicResponse(
            success: json['success'],
            message: json['message'],

```

```

        data: json['data'] != null ? fromJsonT(json['data']) : null,
    );
}
}

```

Primjer poziva *backenda* i vraćanje BasicResponse objekta:

```

Future<BasicResponse> createReservation({
    required int venueId,
    required int numberOfGuests,
    required DateTime reservationDate,
    int? userId,
    String? userEmail,
}) async {
    try {
        Response response = await dio.post(
            ApiRoutes.reservations,
            data: {
                'userId': userId,
                'userEmail': userEmail,
                'venueId': venueId,
                'reservationDate': reservationDate.toIso8601String(),
                'numberOfGuests': numberOfGuests,
            },
        );
        return BasicResponse.fromJson(response.data, (json) => json);
    } catch (e) {
        logger.e('Error creating reservation: $e');
        return BasicResponse(
            success: false,
            message: 'Failed to create reservation',
        );
    }
}

```

Sljedeća takva klasa je klasa PagedResponse, čija je glavna svrha standardizirano rukovanje paginiranim podacima koje *backend* vrača u slučaju da je rezultat prevelik da bi se isporučio odjednom. Ova klasa omogućuje lakše upravljanje dohvaćenim podacima, dijeljenjem rezultata u stranice i pružanjem meta podataka vezanih za strukturu tih rezultata. Klasa se sastoji od int page koji označuje redni broj trenutačno dohvaćene stranice, int size koji označuje broj elemenata na trenutačnoj stranici, int totalElements koji označuje ukupan broj elemenata u cijelom rezultatu, int totalPages koji označuje ukupan broj stranica u cijelom rezultatu i listu generičkih tipova content s podacima koji se nalaze na određenoj stranici. Uz to, klasa sadrži i pomoći *getter* items koji vraća listu elemenata sadržanih unutar atributa content, čime se olakšava daljnja obrada. Time, PagedResponse omogućuje korisničkom sučelju da učinkovito upravlja prikazom i obradom velikih

skupova podataka, bez potrebe da se svi podaci vraćaju odjednom, što značajno doprinosi optimizaciji performansi i boljem korisničkom iskustvu.

```
class PagedResponse<T> {
    final List<T> content;
    final int page;
    final int size;
    final int totalElements;
    final int totalPages;

    PagedResponse({
        required this.content,
        required this.page,
        required this.size,
        required this.totalElements,
        required this.totalPages,
    });

    factory PagedResponse.fromJson(Map<String, dynamic> json,
        T Function(dynamic) fromJsonT,) {
        return PagedResponse<T>(
            content: (json['content'] as List).map(fromJsonT).toList(),
            page: json['page'],
            size: json['size'],
            totalElements: json['totalElements'],
            totalPages: json['totalPages'],
        );
    }

    List<T> get items => content;
}
```

Primjer poziva *backenda* i vraćanje PagedResponse objekta:

```
Future<PagedResponse<Venue>> getAllVenues(
    int page,
    int size,
    String? searchQuery,
    List<int>? typeIds,
) async {
    try {
        final Map<String, dynamic> queryParams = {'page': page, 'size':
size};

        if (searchQuery != null && searchQuery.trim().isNotEmpty) {
            queryParams['searchQuery'] = searchQuery.trim();
        }

        if (typeIds != null && typeIds.isNotEmpty) {
            queryParams['typeIds'] = typeIds;
        }

        final Response response = await dio.get(
            ApiRoutes.venues,
            queryParameters: queryParams,
```

```

    ) ;

    return PagedResponse.fromJson(
        response.data,
        (json) => Venue.fromJson(json),
    );
} catch (e) {
    logger.e('Error fetching all venues: $e');
    return PagedResponse<Venue>(
        content: [],
        page: 0,
        size: 0,
        totalElements: 0,
        totalPages: 0,
    );
}
}
}

```

### 3.1.5 Testovi

Za osiguravanje pouzdanosti i kvalitete aplikacije provedena su različita testiranja na više razina. Na najnižoj razini korišteni su jedinični (eng. *unit*) testovi, kojima je cilj provjeriti ispravnost pojedinih metoda i klasa u izolaciji. Testirani su svi API pozivi, svi podatkovni objekti, kako bi se osiguralo pravilno mapiranje podataka dobivenih s *backenda* i testirane su sve pomoćne metode koje se koriste unutar aplikacije. U jediničnim testovima korišteni su *mockovi*, s pomoću paketa *mocktail*, čime se omogućilo simuliranje odgovora s *backenda* i kontrolirani uvjeti testiranja.

Na višoj razini implementirani su *widget* testovi, koji provjeravaju ispravan prikaz i funkcioniranje glavnih stranica aplikacije, kako u mobilnoj tako i u web verziji. Za potrebe ovih testova korišteni su *fake* modeli u kojima su metode namjerno prilagođene da vraćaju uspješne ili neuspješne odgovore, čime se provjerava ponašanje aplikacije u različitim scenarijima. Time testira se korisničko iskustvo bez potrebe za stvarnom komunikacijom s *backendom*.

Na najvišoj razini provedeni su *end-to-end* (E2E) ili integracijski testovi, kojima se simulira stvarno korištenje aplikacije. Ovi testovi pokrivaju glavne korisničke tijekove korištenja, poput autentifikacije, kreiranje ugostiteljskog objekta ili kreiranje rezervacije. Za razliku od jediničnih i *widget* testova, ovdje se sustav ne oslanja na *mockove*, nego se testira cjelokupna integracija između korisničkog sučelja i *backenda*, čime se potvrđuje da svi dijelovi sustava zajedno rade ispravno.

Za implementaciju testova korišteni su službeni Flutter i Dart paketi (*flutter\_test* i *integration\_test*), dok je za izradu *mockova* korišten paket *mocktail*. Tako je ostvareno

pokrivanje svih ključnih razina testiranja, od najmanjih logičkih jedinica, do integriranog sustava u stvarnom okruženju.

Primjer jediničnog testa koji potvrđuje očekivani rad `signUp()` metode kada je odgovor pozitivan:

```
test('should return BasicResponse with userId on success', () async {
  final mockResponse = Response(
    data: {
      'success': true,
      'message': 'User signed up',
      'data': testUserId,
    },
    requestOptions: RequestOptions(path: ApiRoutes.signUp),
  );

  when(
    () => mockDio.post(
      ApiRoutes.signUp,
      queryParameters: any(named: 'queryParameters'),
    ),
  ).thenAnswer((_) async => mockResponse);

  final result = await accountApi.signUp(
    testUserEmail,
    testUsername,
    testPassword,
    false,
  );

  expect(result.success, true);
  expect(result.data, testUserId);
  expect(result.message, 'User signed up');
});
```

Primjer `widget` testa u kojem se testira ispravan prikaz Homepage stranice na mobilnoj aplikaciji:

```
testWidgets('should display widget correctly with user', (tester) async {
  setupSharedPreferencesMock(initialValues: {'userId': userId});
  final welcomeText = find.byKey(const Key('welcomeText'));
  final welcomeUserText = find.text('Welcome back, username!');
  final carouselSlider = find.byKey(const Key('carouselSlider'));
  final carouselItem = find.byKey(const Key('carouselItem'));
  final nearbyVenuesTitle = find.text('Nearby Venues');
  final newVenuesTitle = find.text('New Venues');
  final trendingVenuesTitle = find.text('Trending Venues');
  final suggestedVenuesTitle = find.text('We suggest');
  final seeAllButton = find.byKey(const Key('seeAllButton'));
  final venueCardsScrollView = find.byKey(const
  Key('venueCardsScrollView'));
  final venueSuggestedCardsScrollView = find.byKey(
```

```

        const Key('venueSuggestedCardsScrollView'),
    );
    final venueCard = find.byKey(const Key('venueCard'));
    final venueSuggestedCard = find.byKey(const
Key('venueSuggestedCard'));

await tester.pumpWidget(
    MaterialApp(
        home: Scaffold(
            body: Homepage(userId: userId, modelOverride: model),
        ),
    ),
);
await tester.pumpAndSettle();

expect(welcomeText, findsOneWidget);
expect(welcomeUserText, findsOneWidget);
expect(carouselSlider, findsOneWidget);
expect(carouselItem, findsNWidgets(3));
expect(nearbyVenuesTitle, findsOneWidget);
expect(newVenuesTitle, findsOneWidget);
expect(trendingVenuesTitle, findsOneWidget);
expect(suggestedVenuesTitle, findsOneWidget);
expect(seeAllButton, findsNWidgets(4));
expect(venueCardsScrollView, findsNWidgets(3));
expect(venueSuggestedCardsScrollView, findsOneWidget);
expect(venueCard, findsNWidgets(9));
expect(venueSuggestedCard, findsNWidgets(3));
});

```

Primjer integracijskog (E2E) testa u kojem se testira stvaranje ugostiteljskog objekta preko web aplikacije:

```

testWidgets('should be able to create new venue', (tester) async {
    final getStartedButton = find.byKey(const Key('getStartedButton'));
    final logInEmailField = find.byKey(const Key('logInEmailField'));
    final logInPasswordField = find.byKey(const
Key('logInPasswordField'));
    final logInButton = find.byKey(const Key('logInButton'));
    final logInTab = find.byKey(const Key('logInTab'));
    final createVenueButton = find.byKey(const Key('createVenueButton'));
    final venueNameField = find.byKey(const Key('venueNameField'));
    final venueLocationField = find.byKey(const
Key('venueLocationField'));
    final maxCapacityField = find.byKey(const Key('maxCapacityField'));
    final venueDescriptionField = find.byKey(
        const Key('venueDescriptionField'),
    );
    final venueTypeDropdown = find.byKey(const Key('venueTypeDropdown'));
    final workingHoursField = find.byKey(const Key('workingHoursField'));
    final dayChipMonday = find.byKey(const Key('dayChip_Monday'));
    final dayChipTuesday = find.byKey(const Key('dayChip_Tuesday'));
    final dayChipWednesday = find.byKey(const Key('dayChip_Wednesday'));
    final dayChipThursday = find.byKey(const Key('dayChip_Thursday'));
    final dayChipFriday = find.byKey(const Key('dayChip_Friday'));
}
);

```

