SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042

RESTAURANT TABLE RESERVATION SYSTEM

23CS503 – APP DEVELOPMENT

A PROJECT REPORT

Submitted by

ARTHUR BALAJI R - 727823TUCS018

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

AUGUST 2025

SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042

BONAFIDE CERTIFICATE

Certified that this project report **"RESTAURANT TABLE RESERVATION SYSTEM"** is the bonafide work of **"ARTHUR BALAJI R"** who carried out the project work under my supervision.

SIGNATURE

Mr. P. SURESH

SUPERVISOR

Assistant Professor,
Department of Computer Science and Engineering
Sri Krishna College of Technology,
Coimbatore-641042.

SIGNATURE

Dr. M. UDHAYAMOORTHI

HEAD OF THE DEPARTMENT

Associate Professor,
Department of Computer Science and Engineering
Sri Krishna College of Technology,
Coimbatore-641042.

Certified that the candidates were examined by us in the Project Viva Voce examination held on_____at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641042

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, **Dr. M.G. SUMITHRA,** for providing all facilities.

With the grateful heart, our sincere thanks to our Head of the Department **Dr. M. UDHAYAMOORTHI**, Department of Computer Science and Engineering for the motivation and all support to complete the project work.

We are greatly indebted to our Industry Mentor **Mr . SELVARAJ K** for his valuable guidance and suggestions in all aspects that aided us to ameliorate our skills.

We thank **Mr. P. SURESH ,** Department of Computer Science and Engineering, for his motivation and support

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Computer Science and Engineering and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our family members and our beloved friends, who had been strongly supporting us in all our endeavour.

ABSTRACT

The Restaurant Table Reservation System is a web-based application designed to provide users with a seamless and efficient dining reservation experience. The system allows customers to search for restaurants, create and manage reservations, and receive notifications regarding their bookings. Restaurant owners can manage their restaurant profiles, update table availability, and handle reservation requests, while administrators oversee user management and system reports. The backend is built using Spring Boot, offering robust validation, data consistency, and error handling, while the frontend leverages React.js to deliver an interactive and intuitive user interface. MySQL is used as the database, ensuring reliable and secure data storage for all users, restaurants, and reservations.

The backend enforces business rules such as table capacity checks, prevention of double bookings, and proper role-based access control, ensuring that reservations are processed accurately and users interact with the system according to their roles. The frontend features responsive dashboards, easy-to-use reservation forms, and clear feedback for both successful and unsuccessful actions. Communication between the client and server is managed via RESTful APIs, enabling real-time updates and smooth user interactions.

This project demonstrates the integration of modern web technologies to build a dependable, scalable, and user-centric table reservation system. It emphasizes the importance of validation, efficient workflow, and secure data handling in service-oriented applications, while laying a solid foundation for future enhancements such as advanced analytics, loyalty programs, and mobile app integration.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	ACRONYM
HTML	HYPERTEXT MARKUP LANGUAGE
CSS	CASCADING STYLESHEET
JS	JAVASCRIPT
DBMS	DATABASE MANAGEMENT SYSTEM
HTTP	HYPERTEXT TRANSFER PROTOCOL
API	APPLICATION PROGRAMMING INTERFACE
REST	REPRESENTATIONAL STATE TRANSFER
DOM	DOCUMENT OBJECT MODEL

CHAPTER 1
INTRODUCTION

The Restaurant Table Reservation System is a streamlined reservation platform that enables users to book tables at restaurants efficiently and conveniently. It offers core features such as searching for restaurants, creating and managing reservations, and viewing booking history. Restaurant owners can oversee their establishments and handle reservation requests through the system. Built with a Spring Boot backend and a React.js frontend, the application ensures both robustness and an intuitive user experience, while MySQL serves as the secure and reliable database for storing all relevant data.

1.1 Problem Statement

Traditional restaurant reservation methods can be inconvenient, time-consuming, and susceptible to errors due to manual processes. Customers often struggle to check table availability, manage their bookings, or receive timely confirmations, while restaurant staff face challenges in tracking reservations and preventing overbooking. Without a streamlined digital solution, ensuring real-time updates, efficient reservation management, and accurate communication becomes difficult. Therefore, there is a need for an intuitive online system that offers easy table booking, transparent reservation tracking, and reliable validation for both customers and restaurant owners.

1.2 Overview of the Project

The proposed system integrates a web-based user interface with a secure backend to streamline the restaurant reservation process. The frontend enables users to search for restaurants, book tables, and manage reservations through intuitive forms and interactive dashboards. The backend enforces business rules, validations, and data consistency, such as preventing double bookings and ensuring accurate table availability. RESTful APIs connect the frontend and backend, providing real-time updates and smooth user experiences. This project demonstrates how modern

frameworks like Spring Boot and React can be effectively combined to develop a dependable and user-friendly reservation application.

1.3 Objectives

- To implement restaurant and user management, including creation, retrieval, and validation of restaurant profiles and user accounts.
- To enable reservation processing (creation, modification, cancellation) with data consistency.
- To provide users and restaurant owners with a clear and organized reservation history.
- To ensure validation, error handling, and real-time feedback for all reservation operations.

CHAPTER 2
SYSTEM SPECIFICATION

In this chapter, we describe the software tools and technologies used in building the Restaurant Table Reservation System. These tools play a crucial role in ensuring that the project is reliable, efficient, and user-friendly. Each technology has been carefully selected to support both the frontend and backend of the system, as well as database management and testing.

2.1 VISUAL STUDIO CODE (VS Code)

Visual Studio Code is the primary code editor used in the development of this project. It is a lightweight yet powerful source code editor developed by Microsoft, supporting Windows, Linux, and macOS. VS Code offers built-in features such as debugging, Git version control, intelligent code completion, syntax highlighting, and a vast extension marketplace, all of which contribute to an efficient and productive development workflow.

For the Restaurant Table Reservation System, VS Code was particularly valuable for both frontend (React.js) and backend (Spring Boot) development. It provides excellent support for JavaScript, JSX, TypeScript, and Java—the main languages used in this system. The integrated terminal enabled smooth execution of Maven commands for backend builds and npm commands for frontend builds without leaving the editor. Additionally, a wide range of extensions, including Spring Boot tools, React developer tools, and database connectors, further streamlined the development process and simplified debugging tasks.

2.2 LOCAL STORAGE

Local Storage is a client-side web storage mechanism that enables websites and web applications to store data directly within the browser. Compared to cookies, Local Storage can store a much larger amount of data (up to 5MB per domain), and the information is not transmitted to the server with every request. This makes it an efficient and secure solution for handling non-sensitive user-specific data.

In the Restaurant Table Reservation System, Local Storage plays a key role in maintaining temporary session data and enhancing application performance. For example, reservation details or restaurant search results fetched from the backend can be cached in Local Storage, allowing the data to remain available as users navigate between different pages. This reduces the number of API calls to the backend, optimizes response times, and ensures a smoother user experience.

Persistence is another advantage of Local Storage. Unlike session storage, which clears data when the browser is closed, Local Storage preserves information until it is explicitly removed by the user or the application. This means that users can continue their reservation process or view recently searched restaurants even after closing and reopening the browser.

From a security standpoint, data stored in Local Storage is accessible only within the browser and is not automatically sent to the server. Sensitive information such as passwords or payment details is not stored in Local Storage; instead, it is used for less critical data like user preferences, search filters, or cached reservation history. This approach reduces the need for repeated server requests and minimizes unnecessary network usage.

Local Storage also contributes to the reliability and performance of the system. Since the data is stored locally on the user's device, network delays or temporary backend outages do not prevent the application from functioning for certain features. For example, users can still view their last known reservation history or restaurant preferences even if the backend is temporarily unavailable, which improves the overall robustness of the system.

In summary, Local Storage in this project acts as a fast, reliable, and persistent storage solution on the client side. It improves the efficiency of the Restaurant Table Reservation System by caching frequently used data, reducing dependency on the server, enhancing user experience, and ensuring that non-sensitive information is instantly available whenever needed.

CHAPTER 3
PROPOSED SYSTEM

This chapter gives a brief description of the proposed idea behind the development of our website.

3.1 Proposed System

The proposed Restaurant Table Reservation System is a streamlined web-based application designed to provide users with an efficient and user-friendly way to book tables at restaurants. It addresses the challenges of traditional manual reservation methods by offering a digital platform that ensures speed, accuracy, and convenience for both customers and restaurant owners.

The system is built with a React.js frontend, a Spring Boot backend, and a MySQL database. The frontend enables customers to search for restaurants, make and manage reservations, and view their booking history through an intuitive dashboard. Restaurant owners can manage restaurant profiles, update table availability, and handle reservation requests in real time. The backend enforces business logic, performs data validation, and maintains consistency across all operations. MySQL serves as the secure and reliable store for restaurant, user, and reservation data, supporting fast retrieval and robust reporting.

The proposed solution follows a modular architecture:

- User and Restaurant Management Module – Allows creation and management of user and restaurant accounts with proper validations such as unique identifiers, contact information, and role assignments.
- Reservation Module – Supports creating, modifying, and canceling reservations, with strict validation rules to prevent double bookings and ensure table capacity.
- Reservation History Module – Maintains a comprehensive log of reservations for every user and restaurant, with sorting and filtering options for enhanced usability.

- Error Handling and Validation – Provides clear feedback and meaningful error messages for situations such as unavailable tables, invalid reservation requests, or scheduling conflicts.
- Data Persistence – Implemented using MySQL, ensuring reliable and permanent storage of reservation and user records.
- Performance Optimization – Achieved by leveraging browser Local Storage to cache frequently accessed data, reduce unnecessary API calls, and improve application speed.

By combining modern web technologies with a robust backend, the proposed system delivers a seamless restaurant reservation experience that is both scalable and reliable for users and restaurant owners alike.

3.2 Advantages of the Proposed System

The proposed Restaurant Table Reservation System offers several advantages over traditional reservation methods and existing manual processes:

- User-Friendly Interface – The React.js frontend presents an interactive and responsive dashboard, making it easy for users to search for restaurants and manage reservations.
- Accuracy and Data Consistency – Reservation operations are validated and processed using Spring Boot, ensuring correctness and preventing issues such as double bookings or scheduling conflicts.
- Faster Processing and Reduced Delays – Unlike manual reservation systems, the online platform allows instant booking, modification, and cancellation of reservations, enhancing customer satisfaction.
- Secure and Reliable Data Management – All user, restaurant, and reservation details are stored in MySQL, guaranteeing data integrity, security, and efficient access.
- Error Handling and Validations – The system prevents invalid actions (e.g., booking unavailable tables or overlapping reservations) and provides users with meaningful error messages.

- Improved Accessibility – Being web-based, the system can be accessed from any device with a modern browser, eliminating the need for phone calls or in-person bookings.
- Enhanced Transparency – The reservation history feature allows users and restaurant owners to monitor all reservation activities, increasing accountability and trust.
- Scalability and Extendibility – The modular architecture supports easy addition of future features such as online payment integration, dynamic seat management, loyalty programs, and mobile app support.
- Performance Optimization – Use of Local Storage enables quick access to cached data, reduces server dependency, and allows partial offline functionality for improved user experience.
- Cost-Effective and Time-Saving – The system significantly reduces manual effort, paperwork, and administrative overhead, making it efficient for both customers and restaurant management.

By adopting this proposed system, restaurants and customers benefit from a modern, transparent, and efficient reservation process that enhances overall service quality and operational effectiveness.

CHAPTER 4
METHODOLOGIES

This chapter describes the system design, workflows, and technical methodologies followed in developing the Restaurant Table Reservation System. It includes workflow descriptions, UML diagrams (Use Case, Class, Sequence, and Activity), and technical implementation details that define how different components of the system interact to ensure smooth and reliable operations.

4.1 System Workflow

The Restaurant Table Reservation System is designed to handle user management, restaurant management, and reservation processing with data consistency, validation, and efficient operations. The workflow is structured as follows:

1. **User Authentication & Registration**
   - Users interact with the React frontend, which communicates with the Spring Boot backend for login and registration validation.
   - Valid credentials grant access to either user or restaurant owner dashboards; invalid attempts trigger error messages.
   - While the current version uses basic authentication logic, it can be extended with features like JWT-based security or OAuth in the future.

2. **Restaurant & User Management**
   - Restaurant owners can register and manage their restaurant profiles by providing details such as restaurant name, location, contact information, and table layout/capacity.
   - Users can register new accounts by entering personal details and email, secured with validations for unique usernames and emails.
   - Both users and owners can view and update their profiles as needed.

3. **Reservation Processing**
   - Reservation Creation: Users search for restaurants based on location, cuisine, or name. After selecting a restaurant, they choose the date, time, and party size, and submit a reservation request.
     - The backend validates table availability for the requested slot, prevents overbooking, and ensures input correctness.
     - If a suitable table is available, the reservation is saved and both user and owner are notified.
     - If no table is available, the user receives a clear error message.
   - Reservation Modification/Cancellation: Users can view their existing reservations and modify or cancel them if needed.
     - The system checks for valid modification windows and updates availability accordingly.
   - Owner Management: Restaurant owners can view, accept, or decline reservation requests and update table availability in real time.
   - Data Integrity: All reservation operations are atomic—either all relevant changes are applied, or none are, to prevent inconsistencies in case of failure.

