Hotel Booking System (React)

Project Submitted to the

SRM University AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology in

Computer Science & Engineering

School of Engineering & Sciences

Submitted by

K.suhash kowsick (AP23110011247)

S.Yellareddy (AP23110011648)

B.Akhil (AP23110011662)

G.Sai Kumar (AP23110011649)

Under the Guidance of

MR .YATHARTH SHAHRAWAT

Department of Computer Science & Engineering

SRM University-Ap

Neeru Konda, Mangalgiri, Guntur

Andhra Pradesh - 522 240

December 2025

Table of Contents

Abstract:-

The Hotel Booking System is a comprehensive web-based application designed to automate and streamline the process of hotel room browsing, booking, and management. Built using **React** for the frontend and **JSON Server** as a lightweight backend, the system provides an intuitive and efficient platform for both customers and administrators. The primary goal of the project is to replace traditional manual room booking methods with a digital solution that enhances accessibility, accuracy, and user experience.

For customers, the system allows seamless account creation, secure login, room exploration, detailed room viewing, and convenient booking with check-in and check-out date selection. Users can manage their reservations by editing or cancelling bookings directly from their dashboard. A dedicated profile section enables users to update their information and view account-related statistics, ensuring personalized interaction.

On the administrative side, the system offers a separate login portal where admins can add new rooms, update existing room information, delete rooms, and monitor all user bookings. This ensures full control over room availability and booking operations. The use of **Context API** enables smooth state management across the application, while **React Router** ensures fast and structured navigation between pages.

JSON Server serves as a mock backend, storing persistent data such as rooms, users, bookings, and admin credentials in a structured manner. It supports complete CRUD operations, allowing the system to mimic real-world backend behavior during development. Tailwind CSS is used to enhance the visual appeal and responsiveness of the user interface.

This project demonstrates essential concepts in full-stack development, including component-based architecture, API integration, state management, routing, and backend simulation. The Hotel Booking System provides a scalable and user-friendly platform that can be expanded with real backend services in the future. Overall, it showcases an efficient and modern approach to hotel booking management, offering a reliable solution for both customers and administrators.

Introduction:-

Welcome to the **Hotel Booking System**, a modern and user-friendly online hotel reservation platform built using the power of **React.js, Vite, Tailwind CSS, and JSON Server**. Designed with a clean, responsive, and intuitive interface, this application transforms the way users explore rooms, manage bookings, and experience hotel services digitally.

Crafted for today's digital travelers, the platform combines real-time room availability with a smooth and interactive design that allows users to browse rooms, view detailed information, and book instantly with just a few clicks. Whether it's a Deluxe Room, Family Room, or a Luxury Suite, users can easily find the perfect stay with transparent pricing in Indian Rupees and full booking control.

Powered by **React's component-driven architecture** and **Vite's high-speed build system**, the Hotel Booking System ensures a fast, seamless, and highly responsive experience across all devices. From browsing rooms to managing profiles and bookings, every feature is optimized for simplicity, speed, and reliability.

Say goodbye to traditional, time-consuming hotel reservations — the Hotel Booking System delivers a smarter, faster, and more efficient way to book your perfect stay. Welcome to the future of digital hotel management.

Scenario-Based Intro:-

Imagine it's early morning. You wake up, grab your phone, and start planning a quick trip. Instead of calling hotels or searching multiple websites, you simply open the **Hotel Booking System**. The platform welcomes you with a clean and responsive interface showing beautifully listed rooms with prices, images, and amenities.

While traveling, you browse through Deluxe, Family, and Suite rooms with ease. With just a few taps, you check room availability, compare prices in Indian Rupees, and view detailed descriptions. Once you find the perfect room, you log in, select your check-in and check-out dates, and complete your booking instantly.

Later at your destination, you open the system on your laptop. Thanks to smooth navigation and real-time data updates, your booking details, profile information, and room status are perfectly synchronized. You can view, edit, or even cancel your reservation effortlessly.

At the same time, the hotel admin logs into the admin panel and manages room details, updates prices, adds new rooms, and monitors all customer bookings in real time. With its responsive design, fast performance, and secure access, the Hotel Booking System delivers a seamless digital experience for both customers and administrators — all in one powerful platform.

Target Audience:-

The **Hotel Booking System** is tailored for:

- **Travelers & Tourists**
  Individuals who want quick, easy, and reliable access to hotel room bookings without visiting multiple websites or making phone calls.
- **Comfort Seekers**
  Users who prefer to compare rooms, prices, and amenities before booking the perfect stay.
- **Hotel Owners & Managers**
  Business owners who want a simple digital platform to manage rooms, update availability, and track customer bookings efficiently.
- **Multi-Device Users**
  People who browse and book using both mobile phones and laptops and expect a smooth, consistent booking experience across all devices.

Project Goals and Objectives:-

Primary Goal

The primary goal of our project is to create a **modern, fast, and user-friendly hotel booking platform** that provides users with a smooth and reliable online reservation experience. The system focuses on **performance, simplicity, and real-time availability**, allowing users to browse rooms, compare prices, and book instantly with ease.