```

final dayChipSaturday = find.byKey(const Key('dayChip_Saturday'));
final dayChipSunday = find.byKey(const Key('dayChip_Sunday'));
final submitButton = find.byKey(const Key('submitButton'));

app.main();

await tester.pumpAndSettle();

expect(getStartedButton, findsOneWidget);

await tester.tap(getStartedButton);
await tester.pumpAndSettle();

expect(logInTab, findsOneWidget);

await tester.tap(logInTab);
await tester.pumpAndSettle();

expect(logInEmailField, findsOneWidget);
expect(logInPasswordField, findsOneWidget);
expect(logInButton, findsOneWidget);

await tester.enterText(logInEmailField, email);
await tester.enterText(logInPasswordField, password);
await tester.pumpAndSettle();
await tester.tap(logInButton);
await tester.pumpAndSettle();

expect(createVenueButton, findsOneWidget);

await tester.tap(createVenueButton);
await tester.pump();
await tester.pump(const Duration(seconds: 1));

expect(find.byType(CreateVenueModal), findsOneWidget);

expect(venueNameField, findsOneWidget);
expect(venueLocationField, findsOneWidget);
expect(maxCapacityField, findsOneWidget);
expect(venueDescriptionField, findsOneWidget);
expect(venueTypeDropdown, findsOneWidget);
expect(workingHoursField, findsOneWidget);
expect(dayChipMonday, findsOneWidget);
expect(dayChipTuesday, findsOneWidget);
expect(dayChipWednesday, findsOneWidget);
expect(dayChipThursday, findsOneWidget);
expect(dayChipFriday, findsOneWidget);
expect(dayChipSaturday, findsOneWidget);
expect(dayChipSunday, findsOneWidget);
expect(submitButton, findsOneWidget);

await tester.enterText(venueNameField, 'Test Venue');
await tester.enterText(venueLocationField, 'Poreč, Croatia');
await tester.enterText(maxCapacityField, '50');
await tester.enterText(workingHoursField, '00:01 - 23:59');
await tester.tap(dayChipMonday);
await tester.tap(dayChipTuesday);
await tester.tap(dayChipWednesday);

```

```

    await tester.tap(dayChipThursday);
    await tester.tap(dayChipFriday);
    await tester.tap(dayChipSaturday);
    await tester.tap(dayChipSunday);
    await tester.enterText(venueDescriptionField, 'Test venue.');
    await tester.tap(venueTypeDropdown);
    await tester.pump();
    await tester.pump(Duration(seconds: 1));
    final venueTypeOption = find.text('Japanese').last;
    expect(venueTypeOption, findsOneWidget);
    await tester.tap(venueTypeOption);
    await tester.pump();
    await tester.pump(Duration(milliseconds: 500));

    await tester.tap(submitButton);
    await tester.pumpAndSettle();

    expect(find.byType(WebVenuePage), findsOneWidget);
} );

```

## 3.2 Implementacija backenda

*Backend* sustav je izrađen koristeći Spring Boot, koji omogućuje jednostavnu izradu i konfiguraciju REST servisa. Spring Boot pokreće ugrađeni web server (Tomcat) te kroz modul Spring Web (Spring MVC) omogućuje obradu HTTP zahtjeva i vraćanje odgovora u JSON formatu. Ulazna točka aplikacije je klasa *Application.java* dok je ostatak logike implementiran u Kotlinu.

Komunikacija između korisničkog sučelja i *backenda* odvija se putem REST API-ja, pri čemu svaki zahtjev dolazi na kontroler sloj *backenda* koji zatim propagira zahtjev kroz ostatak servisa. Kako bi se korisničkom sučelju omogućila sigurna komunikacija s *backendom*, korišten je CORS filter. Browzeri blokiraju zahtjeve koji dolaze s drugih domena, s toga CORS konfiguracija definira koji izvori (eng. *origins*), metode i zaglavlja (eng. *headers*) su dopušteni. U ovome projektu dopušteni su svi izvori, kako bi se olakšao razvoj i testiranje, dok bi se u nekom produkcijskom okruženju moglo ograničiti samo na pouzdane domene.

Cjelokupna funkcionalnost API-ja je grupirana kroz verzionirane putanje (eng. *paths*) (svaka putanja počinje sa /api/v1/path), koje su dodatno organizirane prema pripadajućem kontroleru.

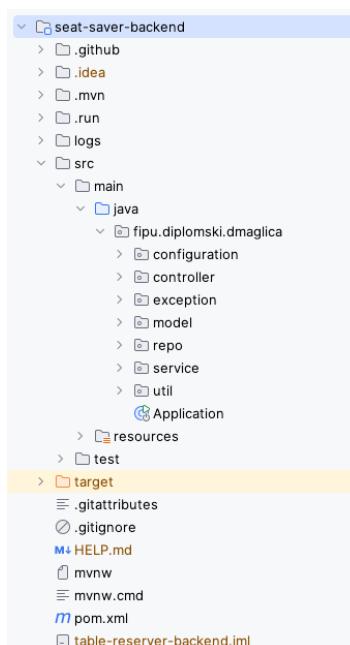
### 3.2.1 Struktura backenda

*Backend* projekt organiziran je prema standardnoj arhitekturi Spring Boot aplikacija, što omogućuje jasnu podjelu odgovornosti i jednostavnije održavanje koda. Ulazna

točka aplikacije se nalazi u klasi `Application` i ona pokreće cijeli sustav. Unutar paketa definirano je nekoliko mapa:

- `configuration` - sadrži konfiguracijske klase, poput postavki za CORS i drugih globalnih postavki aplikacije.
- `controller` - definira i izlaže REST API prema vanjskim klijentima (mobilnoj i web aplikaciji) i obrađuje ulazne HTTP zahtjeve.
- `exception` - definira rukovanje greškama i iznimkama koje se mogu pojaviti tijekom izvođenja aplikacije kroz `GlobalExceptionHandler`.
- `model` - sadrži podatkovne objekte (eng. *data classes*) koje se koriste u aplikaciji.
- `repo` - sadrži sve entitete koji predstavljaju tablice u bazi podataka i repozitorije koji nasljeđuju `JpaRepository` i služe za komunikaciju s bazom podataka.
- `service` - obuhvaća poslovnu logiku aplikacije, odnosno implementaciju funkcionalnosti koje se pozivaju iz kontrolera.
- `util` - sadrži pomoćne metode koje se koriste na više mesta u aplikaciji.

Osim navedenih direktorija, projekt sadrži i direktorij `resources` u kojem se nalaze konfiguracijske datoteke (`application.properties`) i SQL skripte za kreiranje i inicijalno punjenje baze podataka. Testovi su organizirani u zasebnom direktoriju `test`. Struktura projekta je prikazana na slici 3.



Slika 3 Struktura backend djela projekta

### 3.2.2 Upravljački sloj

Upravljački sloj čini ulaznu točku REST API-ja i odgovoran je za prihvatanje HTTP zahtjeva sa korisničkog sučelja i proslijedivanje istih prema servisnom sloju. Svaka klasa se mapira na određeni dio funkcionalnosti sustava i definira pripadajuće API rute.

U sklopu ovog projekta implementirano je pet glavnih klasa:

- `UserController` - prima HTTP pozive vezane za dohvaćanje i upravljanje korisničkim računima i poziva odgovarajući servis.
- `VenueController` - prima HTTP pozive vezane za dohvaćanje, i upravljanje ugostiteljskim objektima, slikama objekta i recenzijama vezanim uz objekte i poziva odgovarajući servis.
- `ReservationController` - prima HTTP pozive vezane za sve zahtjeve vezane za rezervacije i iste proslijedi odgovarajućem servisu.
- `GeolocationController` - prima HTTP pozive vezane za dohvat obližnjih gradova i poziva odgovarajući servis.
- `SupportController` - prima HTTP pozive vezane za korisničku podršku i poziva odgovarajući servis.

Time upravljački sloj osigurava jasnu podjelu odgovornosti, pridržavanje REST principa i jednostavnu komunikaciju između korisničkog sučelja i poslovne logike definirane u servisnom sloju. Bitno je naglasiti da klase u upravljačkom sloju same po sebi ne sadrže poslovnu logiku sustava, već djeluju kao posrednici koji primaju zahtjev, pozivaju odgovarajuću metodu u odgovarajućem servisu i vraćaju rezultat prema korisničkom sučelju. Upravo se u servisnom sloju nalazi jezgra aplikacije, gdje su implementirana validacija podataka i koordinacija pristupa bazi podataka.

Primjer klase `SupportController` koji poziva `SupportService`:

```
@RestController
class SupportController(
    private val supportService: SupportService
) {

    /**
     * Processes and forwards a user support request via email.
     *
     * Sends an email to the configured support address with:
     * - Subject: "Support Ticket from [userEmail] - [subject]"
     * - Body: The user's original message content
     *
     * @param userEmail The email address of the requester
    
```

```

    * @param subject Brief description of the support issue
    * @param body Detailed description of the support request
    * @return BasicResponse with:
    *   - success: true if email was queued successfully
    *   - message: Delivery status notification
    *
    */
    @PostMapping(Paths.SEND_EMAIL)
    fun sendEmail(
        @RequestParam("userEmail") userEmail: String,
        @RequestParam("subject") subject: String,
        @RequestParam("body") body: String
    ): BasicResponse = supportService.sendEmail(userEmail, subject,
body)
}

```

### 3.2.3 Servisni sloj

Servisni sloj čini središnji dio aplikacijske arhitekture jer implementira poslovnu logiku sustava. Dok kontroleri samo preusmjeravaju HTTP zahtjeve, servisi sadrže poslovnu logiku, validaciju podataka i koordinaciju između više izvora podataka. Time se postiže jasno odvajanje upravljačkog sloja od stvarne logike aplikacije.

U ovom projektu svaka klasa upravljačkog sloja ima pripadajući servis koji obrađuje zahtjeve i definira kako će se podaci dohvaćati, obrađivati i spremati u bazu podataka.

- **UserService** - upravlja korisničkim računima i povezanim operacijama. Omogućuje dohvaćanje, registraciju i autentifikaciju korisnika, uređivanje korisničkih podataka (email adresa, korisničko ime, lozinka), upravljanje lokacijskim podacima i postavkama obavijesti. Posebna pažnja posvećena je sigurnosti, lozinke se nikad ne vraćaju u odgovorima, a osjetljive operacije zahtijevaju autentifikaciju.
- **VenueService** - zadužen je za rad s ugostiteljskim objektima. Omogućuje dohvaćanje, stvaranje, uređivanje i brisanje objekata, rukovanje slikama objekta i menija, dohvati i upravljanje recenzijama te filtriranje objekata prema lokaciji, tipu ili pretraženom pojmu. Također implementira paginaciju i računa dostupnost objekata u stvarnom vremenu.
- **ReservationService** - obrađuje sve zahtjeve vezane uz rezervacije. Putem ovog kontrolera moguće je kreirati novu rezervaciju, ažurirati postojeću ili je obrisati. Također, nudi pregled rezervacija po korisniku, vlasniku ili objektu.

- GeolocationServicer - omogućuje dohvat obližnjih gradova na temelju geolokacijskih podataka.
- SupportService - služi za korisničku podršku, odnosno slanje upita ili prijava problema prema sustavu.

Ovakva organizacija olakšava testiranje i održavanje aplikacije, jer su sve ključne funkcionalnosti izolirane u servisnom sloju, dok kontroleri ostaju čisti i fokusirani na komunikaciju s korisničkim sučeljem.

Primjer servisa SupportService koji omogućava korisnicima slanje upita ili prijave problema:

```
@Service
class SupportService(
    private val mailSender: JavaMailSender,
    private val mailSenderConfiguration: MailSenderConfiguration
) {

    companion object {
        private val logger =
KotlinLogging.logger(SupportService::class.java.name)
    }

    fun sendEmail(userEmail: String, subject: String, body: String): BasicResponse {
        val message = SimpleMailMessage()
        message.setTo(mailSenderConfiguration.getUsername())
        message.subject = "Support Ticket from $userEmail - $subject"
        message.text = body

        return try {
            mailSender.send(message)
            BasicResponse(true, "Email sent successfully.")
        } catch (e: Exception) {
            logger.error { "There was an error while sending the email: ${e.message}" }
            BasicResponse(false, "There was an error while sending the email. Please try again later.")
        }
    }
}
```

### 3.2.4 Repozitoriji i komunikacija s bazom

Za pohranu i upravljanje podacima u *backend* sustavu korištena je prije spomenuta H2 baza podataka u memorijskom (eng. *in-memory*) načinu rada, što omogućuje brzo testiranje i razvoj bez potrebe za vanjskim serverskim sustavom baze. Konfiguracija baze definirana je za tri različita profila - testni, razvojni (eng.

*development*) i produkcijski (eng. *production*) i nalazi se u tri različite datoteke. To je bitno jer omogućuje odvojeno upravljanje konfiguracijom baze za različita okruženja. Profil se ubrizgava s pomoću Spring Boot *run* konfiguracije stavkom “Active Profiles”. Razvojni (eng. *development*) profil se koristi tijekom razvoja aplikacije i pri lokalnom pokretanju iste, test profil se koristi pri pokretanju testova kako bi baza uvijek bila čista i kako bi se spriječilo utjecanje testova na razvojne ili produkcijske podatke, a produkcijski profil se koristi pri pravom korištenju aplikacije jednom kada je *deployana*, u zadnjem slučaju koristila bi se jača baza podataka (PostgreSQL, MySQL ili slično) i postavila bi se strože sigurnosne postavke. Za svrhe ovog rada odlučeno je zadržati H2 bazu za sva tri profila pošto se aplikacija nije nigdje *deployala*.

U konfiguraciji aplikacije potrebno je specificirati driver baze, URL baze, *hibernate* dijalekt kao i putanje do SQL skripti za automatsku inicijalizaciju shema i početnih podataka.

Primjer konfiguracije lokalne H2 baze:

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:devdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=none
spring.sql.init.mode=always
spring.sql.init.schema-
locationsclasspath:schema/roles.sql,classpath:schema/users.sql,classpath:schema/notification_options.sql,classpath:schema/venue_types.sql,classpath:schema/venues.sql,classpath:schema/venue_ratings.sql,classpath:schema/reservations.sql,classpath:schema/working_days.sql,classpath:schema/venue_images.sql,classpath:schema/menu_images.sql
spring.sql.init.data-
locationsclasspath:data/roles.sql,classpath:data/venue_types.sql
```

Za interakciju s bazom koristi se Spring Data JPA, koji omogućuje jednostavno mapiranje Java/Kotlin objekata na tablice u bazi podataka kroz koncept entiteta. Svaki entitet predstavlja jednu tablicu u bazi dok repozitoriji omogućuju CRUD operacije i dodatne upite prema specifičnim kriterijima. Entiteti se nalaze unutar repo direktorija u dodatnom entity direktoriju.

Repozitoriji se nalaze u direktoriju repo i proširuju sučelje JpaRepository, čime se automatski dobivaju osnovne metode za rad s bazom, kao što su *save()*, *findById()*, *findAll()* i *delete()*. Također, moguće je definirati vlastite metode upita prema potrebama aplikacije.

Primjer repozitorija za korisnike koji je proširen s metodama `findByEmail()` i `findByIdIn()`:

```
interface UserRepository : JpaRepository<UserEntity, Int> {
    fun findByEmail(email: String): UserEntity?
    fun findByIdIn(ids: List<Int>): List<UserEntity>
}
```

Primjer entiteta koji mapira tablicu users. Sadrži polja id, username, email, password, lastKnownLatitude, lastKnownLongitude i roleId. Svako polje entiteta odgovara stupcu u tablici baze a anotacije `@Entity` i `@Table` omogućuju JPA da prepozna vezu između klase i tablice:

```
@Entity
@Table(name = "users")
class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Int = 0

    var username: String = ""

    var email: String = ""

    var password: String = ""

    var lastKnownLatitude: Double? = null

    var lastKnownLongitude: Double? = null

    var roleId: Int = 0
}
```

### 3.2.5 Testovi

Kako bi se osigurala ispravnost poslovne logike i stabilnost aplikacije, implementirani su jedinični (eng. *unit*) i integracijski testovi. Testovi su pisani koristeći JUnit5 kao testni okvir, dok se za oponašanje (eng. *mocking*) koristi Mockito.

Testovi su podijeljeni prema akcijama, dakle CreateReservationTest, DeleteReservationTest i slično, kako bi se održala preglednost i jasna struktura koda. Pomoću tog sustava imenovanja klasa moguće je i brzo uočiti koje su funkcionalnosti pokrivene i jednostavno je dodavati nove slučajeve za testiranje.

Jedinični testovi služe za izolirano testiranje servisa i poslovne logike bez komunikacije s bazom podataka ili vanjskim servisima. Testne klase proširuju apstraktne klase kao na primjer BaseReservationServiceTest, u kojima se

definiraju potrebne varijable, servisi i *mockovi*. Ovisnosti poput repozitorija se zamjenjuju *mockovima* (pomoću @Mock anotacije), a njihovo ponašanje se definira korištenjem Mockito metoda (when(), verify()). Za izvođenje testova s testnim profilom svaka apstraktna klasa je anotirana s @ActiveProfiles("test"), što osigurava izolirano okruženje od razvoja i produkcije.

Primjer jediničnog testa u kojem se testira stvaranje rezervacije na neradni dan nekog objekta:

```
@Test
fun `should return failure response when request is on non-working
day`() {
    `when`(`userRepository.findById(anyInt())`).thenReturn(Optional.of(mocked
User))

    `when`(`venueRepository.findById(anyInt())`).thenReturn(Optional.of(mocke
dVenue))

    `when`(`workingDaysRepository.findAllByVenueId(mockedVenue.id)`).thenRetu
rn(noWorkingDays)

    val response = reservationService.create(requestOnNonWorkingDay)

    response.success `should be equal to` false
    response.message `should be equal to` "The venue is closed on the
selected day. Please choose a different day."

    verify(userRepository, times(1)).findById(mockedUser.id)
    verify(venueRepository, times(1)).findById(mockedVenue.id)
    verify(workingDaysRepository,
times(1)).findAllByVenueId(mockedVenue.id)
    verifyNoMoreInteractions(userRepository, venueRepository,
reservationRepository, workingDaysRepository)
}
```

Integracijski testovi provjeravaju ispravnost cijelog sustava, uključujući komunikaciju s bazom podataka. Kao i u jediničnim testovima, testne klase se proširuju sa apstraktним klasama kao na primjer AbstractReservationIntegrationTest, koje pomoću @Autowired anotacije ubrizgavaju stvarne repozitorije u aplikaciju. Za testiranje se koristi H2 baza podataka unutar testnog okvira, čime se simulira rad aplikacije u realnom okruženju. Svaki test je anotiran s @Transactional, što osigurava da se promjene u bazi poništavaju nakon izvođenja testa, čime baza ostaje u konzistentnom stanju u slučaju da se svi testovi izvršavaju pomoću mvn clean install komande.

Primjer integracijskog testa u kojem se testira stvaranje rezervacije na neradni dan nekog objekta:

```
@Test
fun `should fail when non working day`() {
    val workingDays = createWorkingDays(venue.id, emptyList())
    workingDaysRepository.saveAllAndFlush(workingDays)

    val request = CreateReservationRequest(
        userId = customer.id,
        venueId = venue.id,
        reservationDate =
    LocalDateTime.now().withHour(10).withMinute(0),
        numberOfGuests = 2
    )

    val response: BasicResponse = reservationService.create(request)

    response.success `should be equal to` false
    response.message `should be equal to` "The venue is closed on the
selected day. Please choose a different day."
}
```

## 4 Korisničke upute

### 4.1 Korisničke upute za korištenje mobilne aplikacije

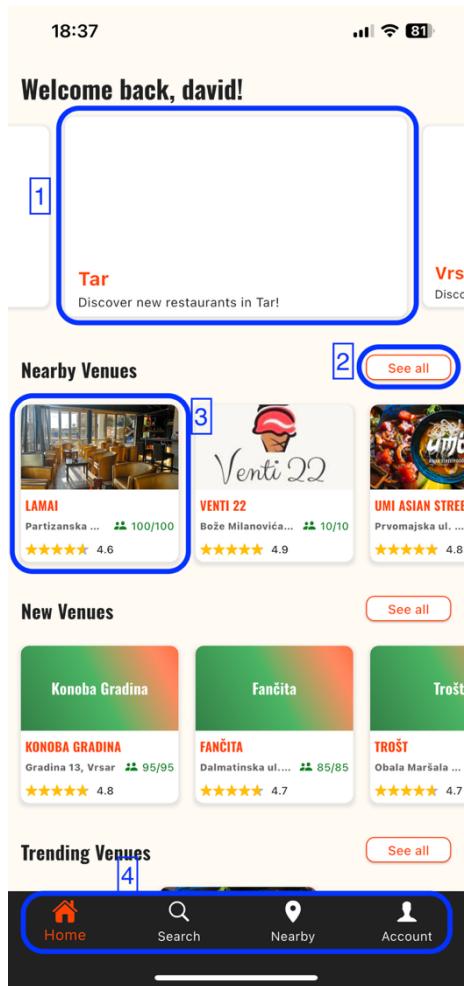
U ovim korisničkim uputama objašnjeno je korištenje aplikacije prijavljenom korisniku, iskustvo je poprilično slično i za neprijavljenog korisnika sa nekoliko iznimki. Neprijavljeni korisnik nema pristup funkcionalnostima koje zahtijevaju autentifikaciju a to su: stvaranje rezervacija, pregled povijesti rezervacija, uređivanje podataka korisničkog računa i pristup korisničkoj podršci. Osim toga, neki elementi korisničkog sučelja nisu dostupni, kao na primjer Carousel komponenta prikazan na slici 4 pod rednim brojem 1, koji se ne prikazuje jer za svoj rad koristi geolokacijske podatke korisnika.