The following sections of this chapter will include detailed UML diagrams and technical explanations on how the system's components interact to provide a smooth reservation experience for users and restaurant owners.

4.2 Diagrams

4.2.1 Use Case Diagram

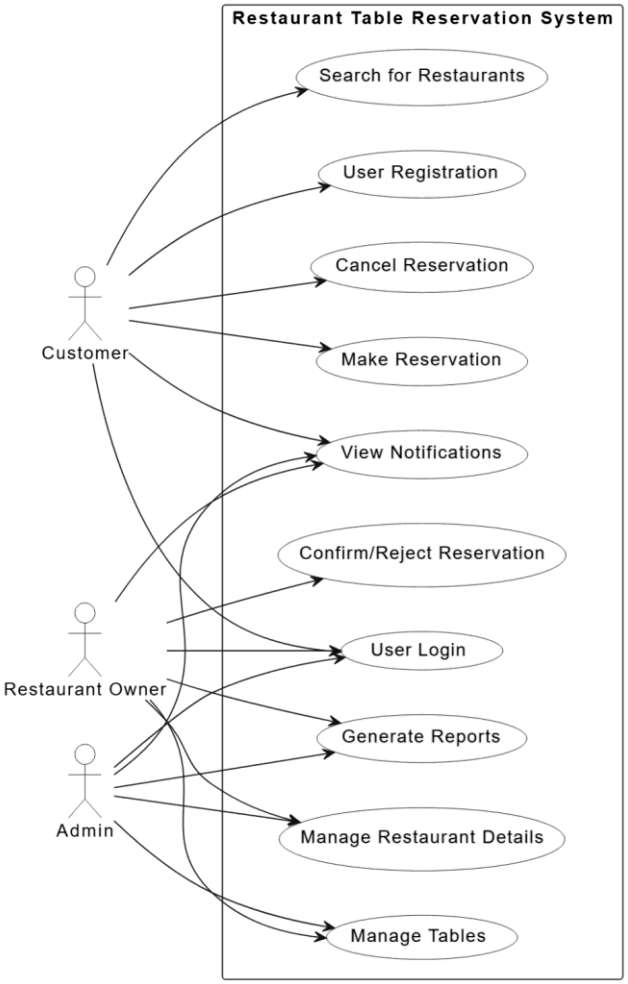


Fig No.1 Use Case Diagram

Description:

- Represents the interactions between users (actors) and the system.
- Identifies primary actors: Customer, Restaurant Owner, Admin.
- Shows main system functionalities such as:
  - Registration and Login
  - Searching for restaurants
  - Making, confirming, canceling, and rejecting reservations
  - Managing restaurant and table information
  - Viewing notifications and reports
- Visualizes which actor can perform which use case.
- Helps gather and communicate system requirements.

4.2.2 Class Diagram

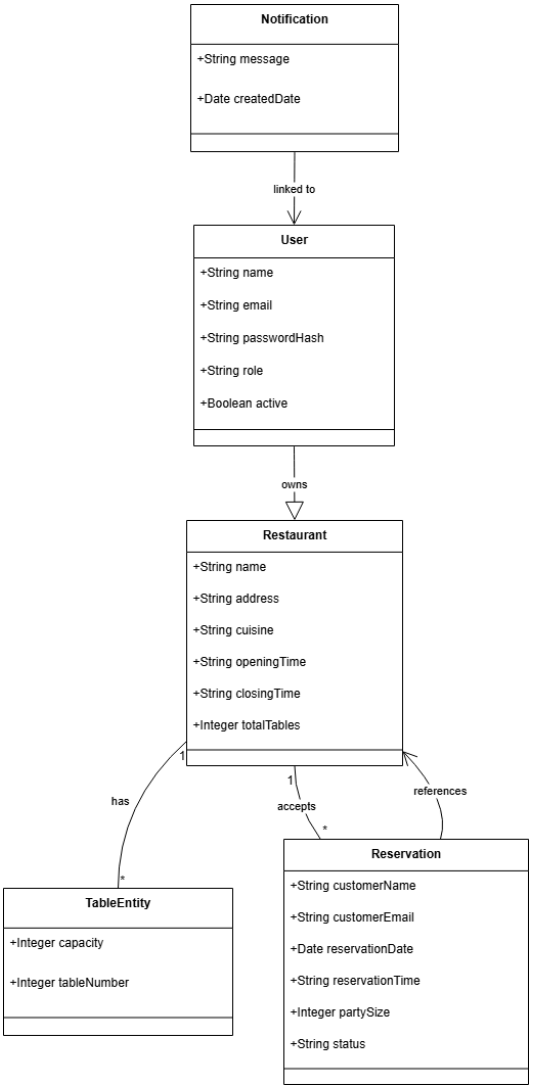


Fig No.2 Class Diagram

Key Classes:

User: Represents system users (Admin, Owner, Customer).

Restaurant: Restaurant details, owned by a User.

TableEntity: Tables in a restaurant.

Reservation: Customer reservation for a restaurant.

Notification: Messages sent to users.

4.2.3 Sequence Diagram

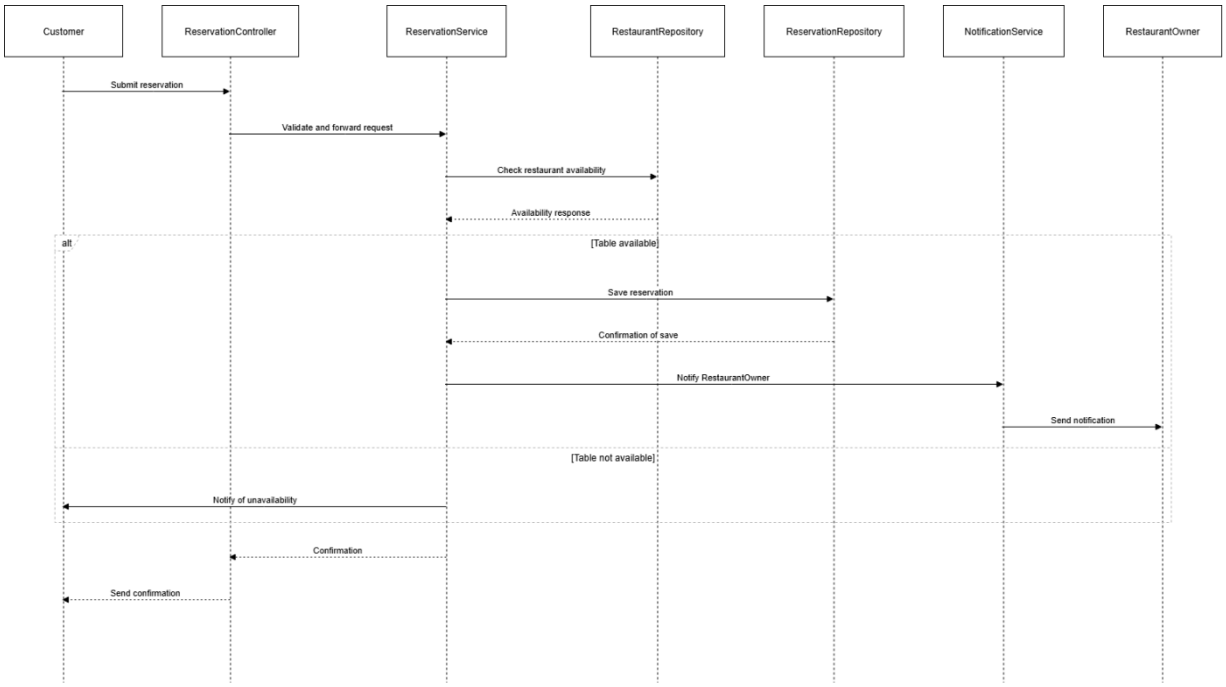


Fig No.3 Sequence Diagram

Process Flow:

- Customer sends reservation request.
- Controller receives and validates request.
- Controller calls ReservationService.
- Service checks restaurant and table availability.
- If available, saves reservation and notifies owner.
- Service returns result to Controller.
- Controller responds to Customer.

4.2.4 Activity Diagram

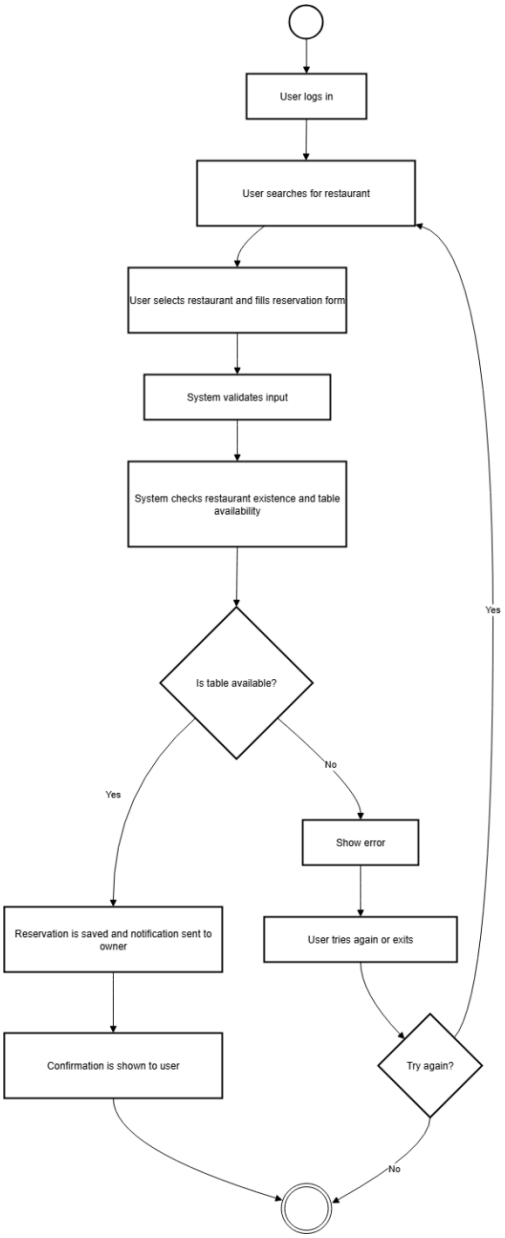


Fig No.4 Activity Diagram

Steps:

1. User logs in.
2. User searches for a restaurant.
3. User selects a restaurant.
4. User fills in the reservation form.
5. System validates the input.
6. System checks if the restaurant exists.
7. System checks table availability for the selected date.

8. [Decision] Are tables available?
    - If **No**: Show "Capacity Full" error to the user.
    - If **Yes**: Proceed to next step.
9. System saves the reservation.
10. System sends a notification to the restaurant owner.
11. System shows confirmation to the user.
12. End.

4.3 Technical Implementation

1. **Backend (Spring Boot)**
    a. Implements REST APIs for User, Restaurant, and Reservation management.
    b. Endpoints:
        i. `POST /api/users` → Register a new user
        ii. `POST /api/restaurants` → Register a new restaurant
        iii. `GET /api/restaurants` → Retrieve list of restaurants
        iv. `GET /api/restaurants/{id}` → Retrieve restaurant details
        v. `POST /api/reservations` → Create a reservation
        vi. `GET /api/reservations/user/{userId}` → Get reservations for a user
        vii. `GET /api/reservations/restaurant/{restaurantId}` → Get reservations for a restaurant
        viii. `PUT /api/reservations/{id}` → Modify a reservation
        ix. `DELETE /api/reservations/{id}` → Cancel a reservation
    c. Business rules: Prevent double bookings, validate table capacity, enforce reservation time windows, ensure only authorized users can manage reservations or restaurant details.
2. **Frontend (React.js)**
    a. Components:
        i. Restaurant Search Component - Allows users to search and filter restaurants
        ii. Reservation Form Component - Handles reservation creation and modification

iii. Reservations Dashboard - Displays user or restaurant owner reservation lists

iv. Restaurant Profile Component - Shows restaurant details, table availability, and management options

b. API Integration: Axios is used for calling REST APIs with robust success/error handling and user-friendly messages.

c. Local Storage: Caches search results, user session data, and recent reservations for improved performance and experience.

3. Database (MySQL)

a. Stores persistent data for users, restaurants, tables, and reservations.

b. Relationships:

i. One-to-Many (One Restaurant → Many Reservations)

ii. One-to-Many (One User → Many Reservations)

iii. One Restaurant → Many Tables (if table-level booking is implemented)

c. Ensures data integrity, supports indexing for fast lookup (e.g., available tables), and provides transaction rollback for failed reservation operations.

CHAPTER 5
IMPLEMENTATION AND RESULTS

This chapter provides a description of the output produced through the development of our website based on the proposed idea.