By enabling users to explore detailed room information, manage bookings, and control their profiles, the platform ensures a **personalized and convenient hotel booking experience**. At the same time, the system empowers administrators with full control over room management and customer bookings through a secure and efficient admin panel.

Our primary goal is to build a **fast, modern, and reliable hotel booking system** where users can book rooms effortlessly and administrators can manage hotel operations digitally, ensuring a smooth and efficient experience for everyone.

Objectives

1. User-Friendly Interface

The project aims to design a **highly intuitive and easy-to-use interface** that enhances the hotel booking experience for all users. This includes:

- **Smooth navigation with well-organized menus and room categories**, enabling users to quickly browse available rooms, view details, and proceed with bookings without confusion.
- **Clean layouts, clear typography, and visually appealing room cards**, ensuring users can easily read room details, pricing, and amenities without strain.
- **Fully responsive design**, allowing the platform to adapt seamlessly across mobile phones, tablets, and desktop screens for a consistent experience everywhere.

  Overall, this objective ensures that the Hotel Booking System feels **modern, comfortable, professional, and accessible** for every user.

2. Advanced Theme System

The project aims to provide users with a **fast, safe, and reliable room booking experience** through a well-structured and secure booking system. This includes:

- **Real-time room availability and instant booking confirmation**, ensuring users can reserve rooms without delays or confusion.
- **Easy date selection and booking management**, allowing users to choose check-in and check-out dates, view booking history, edit details, or cancel reservations smoothly.
- **Data persistence and booking security**, where all booking records are safely stored in the backend and automatically retrieved whenever the user logs in again.

  This objective ensures that users enjoy a **stress-free, secure, and trustworthy hotel booking experience** from start to finish.

3. Personalization Tools

The platform provides several features that make the hotel booking experience **unique and personalized for each user**. These include:

- **Personalized user profiles**, where users can view and update their personal information and manage their booking history.
- **Customized room browsing**, allowing users to explore rooms based on their preferences such as room type, budget, and amenities.
- **Persistent user data**, ensuring that profile details, bookings, and activity remain saved and accessible every time the user logs in.

This objective focuses on giving users the **freedom to personalize their booking journey**, making the Hotel Booking System more convenient, engaging, and user-centric.

4. Modern Technology Use

To ensure **high performance, reliability, and scalability**, the Hotel Booking System is built using a modern and efficient technology stack. This includes:

- **React 18** for fast component rendering, smooth navigation, and a highly interactive user experience.
- **Vite** for rapid development, instant hot reloads, and optimized production builds for better performance.
- **React Context API** for lightweight and effective global state management of authentication and user data.
- **Tailwind CSS** for building clean, responsive, and visually consistent user interface designs quickly.
- **JSON Server** for handling backend operations such as room data, user accounts, and booking records.

This objective ensures that the system remains **fast, maintainable, scalable, and future-ready**, providing a strong technical foundation for real-world hotel booking applications.

Key Features:

1. Centralized Room Listing & Browsing

The platform's core functionality is to act as a **central hub for hotel room discovery** by displaying all available rooms in one unified interface. Instead of users visiting multiple hotel websites or contacting different hotels, the system automatically organizes and presents all room options in a single, easy-to-use platform. Users can view room types, prices, images, and amenities at one place, making the room selection process fast, simple, and efficient.

2. Multiple Room Categories & Types

The platform offers a wide variety of **hotel room categories**, allowing users to choose the perfect stay based on comfort, budget, and purpose. These include:

- **Standard Rooms** – Budget-friendly and comfortable
- **Deluxe Rooms** – Premium comfort with upgraded amenities
- **Family Rooms** – Spacious rooms suitable for families
- **Suite Rooms** – Luxury rooms for premium stays
- **Balcony House / Special Rooms** – For enhanced experience

This feature gives users **flexibility and choice**, ensuring that every customer finds a room that perfectly matches their needs.

3. UI Optimization & Performance Enhancement

The system is optimized for **high performance, smooth navigation, and clear visibility** across all pages. Improved layouts, clean typography, and fast page loading ensure that users can browse rooms, view details, and complete bookings without delays or confusion. This also enhances readability and usability on both light and dark display settings.

4. Booking History & Management System

Users can **save, view, and manage their bookings** through a dedicated bookings page. This feature allows users to:

- View past and upcoming reservations
- Edit booking details
- Cancel bookings when required
  This makes reservation management simple, organized, and user-friendly.

5. Fully Responsive User Interface

The platform is designed using a **mobile-first approach** with adaptive layouts that work smoothly on:

- Mobile phones
- Tablets

- Laptops and Desktop systems
  Users experience consistent performance and navigation across all devices without any layout issues.

6. Admin & User Access Control System

The system supports **separate access for users and administrators**:

- Users can register, log in, and manage their own bookings
- Admins have a dedicated admin panel to manage rooms, pricing, and all customer bookings
  This ensures **secure role-based access** and proper system control.

7. Personalized Profile & Persistent User Data

The platform allows users to maintain a **personalized profile** where their information, booking history, and preferences are securely stored. All user data **persists across sessions**, meaning users can log in anytime and instantly access their profile details and previous bookings without losing information. This feature enhances convenience and provides a more personalized and reliable user experience.