Budući da integracija s Google Places API-jem iziskuje dodatne troškove, slike u *carousel* elementu nisu uključene, isto tako nije prikazana Nearby stranica zbog integracije Google Maps elementa. Ako ugostiteljski objekt nema pridružene slike, prikazuje se njegovo ime na gradijentnoj pozadini kako bi se osiguralo da aplikacija ostaje vizualno konzistentna.

#### 4.1.1 Početna stranica

Glavni elementi početne stranice prikazani na slici 4:

1. Carousel komponenta - Carousel komponenta služi za prikaz obližnjih gradova dohvaćenih pomoću koordinata korisnika. Korisnik može pritisnuti grad koji želi pretražiti čime se otvara stranica pretrage gdje će se ugostiteljski objekti automatski filtrirati po odabranom gradu.
2. See all gumbovi - postoji jedan gumb za svaku "kategoriju" (*nearby, new, trending, suggested*). Ako korisnik pritisne taj gumb, otvara se stranica sa svim ugostiteljskim objektima koji spadaju u odabranu kategoriju, čime se proširuje pregled koji je inicijalno dostupan na početnoj stranici.
3. Kartica ugostiteljskog objekta - Na ovom elementu korisnik može vidjeti ime, lokaciju, prosječnu ocjenu i zauzetost objekta. Kada korisnik pritisne ovaj element preusmjeruje ga se na stranicu samog objekta.
4. Navigacija - Navigacijski elementi omogućavaju korisniku lako kretanje kroz različite stranice aplikacije.



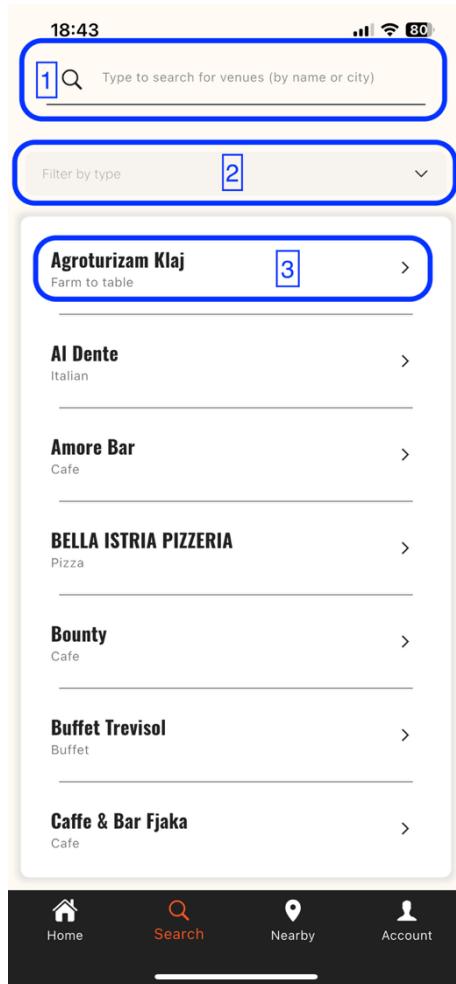
Slika 4 Elementi početne stranice

#### 4.1.2 Stranica pretraživanja

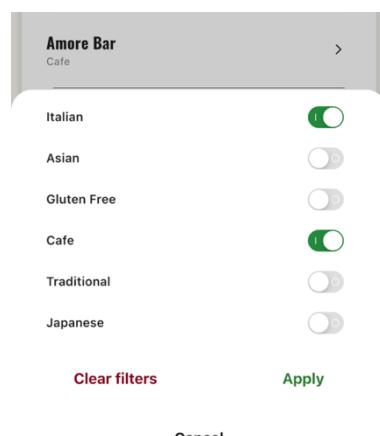
Glavni elementi stranice pretrage i filtriranja objekata prikazani na slici 5:

1. Traka za pretraživanje (eng. *search bar*) - Traka za pretraživanje omogućuje korisniku da pronađe određeni ugostiteljski objekt prema njegovom imenu ili lokaciji. Nakon što korisnik unese pojам u traku, sustav automatski filtrira popis objekata i prikazuje samo one koje odgovaraju unesenom upitu. Time se način značajno ubrzava pronalazak željenog objekta.
2. Filter - Ovaj element omogućuje dodatno filtriranje prema vrsti ugostiteljskog objekta (npr. kafić, restoran). Klikom na dropdown komponentu, prikazuje se dialog s popisom vrsta objekta i korisnik može jednostavno odabrat tipove objekata koje želi filtrirati što je prikazano na slici 6. Kada se dialog zatvori sustav automatski filtrira popis objekta prema odabranim tipovima.

3. Element popisa - Glavni dio stranice zauzima popis svih ugostiteljskih objekata. Korisnik može pregledavati dostupne objekte i pronaći onog kojeg ga zanima. Klikom na odabrani objekt otvara se stranica s detaljnim informacijama o njemu.



Slika 5 Elementi stranice pretrage i filtriranja objekata

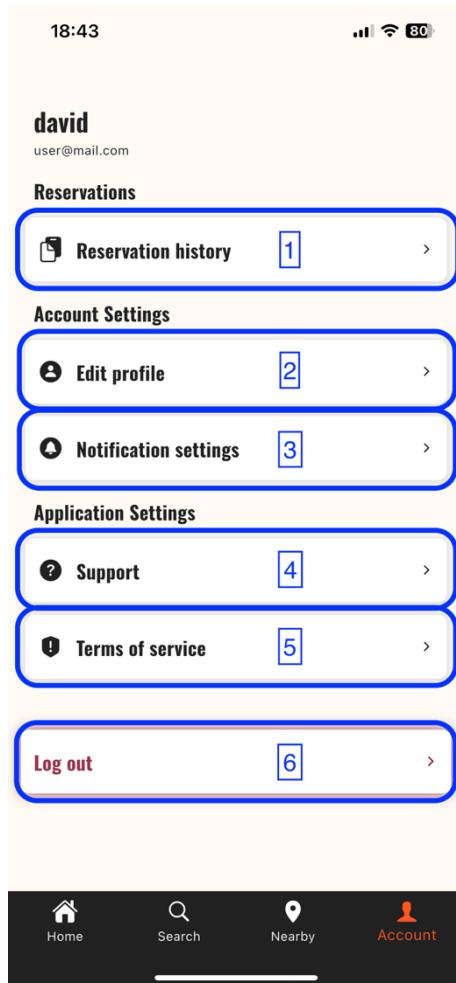


Slika 6 Filter dialog

#### **4.1.3 Stranica korisničkog računa**

Glavni elementi stranice korisničkog računa prikazani na slici 7:

1. Gumb povijesti rezervacija - Klikom na ovaj gumb korisnik pristupa stranici na kojoj može pregledati povijest svojih rezervacija.
2. Gumb promjene korisničkih podataka - Klikom na ovaj gumb korisnik pristupa stranici na kojoj može promjeniti svoje korisničke podatke.
3. Gumb postavki obavijesti - Klikom na ovaj gumb korisnik pristupa stranici na kojoj može promjeniti postavke obavijesti i dopuštenja aplikaciji.
4. Gumb korisničke podrške - Klikom na ovaj gumb korisnik pristupa stranici za korisničku podršku gdje može ili pročitati FAQ ili poslati upit razvojnog timu.
5. Gumb uvjeta korištenja - Klikom na ovaj gumb korisnik može pregledati uvjete korištenja.
6. Gumb za odjavu - Klikom na ovaj gumb, korisnik se može odjaviti iz aplikacije.  
Ako korisnik nije prijavljen, ovaj element mijenja tekst u „Log In“ čime se nudi mogućnost prijave.

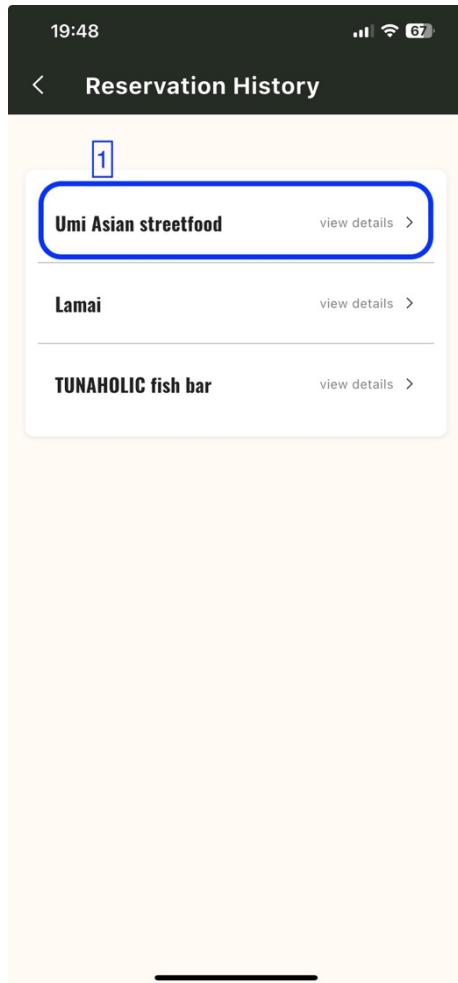


Slika 7 Elementi stranice korisničkog računa

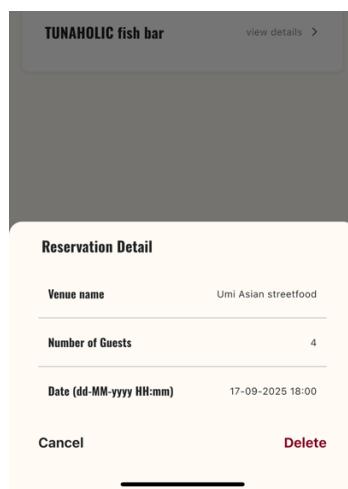
#### 4.1.4 Stranica povijesti rezervacija

Glavni elementi stranice povijesti rezervacija prikazani na slici 8:

1. Element popisa - Na stranici povijesti rezervacija nalazi se popis svih rezervacija povezanih s korisničkim računom. Kada se jedan element popisa dotakne prikazuje se dialog s detaljima rezervacije - ime objekta, broj gostiju i datum i vrijeme rezervacije, prikazano na slici 9.