5.1 REGISTER PAGE

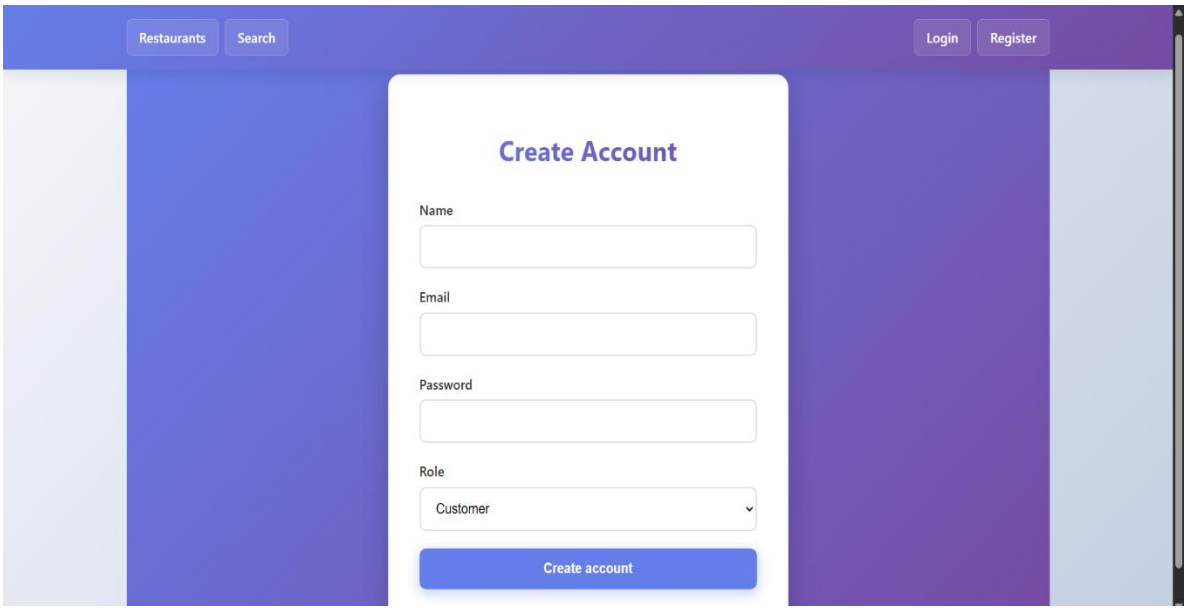


Fig No.5 Register Page

5.2 LOGIN PAGE

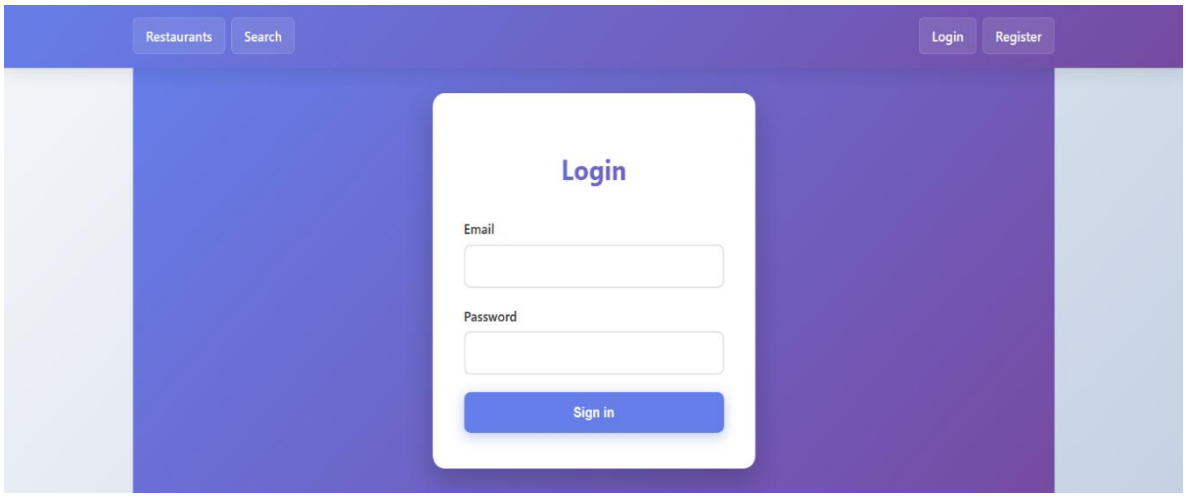


Fig No.6 Login Page

5.3 RESTAURANT LIST

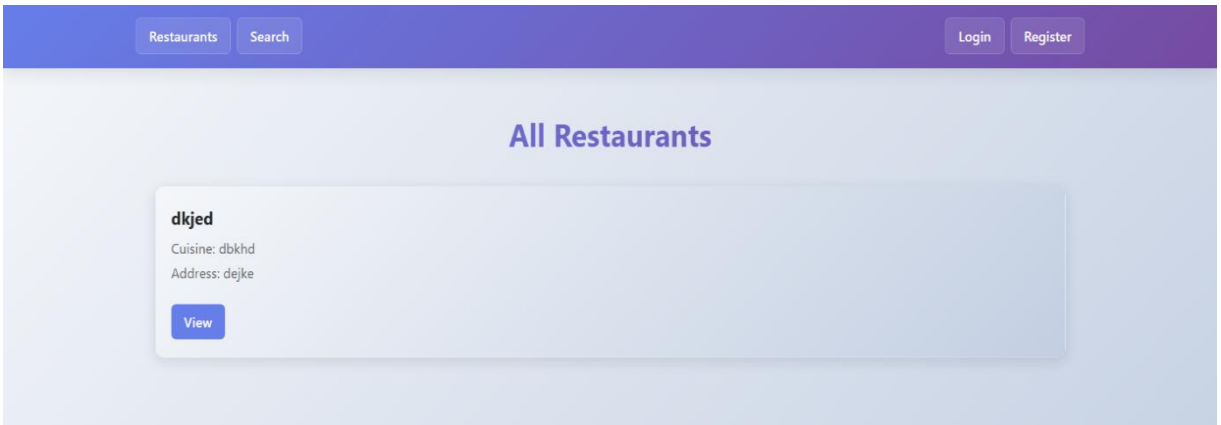


Fig No.7 Restaurant List

5.4 RESTAURANT DETAIL

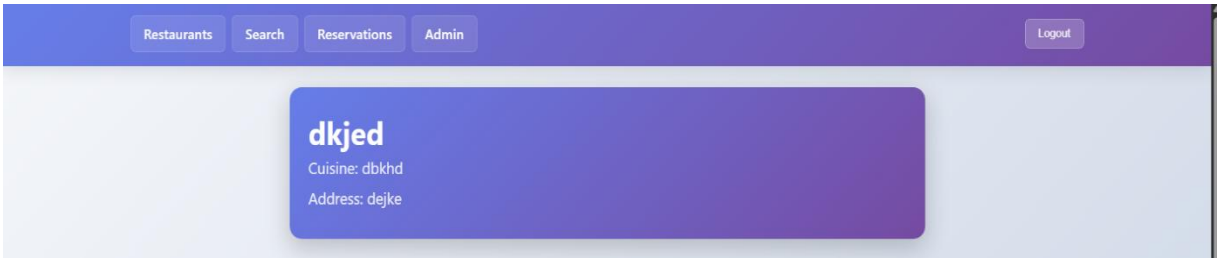
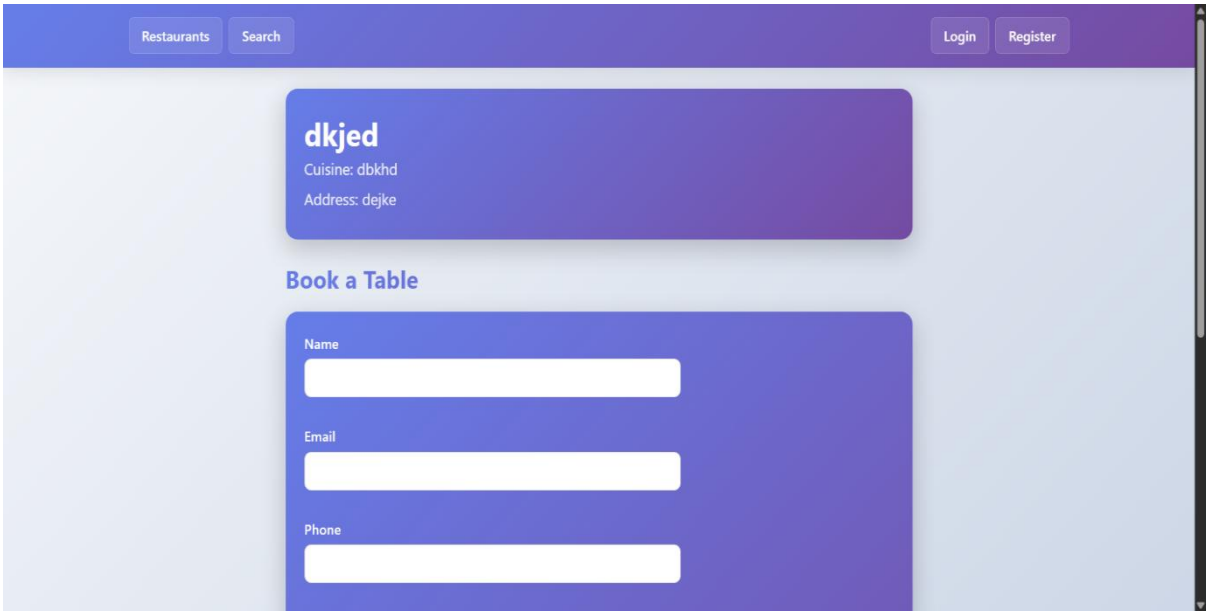


Fig No.8 Restaurant Detail

5.5 RESERVATION FORM



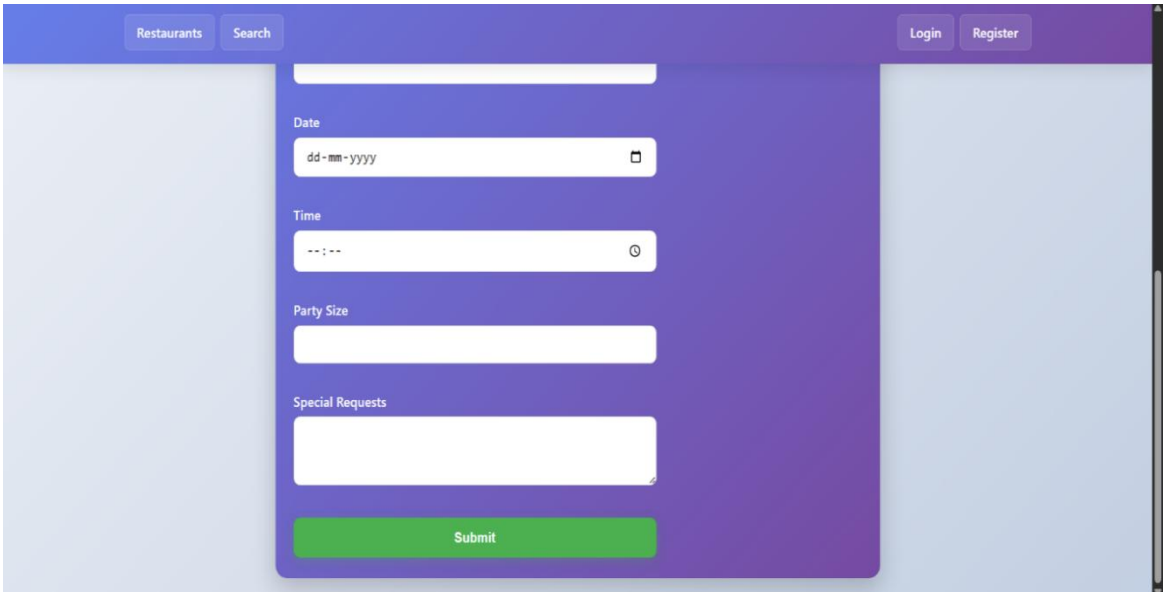


Fig No.9 Reservation Form

5.6 RESTAURANT SEARCH

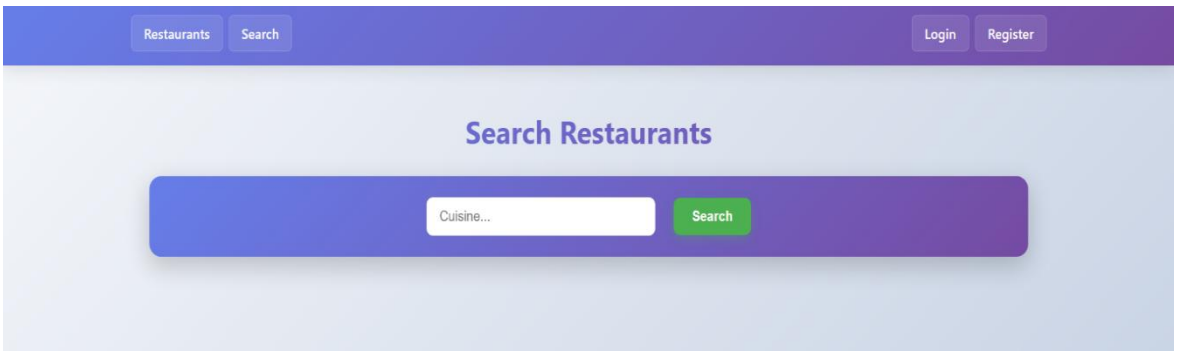
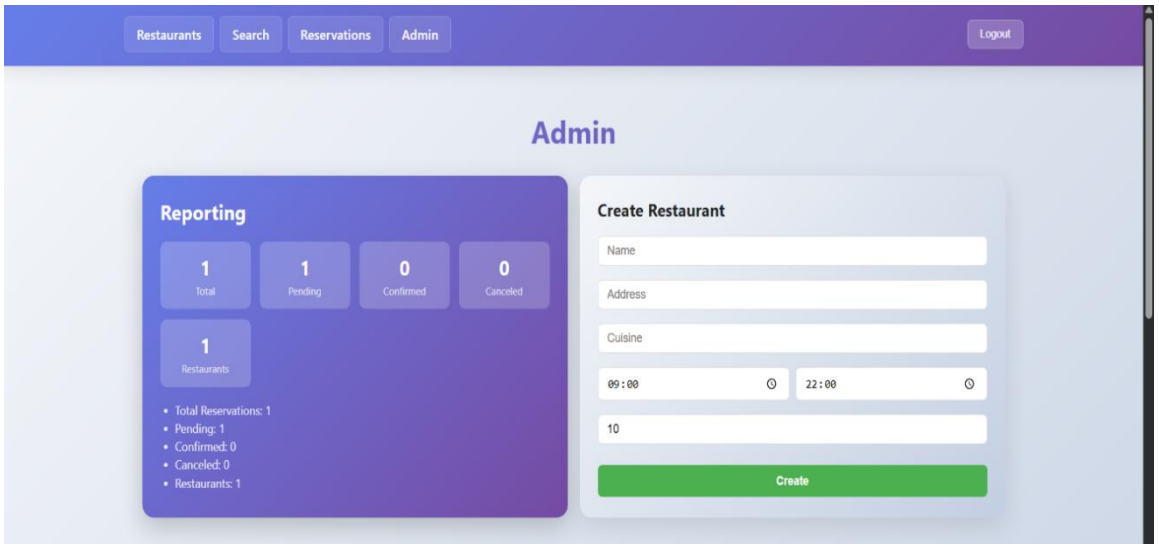