Pre-Requisites:-

1. Node.js & npm

Required for running **React**, **Vite**, and managing all project dependencies. npm (Node Package Manager) is used to install and manage required libraries and tools for the Hotel Booking System.

Download:
https://nodejs.org/en/download/

2. React.js

A JavaScript library used for building the **user interface using reusable components**. React enables fast rendering, smooth navigation, and a dynamic booking experience.

Create Project Using Vite:

```
npm create vite@latest
cd project-name
npm install
npm run dev
```

3. HTML, CSS, and JavaScript

These technologies form the **foundation of the web application**.

- **HTML** is used to structure the content of the Hotel Booking System, such as room cards, navigation bars, booking forms, and layout sections.
- **CSS (Tailwind CSS)** is used to style these components, ensuring the user interface looks visually appealing, responsive, and consistent across all devices.
- **JavaScript** adds interactivity — such as handling button clicks, login validation, booking actions, date selection, admin operations, and dynamic updates from the backend.

4. Version Control – Git & GitHub

Git and GitHub are used for **code management, version tracking, and team collaboration**. They help in maintaining multiple versions of the project, tracking changes, and safely storing the source code online.

5. Development Environment

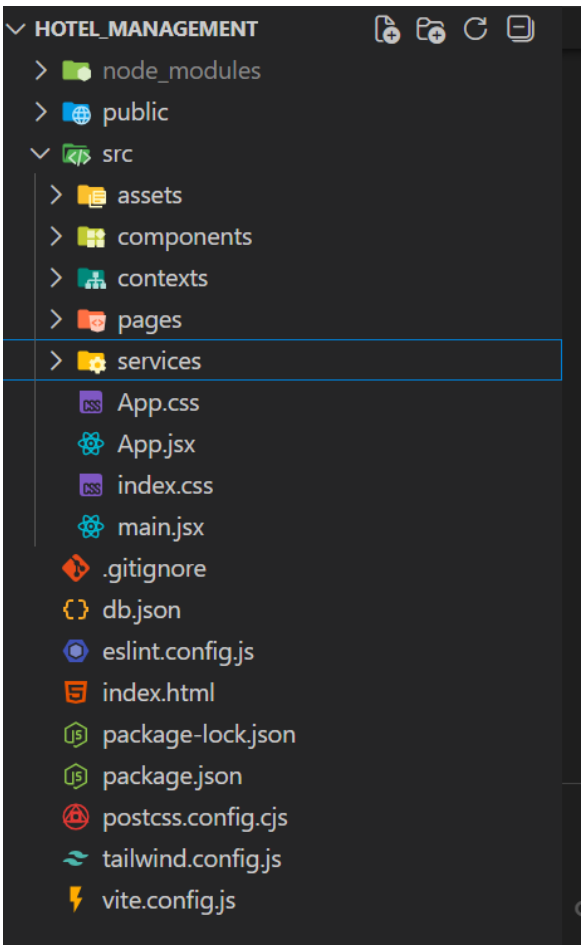
A good code editor is required to write, test, and debug the application efficiently.

Recommended Editors:

- VS Code
- WebStorm
- Sublime Text

These tools provide features such as syntax highlighting, debugging, extensions, and terminal integration for smooth development.

Project Structure:-



PROJECT FLOW:-

Milestone 1: Setup & Configuration

1. Installed required tools:

- React JS
- Vite
- Tailwind CSS
- React Router
- React Icons

2. Created base folder structure
3. Configured theme provider & global state
4. Integrated API services (JSON Server /)

Milestone2: Web Development

Setup React Application

- Create React application.
- Configure Routing.
- Install required libraries.

App.js

src > App.jsx > ...

```jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './contexts/AuthContext';
import { PrivateRoute, AdminRoute, AuthenticatedRoute } from './components/PrivateRoute';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Rooms from './pages/Rooms';
import RoomDetails from './pages/RoomDetails';
import Booking from './pages/Booking';
import BookingsList from './pages/BookingsList';
import Admin from './pages/Admin';
import Login from './pages/Login';
import Signup from './pages/Signup';
import AdminLogin from './pages/AdminLogin';
import Profile from './pages/Profile';

/**
 * Main App Component
 * Sets up routing and authentication for the entire application
 */
function App() {
  return (
    <AuthProvider>
      <Router>
        <div className="min-h-screen bg-gray-50">
          <Navbar />
          <Routes>
            {/* Public Routes */}
            <Route path="/" element={<Home />} />
            <Route path="/login" element={<Login />} />
            <Route path="/signup" element={<Signup />} />
            <Route path="/admin-login" element={<AdminLogin />} />
            <Route path="/rooms" element={<Rooms />} />
            <Route path="/rooms/:id" element={<RoomDetails />} />

            {/* Protected User Routes */}
            <Route
              path="/booking/:id"
              element={
                <PrivateRoute>
                  <Booking />
                </PrivateRoute>
```

```
                </PrivateRoute>
              }
            />
            <Route
              path="/bookings"
              element={
                <AuthenticatedRoute>
                  <BookingsList />
                </AuthenticatedRoute>
              }
            />
            <Route
              path="/profile"
              element={
                <AuthenticatedRoute>
                  <Profile />
                </AuthenticatedRoute>
              }
            />

            {/* Protected Admin Routes */}
            <Route
              path="/admin"
              element={
                <AdminRoute>
                  <Admin />
                </AdminRoute>
              }
            />
          </Routes>
        </div>
      </Router>
    </AuthProvider>
  );
}

export default App;
```