Slika 8 Elementi stranice povijesti rezervacija



Slika 9 Detalji rezervacije

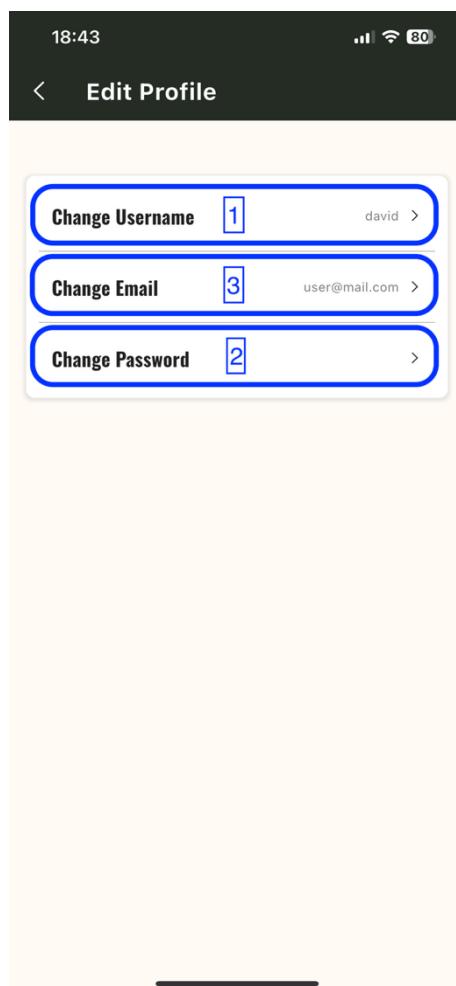
#### 4.1.5 Stranica za izmjenu korisničkih podataka

Glavni elementi stranice za izmjenu korisničkih podataka prikazani na slici 10:

1. Gumb za promjenu korisničkog imena - Na desnoj strani gumba prikazuje te trenutno korisničko ime. Ako ga korisnik želi promijeniti, pritiskom na gumb

otvara se dialog s pomoću kojeg korisnik može unijeti novo korisničko ime i zamijeniti postojeće.

2. Gumb za promjenu email adrese - Na desnoj strani gumba prikazuje se trenutna email adresa. Ako ju korisnik želi promijeniti, pritiskom na gumb otvara se dialog s pomoću kojeg korisnik može unijeti novu email adresu i zamijeniti postojeću.
3. Gumb za promjenu lozinke - pritiskom na gumb otvara se dialog s pomoću kojeg korisnik može unijeti novu lozinku i potvrditi ju, čime se stara lozinka zamjenjuje.

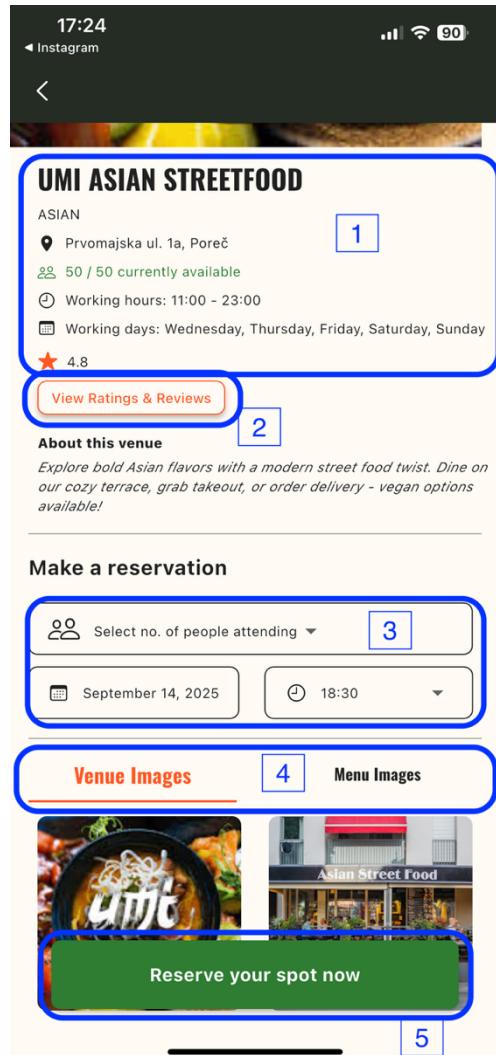


Slika 10 Elementi stranice za izmjenu korisničkih podataka

#### 4.1.6 Stranica ugostiteljskog objekta

Glavni elementi stranice ugostiteljskog objekta prikazani na slici 11:

1. Detalji ugostiteljskog objekta - Na vrhu stranice korisnik može pročitati detalje objekta poput imena, tipa, lokacije, trenutačne zauzetosti, radno vrijeme, prosječna ocjena i opis ugostiteljskog objekta.
2. Gumb za pregled recenzija - Klikom na ovaj gumb korisnik prelazi na stranicu s prikazom svih recenzija za odabrani ugostiteljski objekt. Osim pregleda, korisniku je omogućeno i dodavanje vlastite recenzije.
3. Gumbovi za odabir broja gostiju, datuma i vremena rezervacije - Ovi elementi omogućuju korisniku unos potrebnih parametra za stvaranje rezervacije. Sustav pritom validira unos, rezervacija se može napraviti na radni dan i u vrijeme kada je objekt otvoren. Ako korisnik pokuša odabratи nevažeći termin, prikazuje se odgovarajuća obavijest.
4. *Tab* sustav za pregled slika objekta i menija - Na ovoj stranici korisnik može i pregledati galeriju slika ugostiteljskog objekta kao i galeriju slika menija. Ta je funkcionalnost omogućena *Tab* sustavom.
5. Gumb za stvaranje rezervacije - Klikom na ovaj gumb korisnik šalje zahtjev za stvaranje rezervacije. Ako korisnik nije ulogiran, ili nije popunio sva potrebna polja za stvaranje rezervacije, aplikacija prikazuje notifikaciju u obliku *toastera*, koja informira korisnika o razlogu neuspješne rezervacije.

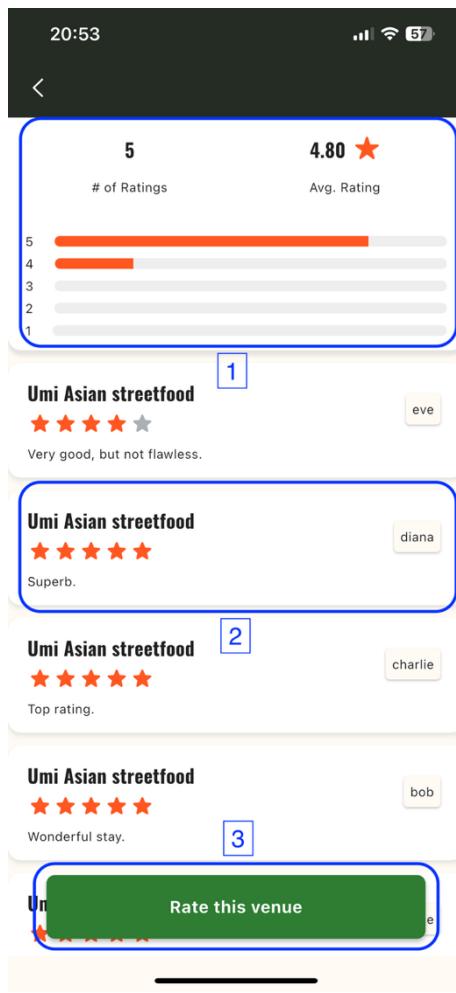


Slika 11 Elementi stranice ugostiteljskog objekta

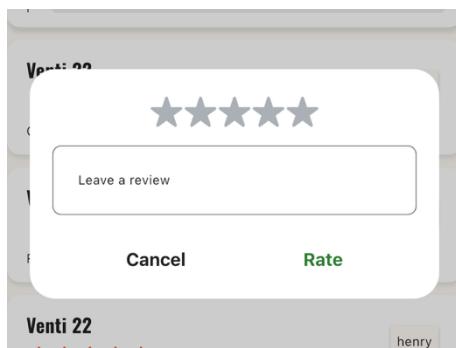
#### 4.1.7 Stranica recenzija

Glavni elementi stranice za recenzije prikazani na slici 12:

1. Sažetak podataka recenzija - U ovom elementu prikazuje se ukupan broj recenzija, prosječna ocjena i raspodjela ocjena od jedan do pet.
2. Recenzija - U nastavku stranice nalazi se popis svih pojedinačnih recenzija koje su ostavili korisnici. Svaka recenzija sadrži ocjenu i komentar, a time se osigurava transparentnost i daje korisnicima mogućnost usporedbe različitih iskustava.
3. Gumb za recenziranje - Pritiskom na ovaj gumb otvara se dialog pomoću kojeg korisnik može odabratи ocjenu (od jedan do pet) i pustiti kratak komentar što je prikazano na slici 13.



Slika 12 Elementi stranice za recenzije



Slika 13 Dialog za recenziranje ugostiteljskog objekta

## 4.2 Korisničke upute za korištenje web aplikacije (administrativnog sučelja)

### 4.2.1 Početna stranica

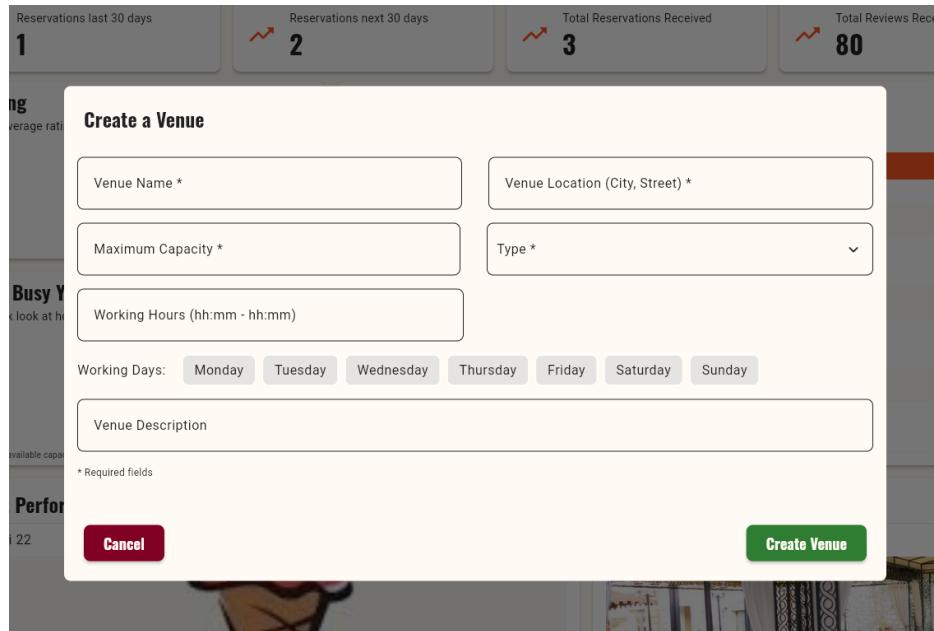
Glavni elementi početne stranice prikazani na slici 14:

1. Navigacija - Navigacija je prisutna na svim glavnim stranicama i omogućuje korisnicima laganu i intuitivnu navigaciju po aplikaciji.
2. *Switch* gumbovi za odabir teme - Pomoću ovih gumbova korisnik može mijenjati izgled aplikacije birajući između svijetle i tamne teme.
3. Podaci vezani za rezervacije i recenzije - Pomoću ovih elementa korisniku se odmah prikazuju bitne metrike vezane za poslovanje. Prikazuju se broj rezervacija u prošlim 30 dana, sljedećih 30 dana, ukupan broj primljenih rezervacija i broj recenzija.
4. Kružni pokazatelji - Pomoću ovih kružnih pokazatelja korisnik može vidjeti ukupnu prosječnu ocjenu svih svojih objekata i ukupnu zauzetost u stvarnome vremenu.
5. Tablica sa registriranim objektima - U ovoj tablici se korisniku prikazuju svi objekti koje je registrirao uz pripadajuće podatke o svakom objektu. Klikom na redak u tablici otvara se stranica tog objekta, dok pomoću *action buttona* na desnoj strani tablice, korisnik može ili izmijeniti podatke odabranog objekta ili ga obrisati. Iznad tablice se nalazi i gumb za dodavanje novog objekta koji otvara modal pomoću kojeg se registrira novi ugostiteljski objekt što je prikazano na slici 15.
6. Najbolje i najgore recenzirani objekti - U ovom elementu su prikazani najbolje i najgore ocjenjeni objekti čime korisnik može brzo procijeniti stanje i zadovoljstvo gostiju.

The screenshot shows the Admin Dashboard interface with the following sections and numbered callouts:

- Platform Navigation:** Includes links for Dashboard, Venues, RSVP, Reservations, Settings, and Account. Callout 1 points to the Dashboard link.
- Light/Dark Mode:** Buttons for Light Mode and Dark Mode. Callout 2 points to the Light Mode button.
- Overview:** Displays metrics for reservations and reviews. Callout 3 points to the 'Create a Venue' button.
- Rating:** Shows average rating across all venues. Callout 4 points to the 'View details' button.
- Your Venues:** A table listing venues with columns for Name, Location, Max. capacity, Avail. capacity, and Rating. Callout 5 points to the edit icon for the first venue.
- How Busy You Are:** Shows current busyness level at 0.0%. Callout 6 points to the 'View details' button.
- Best Performing Venue:** Features Venti 22 with a rating of 4.90 and an illustration of an ice cream cone.
- Worst Performing Venue:** Features Dva Feralia with a rating of 3.80 and a photograph of an outdoor seating area.

Slika 14 Elementi početne stranice



Slika 15 Modal za registriranje novog ugostiteljskog objekta

## 4.2.2 Stranica ugostiteljskih objekata

Glavni elementi stranice objekata prikazani na slici 16:

1. Kartica ugostiteljskog objekta - Ova se stranica sastoji od vizualnog popisa kartica, pri čemu svaka kartica predstavlja jedan registrirani objekt. Klikom na željenu karticu korisnika se preusmjerava na stranicu objekta.

Slika 16 Elementi stranice objekata

### 4.2.3 Stranica rezervacija

Glavni elementi stranice rezervacija prikazani na slici 17:

1. Detalji rezervacije - U tabličnom prikazu korisniku se prikazuju detalji svih rezervacija. Za svaku rezervaciju navedeni su ime objekta, email adresa korisnika koji je napravio rezervaciju, broj gostiju te datum i vrijeme rezervacije.
2. Gumbovi za upravljanje rezervacijom - Za svaku rezervaciju dostupni su gumbovi za uređivanje i brisanje rezervacije.
3. Gumb za stvaranje rezervacije - Pomoću ovog gumba korisnik može stvoriti novu rezervaciju. Klikom na gumb, otvara se modal prikazan na slici 18 i korisnik može unijeti sve potrebne podatke.

Your Reservations

Venue Name	User email	Number of Guests	Reservation Date
Venti 22	user@mail.com	2	16 September, 2025 - 19:33
Umi Asian streetfood	user@mail.com	4	17 September, 2025 - 18:00
TUNAHOLIC fish bar	user@mail.com	5	21 September, 2025 - 19:30

Auto refresh every: Off Refresh data Create a Reservation

Venue Navigation Reservations Settings Light Mode Dark Mode

Slika 17 Elementi stranice rezervacija

Your Reservations

Venue Name	User email	Number of Guests	Reservation Date
Lamai	user@mail.com	2	14 September, 2025 - 20:30
Umi Asian streetfood	user@mail.com	4	17 September, 2025 - 18:00
TUNAHOLIC			

Create a Reservation

Venue \* User Email \*  
Number of Guests \* Reservation Date \*  
\* Required fields

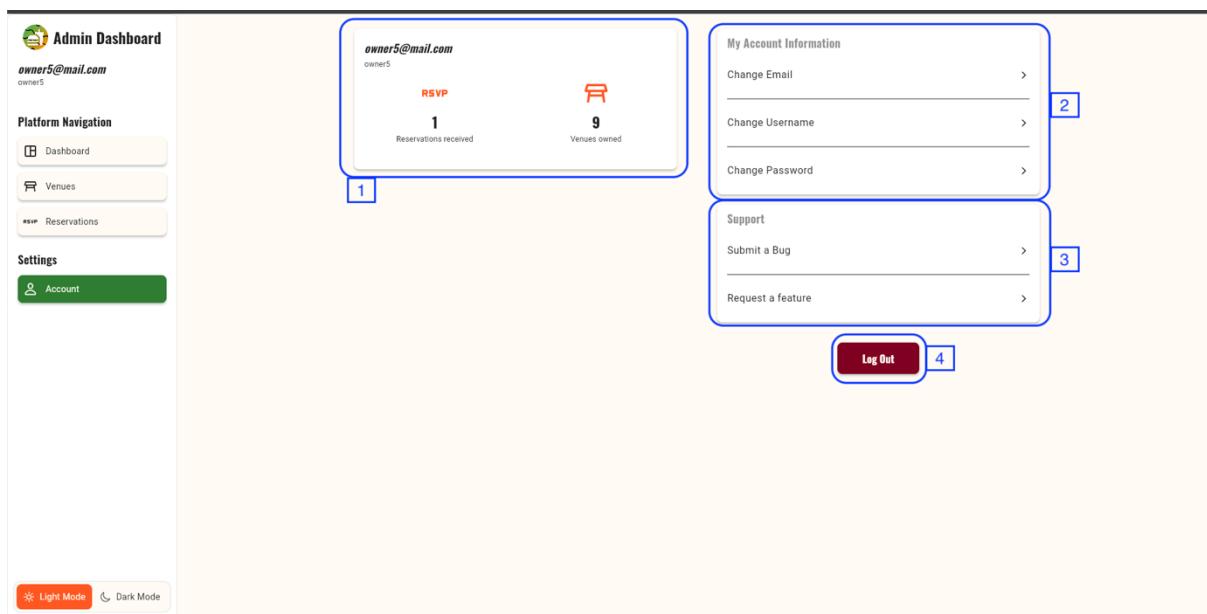
Cancel Create Reservation

Slika 18 Modal za stvaranje rezervacije

#### 4.2.4 Stranica korisničkih podataka

Glavni elementi stranice korisničkog računa prikazani na slici 19:

1. Detalji korisnika - Na ovom djelu stranice korisnik može vidjeti svoju email adresu i korisničko ime te broj ukupno primljenih rezervacija i broj registriranih objekata.
2. Gumbovi za promjenu korisničkih podataka - Klikom na ove gumbove otvara se pripadajuć modal komponenta za izmjenu email adrese, korisničkog imena ili lozinke.
3. Gumbovi za upite korisničkoj službi - Klikom na ove gumbove korisnik može prijaviti smetnju ili zatražiti novu funkcionalnost unutar aplikacije pomoću pripadajuće modal komponente.
4. Gumb za odjavu - Klikom na ovaj gumb korisnik se odjavljuje iz aplikacije i zatvara se trenutačna sesija.



Slika 19 Elementi stranice korisničkog računa

#### 4.2.5 Stranica ugostiteljskog objekta

Glavni elementi stranice objekta prikazani na slici 20:

1. Detalji objekta: U ovome elementu prikazani su podaci o objektu kao što su ime, opis, lokacija, radno vrijeme, maksimalni kapacitet, dostupni kapacitet i ukupan broj rezervacija u objektu.

2. Tab gumb za pristup recenzijama objekta - Klikom na ovaj gumb korisnik može pregledati recenzije za odabrani objekt što je prikazano na slici 22.
3. Tab gumb za pristup slikama objekta - Klikom na ovaj gumb korisnik može pregledati sve slike odabranog objekta , uključujući slike menija što je prikazano na slici 23. Na ovom tabu korisnik može i pomoću dva gumba dodati slike objekta ili menija.
4. Gumb za izmjenu detalja objekta - Klikom na ovaj gumb korisnik može, pomoću pripadajuće modal komponente, izmjeniti podatke odabranog objekta, što je prikazano na slici 21.

The screenshot shows a web page for a venue named 'Cotton Club'. At the top, there is a navigation bar with a back arrow and the venue name. Below the navigation is a large image of a bar interior with wooden counters, shelves filled with bottles, and a well-stocked bar area. Underneath the image, the venue name 'Cotton Club' is displayed again, followed by a blue button labeled 'Edit Venue Details' with a pencil icon and a small number '4' indicating pending changes.