Fig No.10 Restaurant search

5.7 ADMIN DASHBOARD



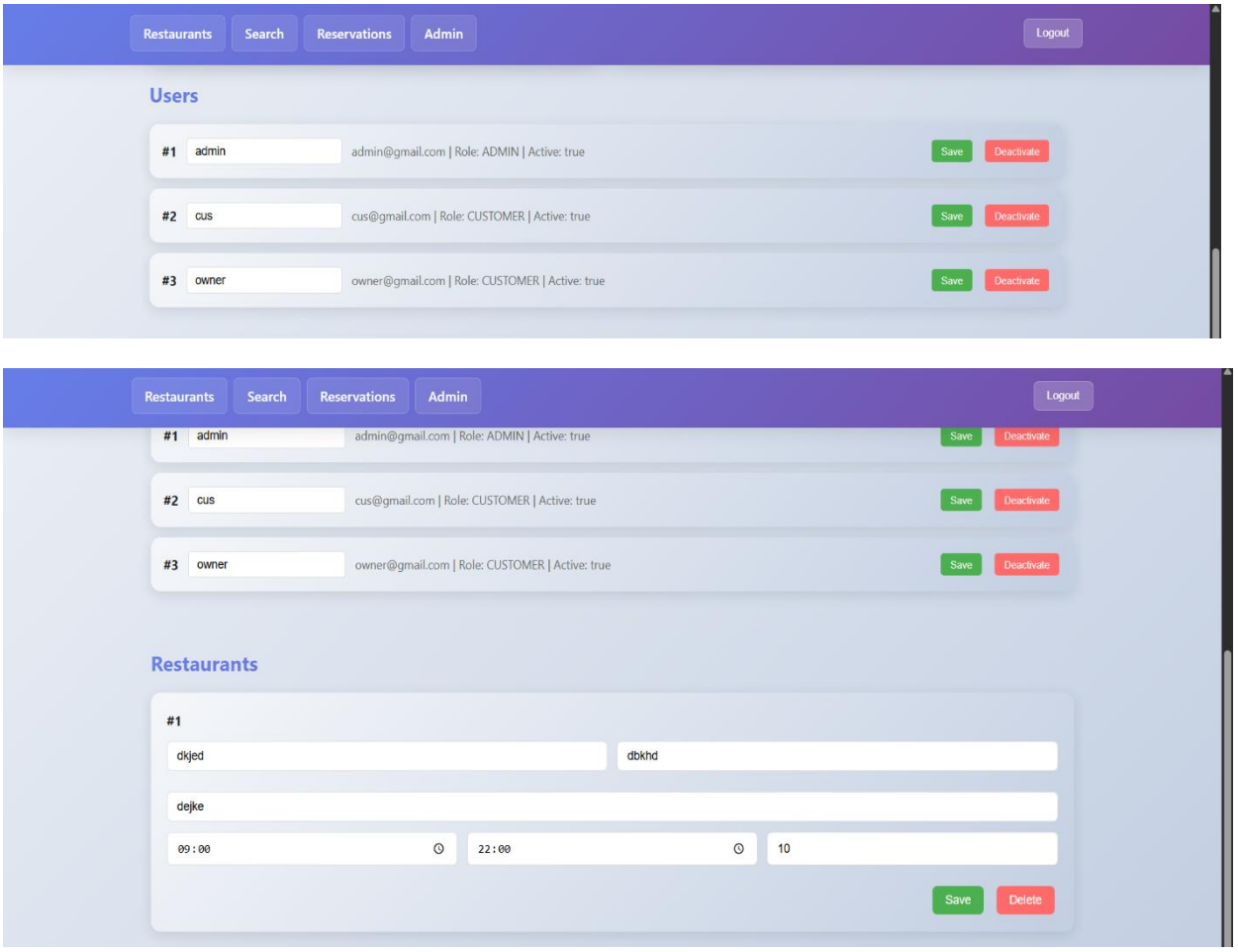


Fig No.11 Admin Dashboard

5.8 CODING

5.8.1 FRONTEND CODE

ReservationForm.jsx

```
import React, { useState } from 'react';
import ReservationService from '../utils/ReservationService';

function ReservationForm({ restaurant }) {
   const [name, setName] = useState('');
   const [email, setEmail] = useState('');
   const [phone, setPhone] = useState('');
   const [date, setDate] = useState('');
   const [time, setTime] = useState('');
   const [partySize, setPartySize] = useState('');
   const [specialRequests, setSpecialRequests] = useState('');
   const [error, setError] = useState('');
   const [success, setSuccess] = useState('');
```

```
const validate = () => {
  if (!name.trim()) {
    setError('Name is required.');
    return false;
  }
  if (!email.includes('@') || !email.includes('.')) {
    setError('Valid email is required.');
    return false;
  }
  if (!phone.trim()) {
    setError('Phone is required.');
    return false;
  }
  if (!date) {
    setError('Date is required.');
    return false;
  }
  if (!time) {
    setError('Time is required.');
    return false;
  }
  const size = parseInt(partySize, 10);
  if (Number.isNaN(size) || size < 1 || size > 20) {
    setError('Party size must be between 1 and 20.');
    return false;
  }
  setError('');
  return true;
};

const handleSubmit = async (e) => {
  e.preventDefault();
  setSuccess('');
  if (!validate()) return;

  const payload = {
    customerName: name.trim(),
    customerEmail: email.trim(),
    customerPhone: phone.trim(),
    reservationDate: date,
    reservationTime: time,
    partySize: parseInt(partySize, 10),
    specialRequests: specialRequests || '',
    restaurantId: restaurant?.id
  };

  try {
```

```
        await ReservationService.create(payload);
        setSuccess('Reservation request submitted!');
        setError('');
        setName('');
        setEmail('');
        setPhone('');
        setDate('');
        setTime('');
        setPartySize('');
        setSpecialRequests('');
    } catch (e2) {
        setError(e2?.message || 'Failed to submit reservation.');
    }
};

const containerStyle = {
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
    borderRadius: '15px',
    padding: '24px',
    boxShadow: '0 10px 30px rgba(0, 0, 0, 0.2)',
    marginTop: '8px',
    color: 'white'
};

const inputStyle = {
    width: '100%',
    padding: '12px 16px',
    borderRadius: '8px',
    border: '1px solid #e0e0e0',
    fontSize: '16px',
    marginBottom: '16px',
    backgroundColor: 'white',
    color: '#333',
    transition: 'border-color 0.3s ease'
};

const buttonStyle = {
    width: '100%',
    padding: '14px',
    backgroundColor: '#4CAF50',
    color: 'white',
    border: 'none',
    borderRadius: '8px',
    fontSize: '16px',
    fontWeight: 'bold',
    cursor: 'pointer',
    transition: 'all 0.3s ease',
```

```
    boxShadow: '0 4px 12px rgba(76, 175, 80, 0.3)'
  };

  const labelStyle = {
    display: 'block',
    marginBottom: '6px',
    fontWeight: '500',
    color: 'white'
  };

  return (
    <div style={containerStyle}>
      <form onSubmit={handleSubmit}>
        {error && (
          <div data-testid="error-message" style={{
            color: '#ff6b6b',
            marginBottom: '16px',
            padding: '12px',
            backgroundColor: 'rgba(255, 255, 255, 0.9)',
            borderRadius: '8px',
            fontWeight: '500'
          }}>
            {error}
          </div>
        )}
        {success && (
          <div data-testid="success-message" style={{
            color: '#51cf66',
            marginBottom: '16px',
            padding: '12px',
            backgroundColor: 'rgba(255, 255, 255, 0.9)',
            borderRadius: '8px',
            fontWeight: '500'
          }}>
            {success}
          </div>
        )}

        <div style={{ display: 'grid', gap: '16px', maxWidth: '480px' }}>
          <div>
            <label style={labelStyle}>Name</label>
            <input
              type="text"
              value={name}
              onChange={e => setName(e.target.value)}
              style={inputStyle}
```

```
        data-testid="name-input"
      />
    </div>
    <div>
      <label style={labelStyle}>Email</label>
      <input
        type="email"
        value={email}
        onChange={e => setEmail(e.target.value)}
        style={inputStyle}
        data-testid="email-input"
      />
    </div>
    <div>
      <label style={labelStyle}>Phone</label>
      <input
        type="text"
        value={phone}
        onChange={e => setPhone(e.target.value)}
        style={inputStyle}
        data-testid="phone-input"
      />
    </div>
    <div>
      <label style={labelStyle}>Date</label>
      <input
        type="date"
        value={date}
        onChange={e => setDate(e.target.value)}
        style={inputStyle}
        data-testid="date-input"
      />
    </div>
    <div>
      <label style={labelStyle}>Time</label>
      <input
        type="time"
        value={time}
        onChange={e => setTime(e.target.value)}
        style={inputStyle}
        data-testid="time-input"
      />
    </div>
    <div>
      <label style={labelStyle}>Party Size</label>
      <input
        type="number"
```

```
                min="1"
                max="20"
                value={partySize}
                onChange={e => setPartySize(e.target.value)}
                style={inputStyle}
                data-testid="party-size-input"
              />
            </div>
            <div>
              <label style={labelStyle}>Special Requests</label>
              <textarea
                value={specialRequests}
                onChange={e => setSpecialRequests(e.target.value)}
                style={{...inputStyle, minHeight: '80px', resize: 'vertical'}}
                data-testid="special-requests-input"
              />
            </div>
            <button
              type="submit"
              style={buttonStyle}
              data-testid="submit-button"
              onMouseOver={e => e.target.style.backgroundColor = '#45a049'}
              onMouseOut={e => e.target.style.backgroundColor = '#4CAF50'}
            >
              Submit
            </button>
          </div>
        </form>
      </div>
    );
}

export default ReservationForm;
```