Code Description:-

- This file defines the **main structure and routing system** of the Hotel Booking Web Application.
- It wraps the entire app inside **AuthProvider** to manage user authentication globally.
- The **Router** handles navigation between pages like Home, Rooms, Login, Signup, and Admin Login.
- Public pages can be accessed by anyone, while **PrivateRoute** restricts booking pages to logged-in users only.
- **AuthenticatedRoute** protects pages like Bookings List and Profile so only logged-in users can view them.
- **AdminRoute** ensures only admins can access the Admin Dashboard.
  A common **Navbar** is displayed on all pages for navigation.
- Overall, this component manages both routing and access control for the entire application.

Api.js :-

```
import axios from 'axios';
const API_BASE_URL = 'http://localhost:3000';
const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});
/**
 * Get all rooms
 * @returns {Promise} Array of rooms
 */
export const getRooms = async () => {
  try {
    const response = await api.get('/rooms');
    return response.data;
  } catch (error) {
    console.error('Error fetching rooms:', error);
    throw error;
  }
};

/**
 * Get a single room by ID
 * @param {number} id - Room ID
 * @returns {Promise} Room object
 */
export const getRoomById = async (id) => {
  try {
    const response = await api.get(`/rooms/${id}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching room:', error);
    throw error;
  }
};

/**
 * Create a new room (Admin)
 * @param {Object} roomData - Room data
 * @returns {Promise} Created room object
```

```
 * @returns {Promise} Created room object
 */
export const createRoom = async (roomData) => {
  try {
    const response = await api.post('/rooms', roomData);
    return response.data;
  } catch (error) {
    console.error('Error creating room:', error);
    throw error;
  }
};

/**
 * Update a room (Admin)
 * @param {number} id - Room ID
 * @param {Object} roomData - Updated room data
 * @returns {Promise} Updated room object
 */
export const updateRoom = async (id, roomData) => {
  try {
    const response = await api.put(`/rooms/${id}`, roomData);
    return response.data;
  } catch (error) {
    console.error('Error updating room:', error);
    throw error;
  }
};

/**
 * Delete a room (Admin)
 * @param {number} id - Room ID
 * @returns {Promise}
 */
export const deleteRoom = async (id) => {
  try {
    const response = await api.delete(`/rooms/${id}`);
    return response.data;
  } catch (error) {
    console.error('Error deleting room:', error);
    throw error;
  }
```

```
    const response = await api.post('/bookings', bookingData);
    return response.data;
  } catch (error) {
    console.error('Error creating booking:', error);
    throw error;
  }
};

/**
 * Update a booking
 * @param {number} id - Booking ID
 * @param {Object} bookingData - Updated booking data
 * @returns {Promise} Updated booking object
 */
export const updateBooking = async (id, bookingData) => {
  try {
    const response = await api.put(`/bookings/${id}`, bookingData);
    return response.data;
  } catch (error) {
    console.error('Error updating booking:', error);
    throw error;
  }
};

/**
 * Delete a booking
 * @param {number} id - Booking ID
 * @returns {Promise}
 */
export const deleteBooking = async (id) => {
  try {
    const response = await api.delete(`/bookings/${id}`);
    return response.data;
  } catch (error) {
    console.error('Error deleting booking:', error);
    throw error;
  }
};
```

```
};
/**
 * Get all bookings
 * @returns {Promise} Array of bookings
 */
export const getBookings = async () => {
  try {
    const response = await api.get('/bookings');
    return response.data;
  } catch (error) {
    console.error('Error fetching bookings:', error);
    throw error;
  }
};

/**
 * Get a single booking by ID
 * @param {number} id - Booking ID
 * @returns {Promise} Booking object
 */
export const getBookingById = async (id) => {
  try {
    const response = await api.get(`/bookings/${id}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching booking:', error);
    throw error;
  }
};

/**
 * Create a new booking
 * @param {Object} bookingData - Booking data (roomId, checkIn, checkOut
 * @returns {Promise} Created booking object
 */
export const createBooking = async (bookingData) => {
  try {
    const response = await api.post('/bookings', bookingData);
    return response.data;
```

```
/**
 * User Signup
 * @param {Object} userData - User data (name, email, password)
 * @returns {Promise} Created user object
 */
export const signupUser = async (userData) => {
  try {
    const response = await api.get(`/users?email=${userData.email}`);
    if (response.data.length > 0) {
      throw new Error('User with this email already exists');
    }

    const newUser = await api.post('/users', userData);
    return newUser.data;
  } catch (error) {
    console.error('Error signing up:', error);
    throw error;
  }
};

/**
 * User Login
 * @param {string} email - User email
 * @param {string} password - User password
 * @returns {Promise} User object if credentials are valid
 */
export const loginUser = async (email, password) => {
  try {
    const response = await api.get(`/users?email=${email}&password=${password}`);
    if (response.data.length === 0) {
      throw new Error('Invalid email or password');
    }
    return response.data[0];
  } catch (error) {
    console.error('Error logging in:', error);
    throw error;
  }
};

/**
```

```
/**
 * Admin Login
 * @param {string} username - Admin username
 * @param {string} password - Admin password
 * @returns {Promise} Admin object if credentials are valid
 */
export const loginAdmin = async (username, password) => {
  try {
    const response = await api.get(`/admins?username=${username}&password=${password}`);
    if (response.data.length === 0) {
      throw new Error('Invalid username or password');
    }
    return response.data[0];
  } catch (error) {
    console.error('Error logging in as admin:', error);
    throw error;
  }
};

export default api;
```