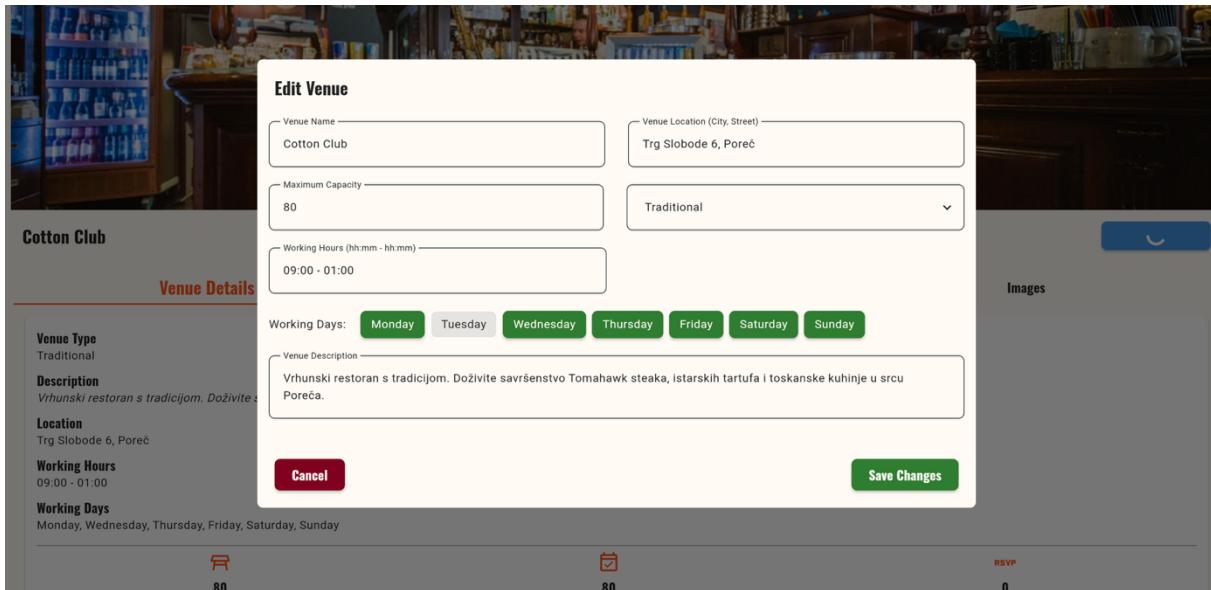
Below the main image, there are three tabs: 'Venue Details' (selected), 'Reviews' (with a count of 2), and 'Images' (with a count of 3). The 'Venue Details' tab contains the following information:

- Venue Type:** Traditional (indicated by a blue box with the number 1)
- Description:** Vrhunski restoran s tradicijom. Doživite savršenstvo Tomahawk steaka, istarskih tartufa i toskanske kuhinje u srcu Poreča.
- Location:** Trg Slobode 6, Poreč
- Working Hours:** 09:00 - 01:00
- Working Days:** Monday, Wednesday, Thursday, Friday, Saturday, Sunday

At the bottom of the 'Venue Details' section, there are three metrics:

- Maximum Capacity:** 80 (indicated by a red icon and the number 80)
- Available Capacity:** 80 (indicated by a green icon and the number 80)
- Lifetime Reservations:** 0 (indicated by an orange icon and the number 0)

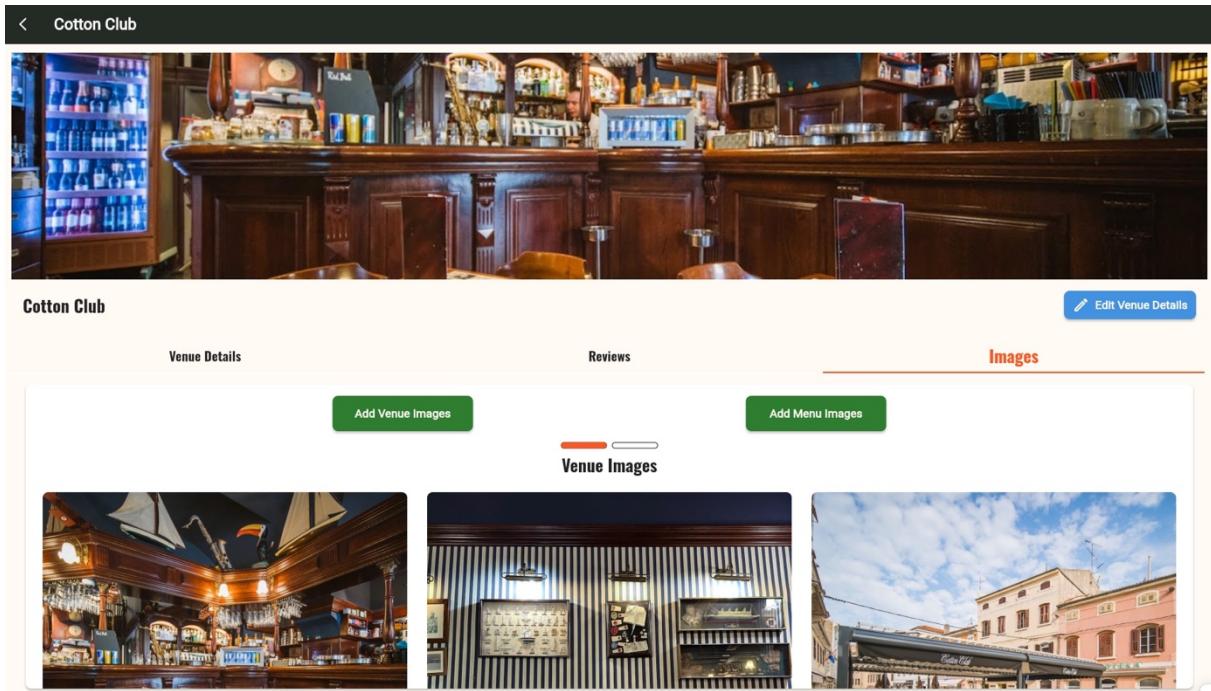
Slika 20 Elementi stranice objekta



Slika 21 Modal za izmjenu podataka ugostiteljskog objekta

Total number of reviews:	10
Average Rating	<span>4.6</span>
jack	<span>4.0</span>
Enjoyable, but not perfect.	
irene	<span>4.0</span>
Good overall.	
henry	<span>4.0</span>
Nice stay.	

Slika 22 Tab za pregled recenzija



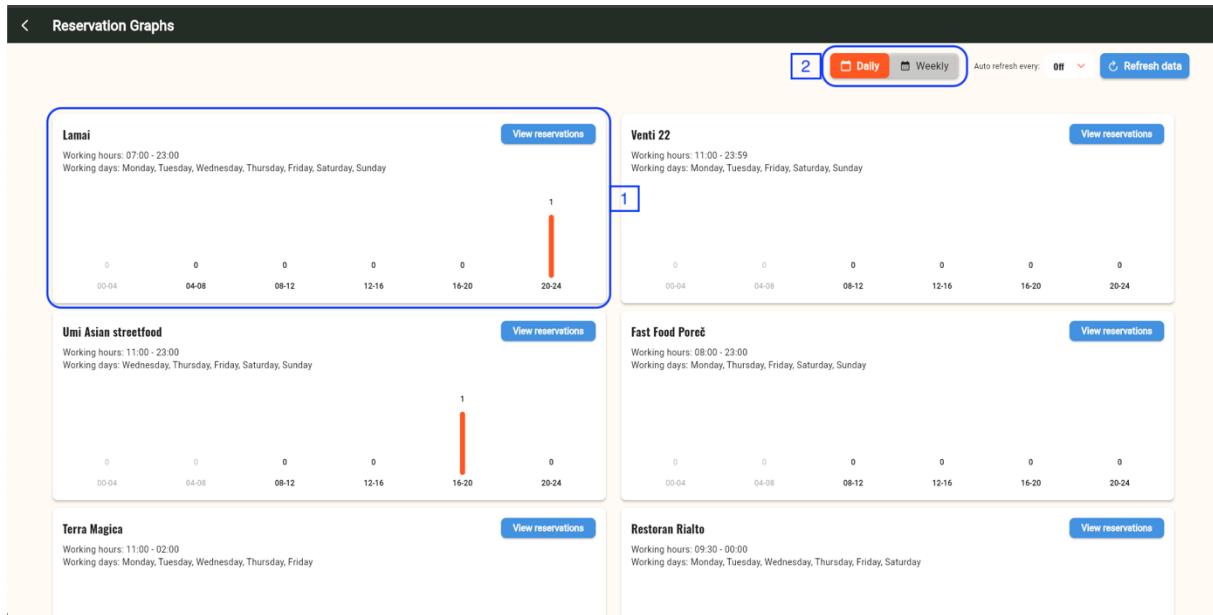
Slika 23 Tab za pregled slika objekta

#### 4.2.6 Stranica grafova rezervacija

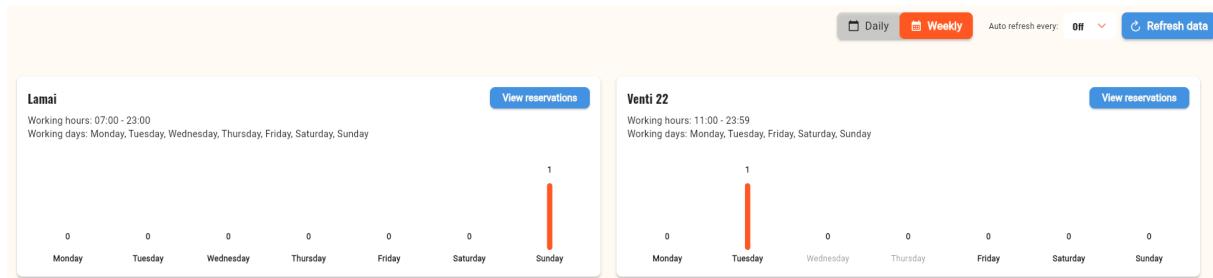
Glavni elementi stranice grafova rezervacija prikazani na slici 24:

1. Kartica s grafom - U ovome elementu se nalazi grafički prikaz rezervacija, koji može biti u dnevnom ili tjednom prikazu. Grafovi omogućavaju korisniku uvid u najatraktivnija vremena za klijente, pomažući u planiranju i optimizaciji poslovanja. Na svakoj kartici se nalazi i gumb pomoću kojeg se preusmjerava korisnika na stranicu rezervacija te mu se nudi pregled rezervacija za odabrani objekt.
2. *Switch* gumb za promjenu vizualizacije - Pomoću ovog gumba korisnik može mijenjati iz prikaz grafa između dnevnog i tjednog prikaza.

Na slici 25 prikazan je tjedni prikaz grafova.