ReservationList.jsx

```
import React, { useEffect, useState } from 'react';
import ReservationService from '../utils/ReservationService';
import ReservationStatus from './ReservationStatus';

function ReservationList() {
  const [reservations, setReservations] = useState([]);
  const [loaded, setLoaded] = useState(false);
  const [err, setErr] = useState('');

  useEffect(() => {
    let mounted = true;
```

```
    (async () => {
      try {
        const data = await ReservationService.getAll();
        if (mounted) setReservations(data || []);
      } catch (e) {
        if (mounted) setErr('Failed to load reservations');
      } finally {
        if (mounted) setLoaded(true);
      }
    })();
    return () => { mounted = false; };
  }, []);

  const handleCancel = async (id) => {
    try {
      await ReservationService.cancel(id);
      setReservations(prev => prev.filter(r => r.id !== id));
    } catch (e) {}
  };

  const handleStatusUpdate = async (id, newStatus) => {
    try {
      await ReservationService.updateStatus(id, newStatus);
      setReservations(prev => prev.map(r => r.id === id ? { ...r, status: newStatus }
: r));
    } catch (e) {}
  };

  if (!loaded) {
    return <div>Loading...</div>;
  }
  if (err) {
    return <div style={{ color: 'red' }}>{err}</div>;
  }

  const titleStyle = {
    fontSize: '32px',
    fontWeight: 'bold',
    color: '#2c3e50',
    marginBottom: '24px',
    textAlign: 'center',
    background: 'linear-gradient(45deg, #667eea, #764ba2)',
    WebkitBackgroundClip: 'text',
    WebkitTextFillColor: 'transparent',
    backgroundClip: 'text'
  };
```

```
const listStyle = {
   listStyle: 'none',
   padding: 0,
   margin: 0
};

const itemStyle = {
   background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
   borderRadius: '12px',
   padding: '20px',
   marginBottom: '16px',
   boxShadow: '0 4px 15px rgba(0, 0, 0, 0.1)',
   border: '1px solid rgba(255, 255, 255, 0.2)',
   transition: 'transform 0.3s ease, box-shadow 0.3s ease'
};

const buttonStyle = {
   padding: '8px 16px',
   margin: '0 4px',
   borderRadius: '6px',
   border: 'none',
   fontWeight: '500',
   cursor: 'pointer',
   transition: 'all 0.3s ease'
};

const cancelButtonStyle = {
   ...buttonStyle,
   backgroundColor: '#ff6b6b',
   color: 'white'
};

return (
   <div>
      <h1 style={titleStyle}>All Reservations</h1>
      {reservations.length === 0 ? (
         <div data-testid="empty" style={{
            textAlign: 'center',
            padding: '40px',
            color: '#666',
            fontSize: '18px'
         }}>
            No reservations found.
         </div>
      ) : (
         <ul style={listStyle}>
            {reservations.map(res => (
```

```
                <li
                    key={res.id}
                    data-testid={`reservation-item-${res.id}`}
                    style={itemStyle}
                    onMouseOver={e => {
                        e.currentTarget.style.transform = 'translateY(-2px)';
                        e.currentTarget.style.boxShadow = '0 8px 25px rgba(0, 0, 0,
0.15)';
                    }}
                    onMouseOut={e => {
                        e.currentTarget.style.transform = 'translateY(0)';
                        e.currentTarget.style.boxShadow = '0 4px 15px rgba(0, 0, 0,
0.1)';
                    }}
                >
                    <div style={{ fontSize: '18px', fontWeight: 'bold', marginBottom:
'8px' }}>
                        <strong>{res.customerName}</strong> ({res.status})
                    </div>
                    <div style={{ color: '#666', marginBottom: '8px' }}>
                        Date: {res.reservationDate} Time: {res.reservationTime} Party:
{res.partySize}
                    </div>
                    <div style={{ color: '#666', marginBottom: '12px' }}>
                        Restaurant ID: {res.restaurantId}
                    </div>
                    <div style={{ display: 'flex', gap: '8px', alignItems: 'center' }}>
                        <ReservationStatus
                            reservationId={res.id}
                            status={res.status}
                            onStatusUpdate={handleStatusUpdate}
                        />
                        <button
                            data-testid={`cancel-button-${res.id}`}
                            onClick={() => handleCancel(res.id)}
                            style={cancelButtonStyle}
                            onMouseOver={e => e.target.style.backgroundColor =
'#ff5252'}
                            onMouseOut={e => e.target.style.backgroundColor =
'#ff6b6b'}
                        >
                            Cancel
                        </button>
                    </div>
                </li>
            ))}
        </ul>
```

```
        )}
      </div>
    );
}

export default ReservationList;
```

ReservationStatus.jsx

```
import React from 'react';

function ReservationStatus({ reservationId, status, onStatusUpdate }) {
  const isConfirmed = status === 'CONFIRMED';

  const handleConfirm = () => {
    if (onStatusUpdate) {
      onStatusUpdate(reservationId, 'CONFIRMED');
    }
  };

  const buttonStyle = {
    padding: '8px 16px',
    borderRadius: '6px',
    border: 'none',
    fontWeight: '500',
    cursor: isConfirmed ? 'not-allowed' : 'pointer',
    transition: 'all 0.3s ease',
    backgroundColor: isConfirmed ? '#a5d6a7' : '#4CAF50',
    color: 'white',
    opacity: isConfirmed ? 0.6 : 1
  };

  return (
    <div>
      <button
        data-testid={`confirm-button-${reservationId}`}
        onClick={handleConfirm}
        disabled={isConfirmed}
        style={buttonStyle}
        onMouseOver={e => {
          if (!isConfirmed) {
            e.target.style.backgroundColor = '#45a049';
          }
        }}
        onMouseOut={e => {
          if (!isConfirmed) {
            e.target.style.backgroundColor = '#4CAF50';
```

```
                }
            }}
        >
            Confirm
        </button>
    </div>
  );
}

export default ReservationStatus;
```

RestaurantDetail.jsx

```
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import RestaurantService from '../utils/RestaurantService';
import ReservationForm from './ReservationForm';

function RestaurantDetail() {
  const { id } = useParams();
  const [restaurant, setRestaurant] = useState(null);
  const [error, setError] = useState('');

  useEffect(() => {
    let mounted = true;
    (async () => {
      try {
        const data = await RestaurantService.getById(id);
        if (mounted) {
          setRestaurant(data);
          setError('');
        }
      } catch (e) {
        if (mounted) setError('Failed to fetch restaurant');
      }
    })();
    return () => { mounted = false; };
  }, [id]);

  if (error) {
    return <div style={{ color: 'red' }}>{error}</div>;
  }
  if (!restaurant) {
    return <div>Loading...</div>;
  }
```

```
const containerStyle = {
   maxWidth: '800px',
   margin: '0 auto'
};

const headerStyle = {
   background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
   borderRadius: '15px',
   padding: '24px',
   color: 'white',
   marginBottom: '24px',
   boxShadow: '0 10px 30px rgba(0, 0, 0, 0.2)'
};

const titleStyle = {
   fontSize: '36px',
   fontWeight: 'bold',
   marginBottom: '16px',
   margin: 0
};

const infoStyle = {
   fontSize: '18px',
   marginBottom: '8px',
   opacity: 0.9
};

const sectionTitleStyle = {
   fontSize: '28px',
   fontWeight: 'bold',
   color: '#2c3e50',
   marginBottom: '16px',
   background: 'linear-gradient(45deg, #667eea, #764ba2)',
   WebkitBackgroundClip: 'text',
   WebkitTextFillColor: 'transparent',
   backgroundClip: 'text'
};

return (
   <div style={containerStyle}>
      <div style={headerStyle}>
         <h1 style={titleStyle}>{restaurant.name}</h1>
         <div style={infoStyle}>Cuisine: {restaurant.cuisine}</div>
         <div style={infoStyle}>Address: {restaurant.address}</div>
      </div>
```

```
        <h2 style={sectionTitleStyle}>Book a Table</h2>
        <ReservationForm restaurant={restaurant} />
      </div>
    );
}

export default RestaurantDetail;
```

RestaurantList.jsx

```
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import RestaurantService from '../utils/RestaurantService';

function RestaurantList() {
  const [restaurants, setRestaurants] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    let mounted = true;
    (async () => {
      try {
        const data = await RestaurantService.getAll();
        if (mounted) {
          setRestaurants(data || []);
          setError('');
        }
      } catch (e) {
        if (mounted) {
          setError('Failed to fetch restaurants');
        }
      } finally {
        if (mounted) setLoading(false);
      }
    })();
    return () => { mounted = false; };
  }, []);

  if (loading) {
    return (
      <div data-testid="loading" style={{
        textAlign: 'center',
        padding: '40px',
        fontSize: '18px',
        color: '#666'
```

```
        }}>
            Loading...
        </div>
    );
}

if (error) {
    return (
        <div data-testid="error" style={{
            color: 'red',
            textAlign: 'center',
            padding: '40px',
            fontSize: '18px'
        }}>
            {error}
        </div>
    );
}

const titleStyle = {
    fontSize: '36px',
    fontWeight: 'bold',
    color: '#2c3e50',
    marginBottom: '32px',
    textAlign: 'center',
    background: 'linear-gradient(45deg, #667eea, #764ba2)',
    WebkitBackgroundClip: 'text',
    WebkitTextFillColor: 'transparent',
    backgroundClip: 'text'
};

const listStyle = {
    listStyle: 'none',
    padding: 0,
    margin: 0,
    display: 'grid',
    gap: '16px'
};

const itemStyle = {
    background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
    borderRadius: '12px',
    padding: '20px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.1)',
    transition: 'all 0.3s ease',
    border: '1px solid rgba(255, 255, 255, 0.2)'
};
```

```
const linkStyle = {
    display: 'inline-block',
    marginTop: '12px',
    padding: '8px 16px',
    backgroundColor: '#667eea',
    color: 'white',
    textDecoration: 'none',
    borderRadius: '6px',
    fontWeight: '500',
    transition: 'background-color 0.3s ease'
};

return (
    <div>
        <h1 style={titleStyle}>All Restaurants</h1>
        {restaurants.length === 0 ? (
            <p style={{ textAlign: 'center', color: '#666', fontSize: '18px' }}>
                No restaurants available.
            </p>
        ) : (
            <ul style={listStyle}>
                {restaurants.map(r => (
                    <li
                        key={r.id}
                        style={itemStyle}
                        onMouseOver={e => {
                            e.currentTarget.style.transform = 'translateY(-2px)';
                            e.currentTarget.style.boxShadow = '0 8px 25px rgba(0, 0, 0,
0.15)';
                        }}
                        onMouseOut={e => {
                            e.currentTarget.style.transform = 'translateY(0)';
                            e.currentTarget.style.boxShadow = '0 4px 15px rgba(0, 0, 0,
0.1)';
                        }}
                    >
                        <div style={{ fontSize: '20px', fontWeight: 'bold', marginBottom:
'8px' }}>
                            <strong>{r.name}</strong>
                        </div>
                        <div style={{ color: '#666', marginBottom: '4px' }}>
                            Cuisine: {r.cuisine}
                        </div>
                        <div style={{ color: '#666', marginBottom: '12px' }}>
                            Address: {r.address}
                        </div>
                        <Link
```

```
                        to={`/restaurants/${r.id}`}
                        style={linkStyle}
                        onMouseOver={e => e.target.style.backgroundColor =
'#5a6fd8'}
                        onMouseOut={e => e.target.style.backgroundColor = '#667eea'}
                      >
                        View
                      </Link>
                    </li>
                  ))}
                </ul>
              )}
            </div>
          );
}

export default RestaurantList;
```

RestaurantSearch.jsx

```
import React, { useState } from 'react';
import RestaurantService from '../utils/RestaurantService';

function RestaurantSearch() {
  const [query, setQuery] = useState('');
  const [results, setResults] = useState([]);
  const [searched, setSearched] = useState(false);
  const [error, setError] = useState('');

  const handleSearch = async () => {
    setError('');
    setSearched(true);
    try {
      const data = await RestaurantService.searchByCuisine(query);
      setResults(data || []);
    } catch (e) {
      setError('Failed to search');
      setResults([]);
    }
  };

  const titleStyle = {
    fontSize: '36px',
    fontWeight: 'bold',
    color: '#2c3e50',
    marginBottom: '24px',
    textAlign: 'center',
```

```
    background: 'linear-gradient(45deg, #667eea, #764ba2)',
    WebkitBackgroundClip: 'text',
    WebkitTextFillColor: 'transparent',
    backgroundClip: 'text'
};

const searchContainerStyle = {
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
    borderRadius: '15px',
    padding: '24px',
    marginBottom: '24px',
    boxShadow: '0 10px 30px rgba(0, 0, 0, 0.2)'
};

const inputStyle = {
    width: '300px',
    padding: '12px 16px',
    borderRadius: '8px',
    border: '1px solid #e0e0e0',
    fontSize: '16px',
    marginRight: '12px',
    backgroundColor: 'white'
};

const buttonStyle = {
    padding: '12px 24px',
    backgroundColor: '#4CAF50',
    color: 'white',
    border: 'none',
    borderRadius: '8px',
    fontSize: '16px',
    fontWeight: 'bold',
    cursor: 'pointer',
    transition: 'background-color 0.3s ease',
    boxShadow: '0 4px 12px rgba(76, 175, 80, 0.3)'
};

const listStyle = {
    listStyle: 'none',
    padding: 0,
    margin: 0,
    display: 'grid',
    gap: '16px'
};

const itemStyle = {
    background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
```

```
    borderRadius: '12px',
    padding: '20px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.1)',
    transition: 'transform 0.3s ease'
  };

  return (
    <div>
      <h1 style={titleStyle}>Search Restaurants</h1>
      <div style={searchContainerStyle}>
        <div style={{ display: 'flex', alignItems: 'center', justifyContent: 'center',
flexWrap: 'wrap', gap: '12px' }}>
          <input
            type="text"
            placeholder="Cuisine..."
            value={query}
            onChange={e => setQuery(e.target.value)}
            style={inputStyle}
            data-testid="search-input"
          />
          <button
            onClick={handleSearch}
            style={buttonStyle}
            data-testid="search-button"
            onMouseOver={e => e.target.style.backgroundColor = '#45a049'}
            onMouseOut={e => e.target.style.backgroundColor = '#4CAF50'}
          >
            Search
          </button>
        </div>
      </div>

      {error && <div style={{ color: 'red', textAlign: 'center', marginBottom: '16px'
}}>{error}</div>}

      {searched && results.length === 0 && (
        <div data-testid="no-results" style={{
          textAlign: 'center',
          padding: '40px',
          color: '#666',
          fontSize: '18px'
        }}>
          No matching restaurants found.
        </div>
      )}
```

```
        {results.length > 0 && (
          <ul style={listStyle}>
            {results.map(r => (
              <li
                key={r.id}
                style={itemStyle}
                onMouseOver={e => e.currentTarget.style.transform = 'translateY(-
2px)'}
                onMouseOut={e => e.currentTarget.style.transform =
'translateY(0)'}
              >
                <div style={{ fontSize: '20px', fontWeight: 'bold', marginBottom:
'8px' }}>
                  <strong>{r.name}</strong> - {r.cuisine}
                </div>
                <div style={{ color: '#666' }}>{r.address}</div>
              </li>
            ))}
          </ul>
        )}
      </div>
    );
}

export default RestaurantSearch;
```