Code Description:-

- This file configures Axios with a base URL (http://localhost:3000) to communicate with the backend API.
- Provides functions to fetch all rooms and fetch a room by ID.
- Includes admin functions to create, update, and delete rooms.
- Handles booking operations such as getting bookings, creating bookings, updating, and deleting bookings.
- Adds user authentication features including signup and login functions.
- Signup checks if a user with the same email already exists before creating a new account.
- Login validates user credentials by searching for matching email and password.
- Includes admin login to validate admin credentials using the admins database.
- Uses try...catch in every API call to safely handle errors and show meaningful messages.

- Exports all API functions so they can be used across the hotel booking system.

Navbar Componet :-

```
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

/**
 * Navbar Component
 * Navigation bar with authentication-aware links
 */
const Navbar = () => {
  const { user, admin, isAuthenticated, isAdmin, logout } = useAuth();
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate('/');
  };

  return (
    <nav className="bg-gradient-to-r from-blue-600 to-purple-600 text-white
shadow-lg">
      <div className="container mx-auto px-4">
        <div className="flex items-center justify-between h-16">
          {/* Logo */}
          <Link to="/" className="text-2xl font-bold hover:text-gray-200
transition">
            🏨 Hotel Booking
          </Link>

          {/* Navigation Links */}
          <div className="flex items-center space-x-6">
            <Link
              to="/"
              className="hover:text-gray-200 transition font-medium"
            >
              Home
            </Link>
            <Link
              to="/rooms"
              className="hover:text-gray-200 transition font-medium"
```

```
            >
              Rooms
            </Link>

            {/* Show based on authentication */}
            {isAuthenticated && (
              <>
                <Link
                  to="/bookings"
                  className="hover:text-gray-200 transition font-medium"
                >
                  My Bookings
                </Link>
                <Link
                  to="/profile"
                  className="hover:text-gray-200 transition font-medium"
                >
                  Profile
                </Link>
              </>
            )}

            {isAdmin && (
              <>
                <Link
                  to="/admin"
                  className="hover:text-gray-200 transition font-medium bg-
white/20 px-3 py-1 rounded-md"
                >
                  Admin Panel
                </Link>
                <Link
                  to="/profile"
                  className="hover:text-gray-200 transition font-medium"
                >
                  Profile
                </Link>
              </>
            )}

            {/* Authentication Buttons */}
            {!isAuthenticated && !isAdmin ? (
              <div className="flex items-center space-x-3">
                <Link
                  to="/login"
```

```
                  className="hover:text-gray-200 transition font-medium"
                >
                  Login
                </Link>
                <Link
                  to="/signup"
                  className="bg-white text-blue-600 px-4 py-2 rounded-lg font-
medium hover:bg-gray-100 transition"
                >
                  Sign Up
                </Link>
              </div>
            ) : (
              <div className="flex items-center space-x-4">
                {/* User/Admin Info - Clickable to go to profile */}
                <Link to="/profile" className="flex items-center space-x-2
hover:bg-white/10 px-3 py-1 rounded-lg transition">
                  <div className="w-8 h-8 bg-white/20 rounded-full flex items-
center justify-center">
                    {isAdmin ? (
                      <svg className="w-5 h-5" fill="currentColor" viewBox="0 0
20 20">
                        <path fillRule="evenodd" d="M10 9a3 3 0 100-6 3 3 0 000
6zm-7 9a7 7 0 1114 0H3z" clipRule="evenodd" />
                      </svg>
                    ) : (
                      <svg className="w-5 h-5" fill="currentColor" viewBox="0 0
20 20">
                        <path fillRule="evenodd" d="M10 9a3 3 0 100-6 3 3 0 000
6zm-7 9a7 7 0 1114 0H3z" clipRule="evenodd" />
                      </svg>
                    )}
                  </div>
                  <span className="text-sm font-medium">
                    {isAdmin ? admin?.name : user?.name}
                    {isAdmin && <span className="ml-1 text-xs">(Admin)</span>}
                  </span>
                </Link>

                {/* Logout Button */}
                <button
                  onClick={handleLogout}
                  className="bg-white/20 hover:bg-white/30 px-4 py-2 rounded-lg
font-medium transition flex items-center space-x-2"
                >
```

```
                  <svg className="w-4 h-4" fill="none" stroke="currentColor"
viewBox="0 0 24 24">
                    <path strokeLinecap="round" strokeLinejoin="round"
strokeWidth="2" d="M17 16l4-4m0 0l-4-4m4 4H7m6 4v1a3 3 0 01-3 3H6a3 3 0 01-3-
3V7a3 3 0 013-3h4a3 3 0 013 3v1" />
                  </svg>
                  <span>Logout</span>
                </button>
              </div>
            )}
          </div>
        </div>
      </div>
    </nav>
  );
};

export default Navbar;
```