Slika 24 Elementi stranice grafova rezervacija

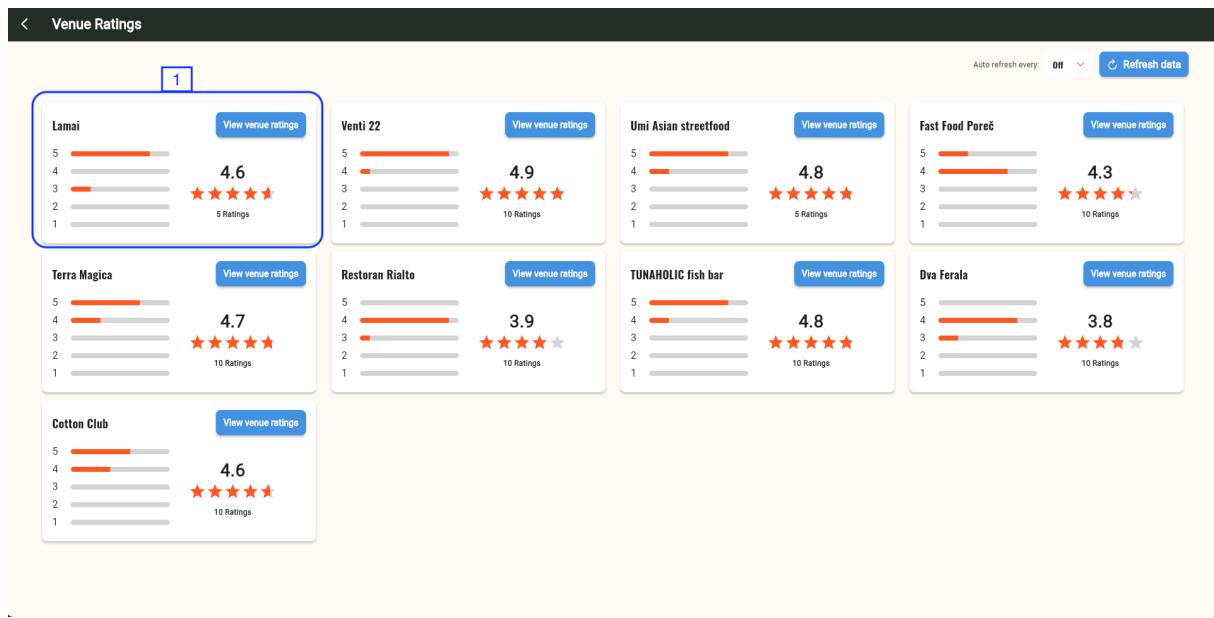


Slika 25 Pregled grafova po danima u tjednu

## 4.2.7 Stranica recenzija

Glavni elementi stranice recenzija prikazani na slici 26:

1. Kartica recenzija - Svaka kartica predstavlja pojedini objekt i sadrži ključne informacije o recenzijama: broj recenzija, prosječnu ocjenu te grafikon koji prikazuje učestalost ocjena od jedan do pet.



Slika 26 Elementi stranice recenzija

## 5 Zaključak

Digitalizacija i modernizacija poslovnih procesa danas su neizbjježan dio gotovo svake industrije, pa tako i ugostiteljstva. Posebno važan segment je transparentno upravljanje kapacitetom i dostupnosti informacija o zauzetosti objekta, što može izravno utjecati na učinkovitost poslovanja. U tom kontekstu *SeatSaver* aplikacija razvijena u okviru ovog rada predstavlja konkretni primjer kako se modernim tehnologijama može omogućiti jednostavan i pouzdan sustav rezervacije mesta u ugostiteljskim objektima.

Kroz implementaciju Fluttera za korisničko sučelje, Kotlin i Spring Boot za *backend* i H2 baze podataka, ostvareno je rješenje koje povezuje krajnje korisnike i vlasnike objekata u jedinstven ekosustav. Krajnjim korisnicima aplikacija omogućuje brzo pronalaženje i rezervaciju mesta, dok vlasnicima pruža alate za praćenje kapaciteta, rezervacija i recenzija, čime se postiže bolja organizacija poslovanja i povećanje zadovoljstva gostiju.

Iako je postignut glavni cilj rada, aplikacija ima značajan potencijal za daljnji razvoj, od unaprjeđenja postojećih funkcionalnosti do implementacije novih. Jedna od većih nadogradnji bio bi prelazak s H2 baze podataka na robusniju PostgreSQL bazu, čime bi se osigurala veća skalabilnost i pouzdanost. Dodatni prostor za razvoj mobilne aplikacije leži u implementaciji personaliziranih preporuka i mogućnosti naručivanja i plaćanja kroz aplikaciju, dok bi se web aplikacija mogla proširiti s naprednjim grafovima i analitikama.

## 6 Literatura

1. Eat App, "The 13 Best Online Restaurant Reservation Systems", 2025. [Na internetu] Dostupno: <https://restaurant.eatapp.co/blog/online-restaurant-reservation-systems> [pristupano 9.9.2025.]
2. Quandoo, "How 'Reserve with Google' Brings More Bookings to Your Restaurant", 2025 [Blog post] Dostupno: <https://restaurants.quandoo.com/en-au/blog/reserve-with-google-for-restaurants> [pristupano: 9.9.2025.]
3. GeeksForGeeks, "Market Analysis: Meaning, Inclusions, Importance & Drawbacks", 2025. [Na internetu]. Dostupno: <https://www.geeksforgeeks.org/marketing/market-analysis-meaning-inclusions-importance-drawbacks/> [pristupano: 9.9.2025.]
4. Flutter, dostupno: <https://docs.flutter.dev/#docs> [pristupano: 10.9.2025.]
5. Bhatt, M., "Understanding Flutter Widget Lifecycle: A Complete Developer's Guide", 2025. [Na internetu] Dostupno: <https://medium.com/@bhattmeet887/understanding-flutter-widget-lifecycle-a-complete-developers-guide-ec736f9bccf5> [pristupano 10.0.2025.]
6. Łojniewski, C., "Flutter vs React Native 2025: What's Better for a Cross-platform App", 2025. [Na internetu] Dostupno: <https://medium.com/pagepro/flutter-vs-react-native-2025-whats-better-for-a-cross-platform-app-cc85c9fad4cb> [pristupano 10.9.2025.]
7. Kotlin, dostupno: <https://kotlinlang.org/docs/home.html> [pristupano: 10.9.2025.]
8. Omnilab, "Java vs Kotlin for Backend and API Development" [Na internetu] Dostupno: <https://omnilabes.com/topics/java-vs-kotlin-for-backend-and-api-development/> [pristupano: 10.9.2025.]
9. Spring Boot, dostupno: <https://spring.io/projects/spring-boot> [pristupano: 10.9.2025.]
10. GeeksForGeeks, "Spring Boot with H2 Database", 2025, [Na internetu] Dostupno: <https://www.geeksforgeeks.org/springboot/spring-boot-with-h2-database/> [pristupano: 10.9.2025.]
11. GeeksForGeeks, "H2 Database vs PostgreSQL: What are the differences?", 2025, [Na internetu] Dostupno: <https://www.geeksforgeeks.org/blogs/mysql-vs-h2/> [pristupano: 10.9.2025.]

12. GeeksForGeeks, “JDBC (Java Database Connectivity)”, 2025. [Na internetu]  
Dostupno: <https://www.geeksforgeeks.org/java/introduction-to-jdbc/>  
[pristupano: 10.9.2025.]
13. GeeksForGeeks, “JPA - Introduction”, 2025. [Na internetu] Dostupno:  
<https://www.geeksforgeeks.org/java/jpa-introduction/> [pristupano: 10.9.2025.]
14. Baeldung, “A Comparisong Between JPA and JDBC”, 2025. [Na internetu]  
Dostupno: <https://www.baeldung.com/jpa-vs-jdbc> [pristupano: 10.9.2025.]

## 7 Popis slika

Slika 1 Razlike MySQL i H2 [11].....	11
Slika 2 Struktura korisničkog sučelja.....	14
Slika 3 Struktura backend djela projekta.....	25
Slika 4 Elementi početne stranice .....	34
Slika 5 Elementi stranice pretrage i filtriranja objekata .....	35
Slika 6 Filter dialog.....	35
Slika 7 Elementi stranice korisničkog računa .....	37
Slika 8 Elementi stranice povijesti rezervacija .....	38
Slika 9 Detalji rezervacije .....	38
Slika 10 Elementi stranice za izmjenu korisničkih podataka .....	39
Slika 11 Elementi stranice ugostiteljskog objekta.....	41
Slika 12 Elementi stranice za recenzije.....	42
Slika 13 Dialog za recenziranje ugostiteljskog objekta .....	42
Slika 14 Elementi početne stranice .....	43
Slika 15 Modal za registriranje novog ugostiteljskog objekta.....	44
Slika 16 Elementi stranice objekata .....	44
Slika 17 Elementi stranice rezervacija .....	45
Slika 18 Modal za stvaranje rezervacije .....	45
Slika 19 Elementi stranice korisničkog računa .....	46
Slika 20 Elementi stranice objekta .....	47
Slika 21 Modal za izmjenu podataka ugostiteljskog objekta .....	48
Slika 22 Tab za pregled recenzija.....	48
Slika 23 Tab za pregled slika objekta .....	49
Slika 24 Elementi stranice grafova rezervacija .....	50
Slika 25 Pregled grafova po danima u tjednu .....	50
Slika 26 Elementi stranice recenzija .....	51

## Sažetak

Cilj ovog diplomskog rada je izrada mobilne aplikacije za rezervaciju raspoloživih mesta u ugostiteljskim objektima. Aplikacija, nazvana *SeatSaver*, osmišljena je kao cijelovito rješenje koje povezuje krajnje korisnike i vlasnike objekata. Krajnjim korisnicima omogućuje jednostavno pretraživanje i rezervaciju, dok vlasnicima pruža administrativno sučelje za upravljanje kapacitetom, rezervacijama i recenzijama.

Za razvoj korisničkog sučelja aplikacije korišten je Flutter okvir, dok je *backend* implementiran u Kotlinu i Spring Bootu, uz H2 bazu za pohranu podataka. Posebna pažnja posvećena je testiranju sustava, i korisničkog sučelja i *backenda*, kako bi se osigurala stabilnost i pouzdanost aplikacije. Aplikacija je razvijena modularno s naglaskom na skalabilnost i mogućnost daljnog razvoja.

Ključne riječi: Flutter, Kotlin, Spring Boot, H2, mobilne aplikacije, web aplikacije

## **Abstract**

The goal of this thesis is to create a mobile application for booking available seats in catering establishments. The application, called *SeatSaver*, is designed as a complete solution that connects end users and owners of establishments. It allows end users to easily search and book, while providing owners with an administrative interface for managing capacities, reservations and reviews.

The Flutter framework was used to develop the frontend part of the application, while the backend was implemented using Kotlin and Spring Boot, with an H2 database for data storage. Special attention was paid to the testing system, both frontend and backend, to ensure the stability and reliability of the application. The application was also developed modularly with an emphasis on scalability and the possibility of further development.

Keywords: Flutter, Kotlin, Spring Boot, H2, mobile applications, web applications