AdminDashboard.jsx

```
import React, { useEffect, useState } from 'react';
import AdminService from '../utils/AdminService';
import UserService from '../utils/UserService';
import RestaurantService from '../utils/RestaurantService';

function AdminDashboard() {
  return (
    <div style={{ width: '100%' }}>
      <h1 style={{
        fontSize: '36px',
        fontWeight: 'bold',
        color: '#2c3e50',
        marginBottom: '24px',
        textAlign: 'center',
        background: 'linear-gradient(45deg, #667eea, #764ba2)',
        WebkitBackgroundClip: 'text',
        WebkitTextFillColor: 'transparent',
```

```
            backgroundClip: 'text'
         }}>
            Admin
         </h1>

         <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit,
minmax(300px, 1fr))', gap: '16px', marginBottom: '32px' }}>
            <AdminReport />
            <QuickCreateRestaurant />
         </div>

         <AdminUsers />
         <div style={{ margin: '32px 0', height: '1px', background: 'linear-
gradient(90deg, transparent, #ddd, transparent)' }}></div>
         <AdminRestaurants />
      </div>
   );
}

function AdminReport() {
   const [report, setReport] = useState(null);
   const [err, setErr] = useState('');

   useEffect(() => {
      (async () => {
         try {
            const data = await AdminService.getReport();
            setReport(data);
         } catch {
            setErr('Failed to load report');
         }
      })();
   }, []);

   if (err) return <div style={{ color: 'red' }}>{err}</div>;
   if (!report) return <div>Loading report...</div>;

   const containerStyle = {
      background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
      borderRadius: '15px',
      padding: '24px',
      color: 'white',
      height: '100%',
      boxShadow: '0 10px 30px rgba(0, 0, 0, 0.2)'
   };

   const statStyle = {
```

```
    display: 'grid',
    gridTemplateColumns: 'repeat(auto-fit, minmax(120px, 1fr))',
    gap: '12px',
    marginBottom: '16px'
  };

  const statCardStyle = {
    background: 'rgba(255, 255, 255, 0.2)',
    borderRadius: '8px',
    padding: '12px',
    textAlign: 'center'
  };

  return (
    <div style={containerStyle}>
      <h2 style={{ fontSize: '24px', marginBottom: '16px', margin: '0 0 16px 0'
}}>Reporting</h2>

      <div style={statStyle}>
        <div style={statCardStyle}>
          <div style={{ fontSize: '24px', fontWeight: 'bold'
}}>{report.totalReservations}</div>
          <div style={{ fontSize: '12px', opacity: 0.8 }}>Total</div>
        </div>
        <div style={statCardStyle}>
          <div style={{ fontSize: '24px', fontWeight: 'bold'
}}>{report.reservationsPending}</div>
          <div style={{ fontSize: '12px', opacity: 0.8 }}>Pending</div>
        </div>
        <div style={statCardStyle}>
          <div style={{ fontSize: '24px', fontWeight: 'bold'
}}>{report.reservationsConfirmed}</div>
          <div style={{ fontSize: '12px', opacity: 0.8 }}>Confirmed</div>
        </div>
        <div style={statCardStyle}>
          <div style={{ fontSize: '24px', fontWeight: 'bold'
}}>{report.reservationsCanceled}</div>
          <div style={{ fontSize: '12px', opacity: 0.8 }}>Canceled</div>
        </div>
        <div style={statCardStyle}>
          <div style={{ fontSize: '24px', fontWeight: 'bold'
}}>{report.restaurants}</div>
          <div style={{ fontSize: '12px', opacity: 0.8 }}>Restaurants</div>
        </div>
      </div>

      <ul style={{ margin: 0, paddingLeft: '20px', fontSize: '14px', opacity: 0.9 }}>
```

```
            <li>Total Reservations: {report.totalReservations}</li>
            <li>Pending: {report.reservationsPending}</li>
            <li>Confirmed: {report.reservationsConfirmed}</li>
            <li>Canceled: {report.reservationsCanceled}</li>
            <li>Restaurants: {report.restaurants}</li>
         </ul>
      </div>
   );
}

function QuickCreateRestaurant() {
   const [form, setForm] = useState({
      name: '',
      address: '',
      cuisine: '',
      openingTime: '09:00',
      closingTime: '22:00',
      totalTables: 10,
   });
   const [msg, setMsg] = useState('');

   const createRestaurant = async () => {
      try {
         await RestaurantService.createOwnerRestaurant({
            name: form.name,
            address: form.address,
            cuisine: form.cuisine,
            openingTime: form.openingTime,
            closingTime: form.closingTime,
            totalTables: Number(form.totalTables),
         });
         setMsg('Restaurant created.');
         setForm({
            name: '',
            address: '',
            cuisine: '',
            openingTime: '09:00',
            closingTime: '22:00',
            totalTables: 10,
         });
      } catch {
         setMsg('Failed to create restaurant.');
      }
   };

   const containerStyle = {
```

```
    background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
    borderRadius: '15px',
    padding: '24px',
    height: '100%',
    boxShadow: '0 10px 30px rgba(0, 0, 0, 0.1)'
  };

  const inputStyle = {
    width: '100%',
    padding: '8px 12px',
    borderRadius: '6px',
    border: '1px solid #ddd',
    fontSize: '14px',
    marginBottom: '8px',
    boxSizing: 'border-box'
  };

  const buttonStyle = {
    width: '100%',
    padding: '10px',
    backgroundColor: '#4CAF50',
    color: 'white',
    border: 'none',
    borderRadius: '6px',
    fontSize: '14px',
    fontWeight: 'bold',
    cursor: 'pointer',
    marginTop: '8px'
  };

  return (
    <div style={containerStyle}>
      <h2 style={{ fontSize: '20px', marginBottom: '16px', margin: '0 0 16px 0'
}}>Create Restaurant</h2>
      <div style={{ display: 'grid', gap: '8px' }}>
        <input
          placeholder="Name"
          value={form.name}
          onChange={(e) => setForm({ ...form, name: e.target.value })}
          style={inputStyle}
        />
        <input
          placeholder="Address"
          value={form.address}
          onChange={(e) => setForm({ ...form, address: e.target.value })}
          style={inputStyle}
        />
```

```
            <input
              placeholder="Cuisine"
              value={form.cuisine}
              onChange={(e) => setForm({ ...form, cuisine: e.target.value })}
              style={inputStyle}
            />
            <div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '8px' }}>
              <input
                type="time"
                value={form.openingTime}
                onChange={(e) => setForm({ ...form, openingTime: e.target.value })}
                style={inputStyle}
              />
              <input
                type="time"
                value={form.closingTime}
                onChange={(e) => setForm({ ...form, closingTime: e.target.value })}
                style={inputStyle}
              />
            </div>
            <input
              type="number"
              min="1"
              placeholder="Total Tables"
              value={form.totalTables}
              onChange={(e) => setForm({ ...form, totalTables: e.target.value })}
              style={inputStyle}
            />
            <button onClick={createRestaurant} style={buttonStyle}>
              Create
            </button>
          </div>
          {msg && <div style={{ color: 'green', marginTop: '8px', fontSize: '14px'
}}>{msg}</div>}
        </div>
    );
}

function AdminUsers() {
    const [users, setUsers] = useState([]);
    const [err, setErr] = useState('');
    const [msg, setMsg] = useState('');

    const load = async () => {
        try {
            const data = await UserService.getAll();
            setUsers(data || []);
```

```
    } catch {
        setErr('Failed to load users');
    }
};

useEffect(() => {
    load();
}, []);

const saveName = async (u) => {
    try {
        await UserService.updateName(u.id, u.name);
        setMsg('User updated.');
        await load();
    } catch {
        setMsg('Failed to update user.');
    }
};

const deactivate = async (id) => {
    try {
        await UserService.deactivate(id);
        setMsg('User deactivated.');
        await load();
    } catch {
        setMsg('Failed to deactivate user.');
    }
};

const titleStyle = {
    fontSize: '24px',
    fontWeight: 'bold',
    color: '#2c3e50',
    marginBottom: '16px',
    background: 'linear-gradient(45deg, #667eea, #764ba2)',
    WebkitBackgroundClip: 'text',
    WebkitTextFillColor: 'transparent',
    backgroundClip: 'text'
};

const userItemStyle = {
    background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
    borderRadius: '12px',
    padding: '16px',
    marginBottom: '12px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.1)',
    display: 'grid',
```

```
    gridTemplateColumns: 'auto 1fr auto auto',
    gap: '12px',
    alignItems: 'center'
  };

  const inputStyle = {
    padding: '6px 10px',
    borderRadius: '4px',
    border: '1px solid #ddd',
    fontSize: '14px'
  };

  const buttonStyle = {
    padding: '6px 12px',
    borderRadius: '4px',
    border: 'none',
    fontSize: '12px',
    fontWeight: '500',
    cursor: 'pointer',
    margin: '0 2px'
  };

  return (
    <section>
      <h2 style={titleStyle}>Users</h2>
      {err && <div style={{ color: 'red' }}>{err}</div>}

      {users.map((u) => (
        <div key={u.id} style={userItemStyle}>
          <strong>#{u.id}</strong>
          <div style={{ display: 'flex', alignItems: 'center', gap: '12px', flexWrap:
'wrap' }}>
            <input
              value={u.name}
              onChange={(e) =>
                setUsers((prev) =>
                  prev.map((x) => (x.id === u.id ? { ...x, name: e.target.value } :
x))
                )
              }
              style={{...inputStyle, minWidth: '150px'}}
            />
            <span style={{ color: '#666', fontSize: '14px' }}>
              {u.email} | Role: {u.role} | Active: {String(u.active)}
            </span>
          </div>
          <button
```

```
                onClick={() => saveName(u)}
                style={{...buttonStyle, backgroundColor: '#4CAF50', color: 'white'}}
            >
                Save
            </button>
            <button
                onClick={() => deactivate(u.id)}
                style={{...buttonStyle, backgroundColor: '#ff6b6b', color: 'white'}}
            >
                Deactivate
            </button>
        </div>
    ))}

    {msg && <div style={{ color: 'green', marginTop: '8px' }}>{msg}</div>}
  </section>
);
}

function AdminRestaurants() {
  const [restaurants, setRestaurants] = useState([]);
  const [err, setErr] = useState('');
  const [msg, setMsg] = useState('');

  const load = async () => {
    try {
      const data = await RestaurantService.getAll();
      setRestaurants(data || []);
    } catch {
      setErr('Failed to load restaurants');
    }
  };

  useEffect(() => {
    load();
  }, []);

  const updateRestaurant = async (r) => {
    try {
      await RestaurantService.updateOwnerRestaurant(r.id, {
        name: r.name,
        address: r.address,
        cuisine: r.cuisine,
        openingTime: r.openingTime,
        closingTime: r.closingTime,
        totalTables: Number(r.totalTables),
      });
```

```
        setMsg('Restaurant updated.');
        await load();
    } catch {
        setMsg('Failed to update restaurant.');
    }
};

const deleteRestaurant = async (id) => {
    try {
        const res = await fetch(
            `https://8080-
fabfbaefdabeacefdbbabadfdbbcbbebfbde.premiumproject.examly.io/api/restaurants/o
wner/${id}`,
            {
                method: 'DELETE',
                headers: {
                    'Content-Type': 'application/json',
                    ...(localStorage.getItem('token')
                        ? { Authorization: `Bearer ${localStorage.getItem('token')}` }
                        : {}),
                },
            }
        );
        if (!res.ok) throw new Error();
        setMsg('Restaurant deleted.');
        await load();
    } catch {
        setMsg('Failed to delete restaurant.');
    }
};

const titleStyle = {
    fontSize: '24px',
    fontWeight: 'bold',
    color: '#2c3e50',
    marginBottom: '16px',
    background: 'linear-gradient(45deg, #667eea, #764ba2)',
    WebkitBackgroundClip: 'text',
    WebkitTextFillColor: 'transparent',
    backgroundClip: 'text'
};

const restaurantItemStyle = {
    background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
    borderRadius: '12px',
    padding: '20px',
```

```
    marginBottom: '16px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.1)'
  };

  const inputStyle = {
    width: '100%',
    padding: '8px 12px',
    borderRadius: '6px',
    border: '1px solid #ddd',
    fontSize: '14px',
    marginBottom: '8px',
    boxSizing: 'border-box'
  };

  const buttonStyle = {
    padding: '8px 16px',
    borderRadius: '6px',
    border: 'none',
    fontSize: '14px',
    fontWeight: '500',
    cursor: 'pointer',
    margin: '0 4px'
  };

  return (
    <section>
      <h2 style={titleStyle}>Restaurants</h2>
      {err && <div style={{ color: 'red' }}>{err}</div>}

      {restaurants.map((r) => (
        <div key={r.id} style={restaurantItemStyle}>
          <div style={{ fontWeight: 'bold', marginBottom: '12px' }}>
            <strong>#{r.id}</strong>
          </div>

          <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit,
minmax(200px, 1fr))', gap: '12px', marginBottom: '16px' }}>
            <input
              placeholder="Name"
              value={r.name}
              onChange={(e) =>
                setRestaurants((prev) =>
                  prev.map((x) => (x.id === r.id ? { ...x, name: e.target.value } :
x))
                )
              }
              style={inputStyle}
```

```
                />
                <input
                  placeholder="Cuisine"
                  value={r.cuisine || ''}
                  onChange={(e) =>
                    setRestaurants((prev) =>
                      prev.map((x) => (x.id === r.id ? { ...x, cuisine: e.target.value }
: x))
                    )
                  }
                  style={inputStyle}
                />
              </div>

              <input
                placeholder="Address"
                value={r.address}
                onChange={(e) =>
                  setRestaurants((prev) =>
                    prev.map((x) => (x.id === r.id ? { ...x, address: e.target.value } :
x))
                  )
                }
                style={{...inputStyle, marginBottom: '12px'}}
              />

              <div style={{ display: 'grid', gridTemplateColumns: 'repeat(auto-fit,
minmax(120px, 1fr))', gap: '12px', marginBottom: '16px' }}>
                <input
                  type="time"
                  value={r.openingTime || ''}
                  onChange={(e) =>
                    setRestaurants((prev) =>
                      prev.map((x) => (x.id === r.id ? { ...x, openingTime:
e.target.value } : x))
                    )
                  }
                  style={inputStyle}
                />
                <input
                  type="time"
                  value={r.closingTime || ''}
                  onChange={(e) =>
                    setRestaurants((prev) =>
                      prev.map((x) => (x.id === r.id ? { ...x, closingTime:
e.target.value } : x))
```

```
                    )
                }
                style={inputStyle}
            />
            <input
                type="number"
                placeholder="Tables"
                value={r.totalTables || 0}
                onChange={(e) =>
                    setRestaurants((prev) =>
                        prev.map((x) =>
                            x.id === r.id ? { ...x, totalTables: Number(e.target.value) } :
x
                        )
                    )
                }
                style={inputStyle}
            />
        </div>

        <div style={{ display: 'flex', justifyContent: 'flex-end', gap: '8px' }}>
                        <button
                onClick={() => updateRestaurant(r)}
                style={{...buttonStyle, backgroundColor: '#4CAF50', color:
'white'}}
            >
                Save
            </button>
            <button
                onClick={() => deleteRestaurant(r.id)}
                style={{...buttonStyle, backgroundColor: '#ff6b6b', color: 'white'}}
            >
                Delete
            </button>
        </div>
    </div>
))}

    {msg && <div style={{ color: 'green', marginTop: '8px' }}>{msg}</div>}
  </section>
);
}

export default AdminDashboard;
```