Code Description:-

- The Navbar component provides the main navigation bar for the Hotel Booking System.
- It uses **React Router** to navigate between pages like Home, Rooms, Bookings, Profile, Admin Panel, Login, and Signup.
- The component listens to authentication status using **useAuth()** to dynamically show different menus.
- If a user is logged in, links like **My Bookings** and **Profile** become visible.
- If an admin logs in, additional options like **Admin Panel** appear.
- If no one is logged in, **Login** and **Signup** buttons are shown.
- The navbar also includes a **Logout** button that logs out the user/admin and redirects to home.
- The design uses Tailwind CSS with a modern gradient background for an attractive UI.

- User/admin information like their name is displayed in the top-right profile section.
- Overall, the Navbar ensures smooth navigation and personalized menus based on login status.

PrivateRoute Component:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

/**
 * PrivateRoute Component
 * Protects routes that require authentication
 */
export const PrivateRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-
blue-600 mx-auto"></div>
          <p className="mt-4 text-gray-600">Loading...</p>
        </div>
      </div>
    );
  }

  return isAuthenticated ? children : <Navigate to="/login" />;
};

/**
 * AdminRoute Component
 * Protects routes that require admin authentication
 */
export const AdminRoute = ({ children }) => {
  const { isAdmin, loading } = useAuth();

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
```

```
          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-
purple-600 mx-auto"></div>
          <p className="mt-4 text-gray-600">Loading...</p>
        </div>
      </div>
    );
  }

  return isAdmin ? children : <Navigate to="/admin-login" />;
};

/**
 * AuthenticatedRoute Component
 * Protects routes that require either user or admin authentication
 */
export const AuthenticatedRoute = ({ children }) => {
  const { isAuthenticated, isAdmin, loading } = useAuth();

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-
blue-600 mx-auto"></div>
          <p className="mt-4 text-gray-600">Loading...</p>
        </div>
      </div>
    );
  }

  return isAuthenticated || isAdmin ? children : <Navigate to="/login" />;
};
```

Code Description:-

- This file defines three special route protection components for the hotel booking system.
- **PrivateRoute** restricts access to pages that only logged-in users should see (like booking pages).
- **AdminRoute** allows only admin users to access admin-level pages such as managing rooms and bookings.

- **AuthenticatedRoute** allows access to both **logged-in users and admins**, used for shared pages like profile or bookings list.
- Each route checks user authentication status using the **AuthContext**.
- While loading authentication state, a loading spinner is shown to avoid UI errors.
- If the user is not allowed, they are automatically redirected to **login** or **admin-login** pages.
- This ensures secure navigation and prevents unauthorized access throughout the application.

RoomCard Component :-

```
import { Link } from 'react-router-dom';

/**
 * RoomCard Component
 * Displays a room card with image, details, and action button
 * @param {Object} room - Room object with id, name, type, price, capacity,
image, description
 */
const RoomCard = ({ room }) => {
  return (
    <div className="bg-white rounded-lg shadow-md overflow-hidden hover:shadow-xl
transition-shadow duration-300">
      {/* Room Image */}
      <img
        src={room.image}
        alt={room.name}
        className="w-full h-48 object-cover"
      />

      {/* Room Details */}
      <div className="p-4">
        <div className="flex justify-between items-start mb-2">
          <h3 className="text-xl font-bold text-gray-800">{room.name}</h3>
          <span className="bg-blue-100 text-blue-800 text-xs font-semibold px-2
py-1 rounded">
            {room.type}
          </span>
        </div>

        <p className="text-gray-600 text-sm mb-3 line-clamp-2">
          {room.description}
        </p>

        <div className="flex justify-between items-center">
```

```
          <div>
            <p className="text-2xl font-bold text-blue-600">₹{room.price}</p>
            <p className="text-xs text-gray-500">per night</p>
          </div>
          <div className="text-right">
            <p className="text-sm text-gray-600">Capacity: {room.capacity}</p>
          </div>
        </div>

        {/* View Details Button */}
        <Link
          to={`/rooms/${room.id}`}
          className="mt-4 w-full block text-center bg-blue-600 text-white py-2
rounded-lg hover:bg-blue-700 transition font-medium"
        >
          View Details
        </Link>
      </div>
    </div>
  );
};

export default RoomCard;
```

Code Description:-

- This component displays a single hotel room in card format.
- It shows the room's image, name, type, price, capacity, and a short description.
- The design includes Tailwind CSS styles for layout, shadows, spacing, and hover effects.
- It neatly organizes room details with bold titles and highlighted room type badges.
- The price and capacity are shown clearly to help users compare rooms.
- A **"View Details"** button links to the full room details page using React Router.

- This component is reusable and is used on pages like **Home** and **Rooms** to list multiple rooms.

AuthContext Contexts:

```
import { createContext, useContext, useState, useEffect } from 'react';

const AuthContext = createContext(null);

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [admin, setAdmin] = useState(null);
  const [loading, setLoading] = useState(true);

  // Load user from localStorage on mount
  useEffect(() => {
    const storedUser = localStorage.getItem('user');
    const storedAdmin = localStorage.getItem('admin');

    if (storedUser) {
      setUser(JSON.parse(storedUser));
    }
    if (storedAdmin) {
      setAdmin(JSON.parse(storedAdmin));
    }
    setLoading(false);
  }, []);

  const login = (userData) => {
    setUser(userData);
    localStorage.setItem('user', JSON.stringify(userData));
  };

  const adminLogin = (adminData) => {
    setAdmin(adminData);
```

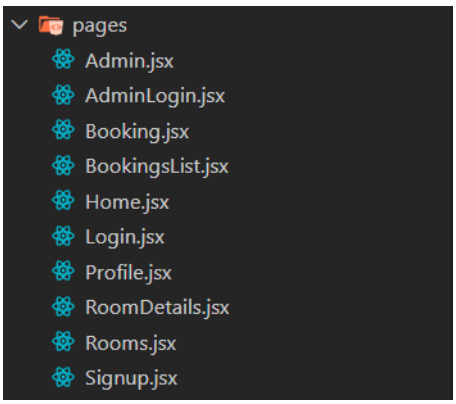
```
    localStorage.setItem('admin', JSON.stringify(adminData));
  };

  const logout = () => {
    setUser(null);
    setAdmin(null);
    localStorage.removeItem('user');
    localStorage.removeItem('admin');
  };

  const value = {
    user,
    admin,
    loading,
    login,
    adminLogin,
    logout,
    isAuthenticated: !!user,
    isAdmin: !!admin,
  };

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};