App.js

```
import React from 'react';
import { Routes, Route, Link, Navigate } from 'react-router-dom';
import RestaurantList from './components/RestaurantList';
import RestaurantDetail from './components/RestaurantDetail';
import RestaurantSearch from './components/RestaurantSearch';
import ReservationList from './components/ReservationList';
import Login from './pages/Login';
import Register from './pages/Register';
import AdminDashboard from './pages/AdminDashboard';
import OwnerDashboard from './pages/OwnerDashboard';
import RequireAuth from './routes/RequireAuth';
import { useAuth } from './context/AuthContext';

function Nav() {
  const { isAuthenticated, role, logout } = useAuth();

  const navStyle = {
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
    padding: '16px 24px',
    boxShadow: '0 4px 20px rgba(0, 0, 0, 0.1)',
    position: 'sticky',
    top: 0,
    zIndex: 1000
  };

  const navContainerStyle = {
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center',



    maxWidth: '1200px',
    margin: '0 auto',
    flexWrap: 'wrap',
    gap: '12px'
  };

  const linkStyle = {
    color: 'white',
    textDecoration: 'none',
    padding: '8px 16px',
    borderRadius: '6px',
    fontWeight: '500',
```

```
    transition: 'all 0.3s ease',
    backgroundColor: 'rgba(255, 255, 255, 0.1)',
    border: '1px solid rgba(255, 255, 255, 0.2)'
  };

  const buttonStyle = {
    ...linkStyle,
    background: 'rgba(255, 255, 255, 0.2)',
    border: '1px solid rgba(255, 255, 255, 0.3)',
    cursor: 'pointer'
  };

  return (
    <nav style={navStyle}>
      <div style={navContainerStyle}>
        <div style={{ display: 'flex', alignItems: 'center', gap: '8px', flexWrap:
'wrap' }}>
          <Link
            to="/restaurants"
            style={linkStyle}
            onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
            onMouseOut={e => e.target.style.backgroundColor = 'rgba(255, 255,
255, 0.1)'}
          >
            Restaurants
          </Link>
          <Link
            to="/search"
            style={linkStyle}
            onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
            onMouseOut={e => e.target.style.backgroundColor = 'rgba(255, 255,
255, 0.1)'}
          >



            Search
          </Link>
          {isAuthenticated && role === 'ADMIN' && (
            <Link
              to="/reservations"
              style={linkStyle}
              onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
              onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
```

```
255, 255, 0.1)'}
                >
                  Reservations
                </Link>
              )}
              {isAuthenticated && role === 'ADMIN' && (
                <Link
                  to="/admin"
                  style={linkStyle}
                  onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
                  onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.1)'}
                >
                  Admin
                </Link>
              )}
              {isAuthenticated && role === 'OWNER' && (
                <Link
                  to="/owner"
                  style={linkStyle}
                  onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
                  onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.1)'}
                >
                  Owner
                </Link>
              )}
            </div>

            <div style={{ display: 'flex', alignItems: 'center', gap: '8px', flexWrap:
'wrap' }}>
              {!isAuthenticated && (
                <Link
                  to="/login"
                  style={linkStyle}

                  onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
                  onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.1)'}
                >
                  Login
                </Link>
              )}
              {!isAuthenticated && (
```

```
                <Link
                  to="/register"
                  style={linkStyle}
                  onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
                  onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.1)'}
                >
                  Register
                </Link>
              )}
              {isAuthenticated && (
                <button
                  onClick={logout}
                  style={buttonStyle}
                  onMouseOver={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.3)'}
                  onMouseOut={e => e.target.style.backgroundColor = 'rgba(255,
255, 255, 0.2)'}
                >
                  Logout
                </button>
              )}
            </div>
          </div>
        </nav>
    );
}

function App() {
    const containerStyle = {
      fontFamily: 'system-ui, -apple-system, sans-serif',
      background: 'linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%)',
      minHeight: '100vh'
    };

    const contentStyle = {
      maxWidth: '1200px',
      margin: '0 auto',
      padding: '24px'
    };

    return (
      <div style={containerStyle}>
        <Nav />
        <div style={contentStyle}>
          <Routes>
```

```
            <Route path="/" element={<Navigate to="/restaurants" />} />
            <Route path="/restaurants" element={<RestaurantList />} />
            <Route path="/restaurants/:id" element={<RestaurantDetail />} />
            <Route path="/search" element={<RestaurantSearch />} />
            <Route
              path="/reservations"
              element={
                <RequireAuth roles={['ADMIN']}>
                  <ReservationList />
                </RequireAuth>
              }
            />
            <Route
              path="/admin"
              element={
                <RequireAuth roles={['ADMIN']}>
                  <AdminDashboard />
                </RequireAuth>
              }
            />
            <Route
              path="/owner"
              element={
                <RequireAuth roles={['OWNER', 'ADMIN']}>
                  <OwnerDashboard />
                </RequireAuth>
              }
            />
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
          </Routes>
        </div>
      </div>
    );
}

export default App;
```

App.css

```
.App {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}
```

```
.navbar {
   background-color: #333;
   color: white;
   padding: 1rem 2rem;
   display: flex;
   justify-content: space-between;
   align-items: center;
}

.nav-brand a {
   color: white;
   text-decoration: none;
   font-size: 1.5rem;
   font-weight: bold;
}

.nav-links {
   display: flex;
   list-style: none;
   margin: 0;
   padding: 0;
   gap: 2rem;
}

.nav-links a {
   color: white;
   text-decoration: none;
   padding: 0.5rem 1rem;
   border-radius: 4px;
   transition: background-color 0.3s;
}

.nav-links a:hover {
   background-color: #555;
}

.main-content {
   flex: 1;
   padding: 2rem;
   max-width: 1200px;
   margin: 0 auto;
   width: 100%;
}

.btn {
   padding: 0.5rem 1rem;
   border: none;
```

```
    border-radius: 4px;
    cursor: pointer;
    text-decoration: none;
    display: inline-block;
    text-align: center;
    transition: background-color 0.3s;
}

.btn-primary {
    background-color: #007bff;
    color: white;
}

.btn-primary:hover {
    background-color: #0056b3;
}

.btn-success {
    background-color: #28a745;
    color: white;
}

.btn-success:hover {
    background-color: #1e7e34;
}

.btn-danger {
    background-color: #dc3545;
    color: white;
}

.btn-danger:hover {
    background-color: #c82333;
}

.btn-info {
    background-color: #17a2b8;
    color: white;
}

.btn-info:hover {
    background-color: #138496;
}

.btn:disabled {
    opacity: 0.6;
    cursor: not-allowed;
}
```

Index.js

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import { AuthProvider } from './context/AuthContext';
import CssBaseline from '@mui/material/CssBaseline';

const container = document.getElementById('root');
const root = createRoot(container);
root.render(
  <BrowserRouter>
    <AuthProvider>
      <CssBaseline />
      <App />
    </AuthProvider>
  </BrowserRouter>
);
```

5.8.2 BACKEND CODE

SwaggerConfig.java

```
package com.examly.springapp.config;

import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.OpenAPI;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
                .info(new Info()
                        .title("Restaurant Reservation System API")
                        .version("1.0")
                        .description("API documentation for Restaurant Table Reservation
System")
                        .contact(new Contact()
                                .name("Restaurant API Support")
                                .email("support@restaurant.com")));
    }
}
```

ReservationController.java

```
package com.examly.springapp.controller;

import com.examly.springapp.dto.ReservationRequest;
import com.examly.springapp.model.*;
import com.examly.springapp.service.ReservationService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDate;
import java.time.LocalTime;
```

```
import java.util.List;

@RestController
@RequestMapping("/api/reservations")
@CrossOrigin(origins="*")
@RequiredArgsConstructor
public class ReservationController {

    private final ReservationService reservationService;

    @PostMapping
    @PreAuthorize("hasRole('CUSTOMER')")
    public ResponseEntity<Reservation> create(@Valid @RequestBody
ReservationRequest req) {
        Reservation toCreate = Reservation.builder()
            .customerName(req.getCustomerName())
            .customerEmail(req.getCustomerEmail())
            .customerPhone(req.getCustomerPhone())
            .reservationDate(LocalDate.parse(req.getReservationDate()))
            .reservationTime(LocalTime.parse(req.getReservationTime()))
            .partySize(req.getPartySize())
            .specialRequests(req.getSpecialRequests() == null ? "" :
req.getSpecialRequests())
            .status(ReservationStatus.PENDING)
            .build();
        Reservation saved = reservationService.createReservation(toCreate,
req.getRestaurantId());
        return ResponseEntity.ok(saved);
    }

    @GetMapping
    @PreAuthorize("hasAnyRole('ADMIN')")
    public ResponseEntity<List<Reservation>> getAll() {
        return ResponseEntity.ok(reservationService.getAllReservations());
    }

    @GetMapping("/by-restaurant/{restaurantId}")
    @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
    public ResponseEntity<List<Reservation>> getByRestaurant(@PathVariable
Long restaurantId) {
        return ResponseEntity.ok(reservationService.getByRestaurant(restaurantId));
    }

    @PutMapping("/{id}/confirm")
    @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
    public ResponseEntity<Reservation> confirm(@PathVariable Long id) {
```

```
            return ResponseEntity.ok(reservationService.updateReservationStatus(id,
ReservationStatus.CONFIRMED));
        }
        @PutMapping("/{id}/reject")
        @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
        public ResponseEntity<Reservation> reject(@PathVariable Long id) {
            return ResponseEntity.ok(reservationService.updateReservationStatus(id,
ReservationStatus.CANCELED));
        }

        @PutMapping("/{id}/cancel")
        @PreAuthorize("hasAnyRole('CUSTOMER','OWNER','ADMIN')")
        public ResponseEntity<Void> cancel(@PathVariable Long id) {
            reservationService.cancelReservation(id);
            return ResponseEntity.noContent().build();
        }


        @PutMapping("/{id}/status")
        @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
        public ResponseEntity<Reservation> updateStatus(@PathVariable Long id,
                @RequestParam ReservationStatus status) {
            return ResponseEntity.ok(reservationService.updateReservationStatus(id,
status));
        }


        @GetMapping("/availability/{restaurantId}")
        public ResponseEntity<Long> getAvailability(@PathVariable Long restaurantId,
                @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
LocalDate date) {
            List<Reservation> sameDay =
reservationService.getByRestaurant(restaurantId).stream()
                    .filter(r -> r.getReservationDate().equals(date))
                    .toList();
            long booked = sameDay.size();
            return ResponseEntity.ok(booked);
        }
    }
```

RestaurantController.java

```
package com.examly.springapp.controller;

import com.examly.springapp.model.Restaurant;
import com.examly.springapp.model.Role;
import com.examly.springapp.model.User;
```

```
import com.examly.springapp.repository.UserRepository;
import com.examly.springapp.service.RestaurantService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/restaurants")
@CrossOrigin(origins="*")
@RequiredArgsConstructor
public class RestaurantController {

    private final RestaurantService restaurantService;
    private final UserRepository userRepository;

    @GetMapping
    public ResponseEntity<List<Restaurant>> getAll() {
        return ResponseEntity.ok(restaurantService.getAllRestaurants());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Restaurant> getById(@PathVariable Long id) {
        return ResponseEntity.ok(restaurantService.getRestaurantById(id));
    }

    @GetMapping("/search")
    public ResponseEntity<List<Restaurant>> search(@RequestParam(required =
false) String cuisine,
            @RequestParam(required = false) String name) {
        if (cuisine != null) {
            return ResponseEntity.ok(restaurantService.searchByCuisine(cuisine));
        }

        return ResponseEntity.ok(restaurantService.getAllRestaurants());
    }

    @PostMapping("/owner")
    @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
    public ResponseEntity<Restaurant> create(@Valid @RequestBody Restaurant
restaurant, Authentication auth) {
```

```
        if (auth != null && auth.getAuthorities().stream().anyMatch(a ->
a.getAuthority().equals("ROLE_OWNER"))) {
            User owner = userRepository.findByEmail(auth.getName()).orElse(null);
            restaurant.setOwner(owner);
        }
        return ResponseEntity.ok(restaurantService.createRestaurant(restaurant));
    }

    @PutMapping("/owner/{id}")
    @PreAuthorize("hasAnyRole('OWNER','ADMIN')")
    public ResponseEntity<Restaurant> update(@PathVariable Long id, @Valid
@RequestBody Restaurant restaurant) {
        return ResponseEntity.ok(restaurantService.updateRestaurant(id, restaurant));
    }

    @DeleteMapping("/owner/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        restaurantService.deleteRestaurant(id);
        return ResponseEntity.noContent().build();
    }
}
```

ResourceNotFoundException.java

```
package com.examly.springapp.exception;

public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String msg) {
        super(msg);
    }
}
```

Reservation.java

```
package com.examly.springapp.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;
import java.time.LocalDate;
import java.time.LocalTime;

@Entity
@Table(name = "reservations")
```

```
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Reservation {

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

    @ManyToOne(optional = false, fetch = FetchType.EAGER)
    @JoinColumn(name = "restaurant_id")
    private Restaurant restaurant;

    @NotBlank
    private String customerName;

    @Email
    @NotBlank
    private String customerEmail;

    @NotBlank
    private String customerPhone;

    @NotNull
    private LocalDate reservationDate;

    @NotNull
    private LocalTime reservationTime;

    @Min(1)
    private Integer partySize;

    @Enumerated(EnumType.STRING)
    private ReservationStatus status;

    @Column(columnDefinition = "TEXT")
    private String specialRequests;
}
```

ReservationStatus.java

```
package com.examly.springapp.model;

public enum ReservationStatus {
   PENDING, CONFIRMED, CANCELED, SEATED
}
```

Restaurant.java

```
package com.examly.springapp.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.*;

import java.time.LocalTime;

@Entity
@Table(name = "restaurants")
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Restaurant {

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

   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "owner_id")
   private User owner;

   @NotBlank
   private String name;

   @NotBlank
   private String address;

   @Column(name = "cuisine")
   private String cuisine;

   private LocalTime openingTime;
   private LocalTime closingTime;
```

```
    private Integer totalTables;
}
```

ReservationRepository.java

```
package com.examly.springapp.repository;

import com.examly.springapp.model.Reservation;
import com.examly.springapp.model.ReservationStatus;
import org.springframework.data.jpa.repository.JpaRepository;

import java.time.LocalDate;
import java.util.List;

public interface ReservationRepository extends JpaRepository<Reservation, Long> {
    List<Reservation> findByRestaurant_IdAndReservationDate(Long restaurantId,
LocalDate reservationDate);

    List<Reservation> findByRestaurant_Id(Long restaurantId);
    List<Reservation> findByStatus(ReservationStatus status);

    long countByStatus(ReservationStatus status);

    long countByRestaurant_Id(Long restaurantId);
}
```

RestaurantRepository.java

```
package com.examly.springapp.repository;

import com.examly.springapp.model.Restaurant;
import com.examly.springapp.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface RestaurantRepository extends JpaRepository<Restaurant, Long> {
    List<Restaurant> findByCuisineIgnoreCase(String cuisine);

    List<Restaurant> findByOwner(User owner);

    List<Restaurant>
findByNameContainingIgnoreCaseOrCuisineContainingIgnoreCase(String name, String
cuisine);
}
```

ReservationService.java

```
package com.examly.springapp.service;

import com.examly.springapp.exception.BadRequestException;
import com.examly.springapp.exception.ResourceNotFoundException;
import com.examly.springapp.model.*;
import com.examly.springapp.repository.ReservationRepository;
import com.examly.springapp.repository.RestaurantRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ReservationService {

    private final ReservationRepository reservationRepository;
    private final RestaurantRepository restaurantRepository;
    private final NotificationService notificationService;


    public Reservation createReservation(Reservation reservation, Long restaurantId)
{
        Restaurant restaurant = restaurantRepository.findById(restaurantId)
                .orElseThrow(() -> new ResourceNotFoundException("Restaurant not
found with id: " + restaurantId));


        reservation.setRestaurant(restaurant);

        List<Reservation> sameDayReservations = reservationRepository

                .findByRestaurant_IdAndReservationDate(restaurantId,
reservation.getReservationDate());

        Integer totalTables = restaurant.getTotalTables() == null ? 0 :
restaurant.getTotalTables();
        if (sameDayReservations.size() >= totalTables) {
            throw new BadRequestException("Capacity full for the selected date.");
        }

        if (reservation.getStatus() == null) {
```

```
        reservation.setStatus(ReservationStatus.PENDING);
            }

            Reservation saved = reservationRepository.save(reservation);


            if (restaurant.getOwner() != null) {
                notificationService.notifyUser(restaurant.getOwner(),
                        "New reservation request from " + reservation.getCustomerName() + "
for "
                                + reservation.getReservationDate());
            }
            return saved;
        }

        public List<Reservation> getAllReservations() {
            return reservationRepository.findAll();
        }

        public Reservation updateReservationStatus(Long reservationId,
ReservationStatus newStatus) {
            Reservation existing = reservationRepository.findById(reservationId)
                    .orElseThrow(() -> new ResourceNotFoundException("Reservation not
found with id: " + reservationId));
            existing.setStatus(newStatus);
            Reservation saved = reservationRepository.save(existing);


            if (existing.getRestaurant() != null && existing.getRestaurant().getOwner() !=
null) {
                notificationService.notifyUser(existing.getRestaurant().getOwner(),
                        "Reservation " + existing.getId() + " status updated to " + newStatus);
            }
            return saved;
        }

        public void cancelReservation(Long reservationId) {
            Reservation existing = reservationRepository.findById(reservationId)
                    .orElseThrow(() -> new ResourceNotFoundException("Reservation not
found with id: " + reservationId));
            reservationRepository.delete(existing);
        }


        public List<Reservation> getByRestaurant(Long restaurantId) {
            return reservationRepository.findByRestaurant_Id(restaurantId);
        }
```

```
    public long countByStatus(ReservationStatus status) {
        return reservationRepository.countByStatus(status);
    }

    public long countByRestaurant(Long restaurantId) {
        return reservationRepository.countByRestaurant_Id(restaurantId);
    }
}
```

RestaurantService.java

```
package com.examly.springapp.service;

import com.examly.springapp.exception.ResourceNotFoundException;
import com.examly.springapp.model.Restaurant;
import com.examly.springapp.model.User;
import com.examly.springapp.repository.RestaurantRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class RestaurantService {

    private final RestaurantRepository restaurantRepository;

    public Restaurant createRestaurant(Restaurant restaurant) {
        return restaurantRepository.save(restaurant);
    }

    public List<Restaurant> getAllRestaurants() {
        return restaurantRepository.findAll();
    }

    public Restaurant getRestaurantById(Long id) {
        return restaurantRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Restaurant not
found with id: " + id));
    }

    public List<Restaurant> searchByCuisine(String cuisine) {
        return restaurantRepository.findByCuisineIgnoreCase(cuisine);
    }
```

```
    public List<Restaurant> getByOwner(User owner) {
        return restaurantRepository.findByOwner(owner);
    }

    public Restaurant updateRestaurant(Long id, Restaurant updates) {
        Restaurant existing = getRestaurantById(id);
        existing.setName(updates.getName());
        existing.setAddress(updates.getAddress());
        existing.setCuisine(updates.getCuisine());
        existing.setOpeningTime(updates.getOpeningTime());
        existing.setClosingTime(updates.getClosingTime());
        existing.setTotalTables(updates.getTotalTables());
        return restaurantRepository.save(existing);
    }

    public void deleteRestaurant(Long id) {
        Restaurant existing = getRestaurantById(id);
        restaurantRepository.delete(existing);
    }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.4.0</version>
        <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.examly</groupId>
  <artifactId>springapp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Restaurant Reservation System</name>
  <description>Restaurant Table Reservation System with Spring Boot</description>
  <url/>
  <licenses>
        <license/>
  </licenses>
  <developers>
        <developer/>
  </developers>
```

```
<scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
</scm>
<properties>
        <java.version>17</java.version>
        <jjwt.version>0.11.5</jjwt.version>
</properties>
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
                <groupId>com.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
        <dependency>
           <groupId>org.springdoc</groupId>
                <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
                <version>2.7.0</version>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
```

```
        </dependency>
        <dependency>
                <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-api</artifactId>
                <version>${jjwt.version}</version>
        </dependency>
        <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt-impl</artifactId>
                <version>${jjwt.version}</version>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt-jackson</artifactId>
                <version>${jjwt.version}</version>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
                <scope>test</scope>
        </dependency>

</dependencies>

<build>
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <annotationProcessorPaths>
                                        <path>
                                                <groupId>org.projectlombok</groupId>
                                                <artifactId>lombok</artifactId>
                                        </path>
                                </annotationProcessorPaths>
                        </configuration>
                </plugin>
                <plugin>
                        <groupId>org.springframework.boot</groupId>
```

```
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

In conclusion, the proposed Restaurant Table Reservation System provides a simplified, secure, and efficient platform for managing restaurant reservations and user interactions. The system enables users to search for restaurants, make and manage reservations, and view detailed booking histories in real time. With its integration of Spring Boot as the backend, React.js as the frontend, and MySQL as the database, the project demonstrates how modern web technologies can be effectively combined to ensure data consistency, accuracy, and reliability in reservation-based applications.

The system enhances transparency by maintaining a complete record of reservations, improves efficiency by automating validations, and ensures atomicity in booking operations. Its modular design makes the application scalable, user-friendly, and adaptable to future enhancements. By providing a responsive interface, meaningful error handling, and reliable performance, the Restaurant Table Reservation System ensures that users and restaurant owners have a seamless digital reservation experience. Overall, this project successfully showcases the application of full-stack development in building real-world solutions for the hospitality industry.

6.2 FUTURE SCOPE

- Integration of Authentication and Security Features: Implementing robust user authentication, JWT-based authorization, and role-based access control to enhance system security and protect sensitive data.
- Mobile Application Development: Creating Android and iOS mobile apps for both customers and restaurant owners, increasing accessibility and allowing reservation management on the go.
- Advanced Analytics and Reporting: Adding features for generating reservation

statistics, customer insights, and peak time analytics to help restaurants optimize operations.

- Online Payment Integration: Extending the system to support secure online payments and reservation deposits.
- Table Management & Dynamic Seating: Supporting advanced features such as dynamic table allocation, waitlist management, and real-time table tracking.
- Enhanced UI/UX Features: Incorporating dashboards with graphs, charts, and data visualization to provide users and owners with clear insights into reservation trends.
- AI and Chatbot Integration: Implementing AI-powered assistants for customer support, automated reservation handling, and recommendation systems.
- Scalability and Cloud Deployment: Deploying the system on cloud platforms (AWS/Azure) to support large-scale usage, high availability, and disaster recovery.

CHAPTER 7

REFERENCES

- **GitHub – Restaurant Reservation System Projects**
  Sample implementations of online reservation systems using React and Spring Boot.
  https://github.com/topics/restaurant-reservation

- **TutorialsPoint – Restaurant Management System**
  Learning resource for building restaurant management applications with database integration.
  https://www.tutorialspoint.com/restaurant-management-system

- **Medium – Building a Reservation Application with React and Spring Boot**
  Articles and tutorials on integrating full-stack applications for reservation systems.
  https://medium.com

- **Stack Overflow – Restaurant Reservation Application Development Discussions**
  Community-driven solutions for common issues in reservation app development.
  https://stackoverflow.com

- **CodeProject – Restaurant and Reservation Management Systems**
  Sample code and project ideas related to online reservation systems.
  https://www.codeproject.com