export default AuthContext;
```

Code Description:-

- This file creates a **global authentication system** using React Context API.
- AuthProvider stores and manages login data for both **users** and **admins**.
- It loads saved login information from **localStorage** when the app starts.
- It provides functions like **login**, **adminLogin**, and **logout** to update authentication state.
- When a user or admin logs in, their info is saved in localStorage so it persists on refresh.
- isAuthenticated becomes true when a normal user is logged in.
- isAdmin becomes true when an admin is logged in.

- Any component can access the authentication state using **useAuth()**.
- It makes it easy to protect routes and show different UI for users and admins.

Pages:



Admin.jsx

Code Description

· This file creates the **admin dashboard** for managing hotel room details.

· It allows the admin to **add new rooms, edit existing rooms, and delete rooms**.

· It fetches room data from the backend when the page loads.

· All room details are stored and managed using React state.

· A loading indicator is displayed until room data is fully fetched.

· It displays all rooms in a **table format** with image, name, type, price, and capacity.

· When the Add Room button is clicked, a form is displayed for entering room details.

· The same form is used for both **adding and editing** room information.

· When editing, the selected room details are automatically filled in the form.

· Amenities are entered as a comma-separated list and stored as an array.

· Room price is converted into a number before saving.

· Guest capacity is converted into an integer.

· After submitting the form, the room list updates instantly without refreshing the page.

· When delete is clicked, a confirmation popup is shown before removing the room.

· Removed rooms are deleted from both the interface and the database.

· A navigation link allows the admin to move to the booking management page.

· The page uses a responsive layout for better viewing on all screen sizes.

· It connects with backend APIs for all create, update, delete, and fetch operations.

· It ensures smooth room management for the hotel system.

· It helps the admin maintain accurate pricing, capacity, and availability information.

· The component improves overall hotel efficiency by centralizing room management.

AdminLogin.jsx

Code Description

· This file creates the **admin authentication page** for the hotel booking management system.

· It provides a secure login interface only for administrators.

· It uses React state to store the entered username and password.

· It clears previous error messages when the user types new input.

· It sends the admin login details to the backend for verification.

· If login is successful, the admin data is saved in the authentication system.

· After a successful login, the admin is redirected to the admin dashboard.

· If login fails, an error message is displayed on the screen.

· A loading indicator appears while the system verifies login credentials.

· The login button becomes disabled while authentication is in progress.

· The page displays a professional UI suitable for an admin portal.

· The design includes a protected login form with a password field.

· A back navigation option is provided to return to the user login page.

· It supports role-based access by separating admin login from user login.

· It integrates with the global authentication context for managing admin sessions.

· The login state remains active after page refresh.

· It helps in preventing unauthorized access to admin features.

· The file improves system security by enforcing admin-level authentication.

· It provides a demo admin account for testing and development purposes.

Booking.jsx – Code Description

· This file creates the **room booking page** for the hotel management system.

· It allows users to book a selected room by entering guest details and stay dates.

· The room ID is taken from the URL to identify which room is being booked.

· Room details are fetched from the backend when the page loads.

· A loading animation is displayed while room information is being retrieved.

· If the room does not exist, an error message is shown.

· The currently logged-in user is linked with the booking automatically.

· User input is stored using React state.

· The form collects:

- Guest name
- Check-in date
- Check-out date

· The system validates user input before creating the booking.

· It prevents booking if any field is empty.

· It ensures the check-out date is always after the check-in date.

· The booking is submitted to the backend only after validation passes.

· A loading state is shown while the booking is being processed.

· On successful booking, a success message is displayed.

· The user is automatically redirected to the bookings page after confirmation.

· If booking fails, an error message is displayed.

· The user can return to the room details page using a back button.

· A summary of the room is shown including:

- Room name
- Room type
- Price per night
- Capacity

· The booking button is disabled while submission is in progress.

· The page layout is responsive and user-friendly.

· This page ensures only authenticated users can make reservations.

· It integrates with backend APIs for:

- Fetching room details
- Creating bookings

· It improves user experience by giving real-time validation and feedback.

· It helps maintain accurate booking records linked to users.

· This component plays a key role in the hotel reservation workflow.

BookingsList.jsx

Code Description

· This file creates the **Bookings List Page** of the application.

· It displays all room bookings in a structured card layout.

· Admin users can view **all bookings in the system**.

· Normal users can only see **their own bookings**.

· The logged-in user and role are obtained using the authentication context.

· When the page loads, the system automatically fetches booking data from the backend.

· A loading animation is shown while booking details are being loaded.

· If an error occurs while fetching data, an error message is displayed.

· Each booking includes:

- Guest name
- Check-in date
- Check-out date
- Room name
- Room price
- Booking ID

· Room details are loaded separately for each booking using the room ID.

· If room details cannot be loaded, a fallback message is shown.

· Users and admins can **delete bookings**.

· A confirmation popup is shown before deletion.

· After deleting, the booking is instantly removed from the UI.

· Users and admins can **edit existing bookings**.

· Clicking the edit button opens the booking in edit mode.

· The edit form allows:

- Updating guest name
- Changing check-in date
- Changing check-out date

· Users can save the updated booking.

· On successful update:

- The booking list refreshes instantly.
- Edit mode is closed automatically.

· Users can cancel editing anytime.

· Admin users also see additional:

- User ID of the person who made the booking
- Admin badge indicating admin mode

· When no bookings are available:

- Admin sees “No bookings in the system yet.”
- User sees “You have no bookings yet.”

· Date values are formatted into readable form.

· Booking prices are shown in Indian Rupees.

· The interface uses modern UI styling for better experience.

· The layout is fully responsive.

· Buttons provide clear actions like edit, delete, and save.

· UI changes immediately reflect backend updates.

· This page improves usability by allowing direct:

- Booking management
- Editing
- Deletion

· It supports role-based access control.

· It ensures users only access their permitted data.

· This component acts as a management hub for reservations.

· It is tightly integrated with backend APIs for real-time data.

· It enhances user control and administrative efficiency.

· This page completes the user booking lifecycle.

Home.jsx – Code Description

· This file creates the **home page** of the hotel booking application.

· It displays a welcome banner and featured rooms.

· This page acts as the **landing page** when the website is opened.

· A hero section is shown on top with:

- A welcome message
- A short brand tagline
- A "Browse All Rooms" button

· The browse button redirects users to the rooms page.

· When the page loads, room data is fetched from the backend.

· Only the first three rooms are selected as featured rooms.

· A loading animation is displayed while rooms are being fetched.

· If rooms fail to load, the error is logged in the console.

· Each room is displayed using a reusable room card component.

· If no rooms are available, a message is shown.

· Featured rooms are displayed in a grid format.

· The layout is responsive for all screen sizes.

· The design uses gradient colors for the hero section.

· A “Why Choose Us” section highlights services including:

- Premium quality
- Best prices
- Excellent support

· Emoji icons are used for better visual experience.

· The UI focuses on user clarity and simplicity.

· This page encourages users to explore bookings.

· The layout improves user engagement.

· It provides branding and marketing value.

· The component updates automatically on refresh.

· The page works as the entry point to the website.

· This file integrates with backend APIs.

· It enhances user trust through visual design.

· It gives users a smooth onboarding experience.

· This page links navigation and content flow.

· It improves conversion by highlighting offers.

Login.jsx

Code Description

· This file creates the **user login page** for the application.
· It uses state to store email, password, loading, and error messages.
· When the user submits the form, the login API is called for authentication.
· On successful login, user details are saved using the AuthContext and the user is redirected to the homepage.
· If login fails, an error message is shown.
· The UI includes input fields, a submit button, demo credentials, and links for signup and admin login.
· A loading spinner appears while processing the login request.
· The layout is clean, responsive, and user-friendly.

Profile.jsx

Code Description

· This file displays the **user or admin profile page** based on login role.
· It retrieves user/admin details from the authentication context.
· Users can view and edit their personal information.
· Admins see their username, while users see their email.
· Logout functionality is provided to end the session.
· Profile edit mode allows name, email, or username updates (demo only).
· Quick navigation buttons link to bookings, rooms, and admin panel.
· Displays basic account details and statistics for users.

RoomDetails.jsx

Code Description

· This file displays **detailed information for a selected room**.
· It fetches room data using the room ID from the URL.
· A loading spinner is shown while data is being loaded.
· If the room is not found, an error message is displayed.
· Room image, type, price, capacity, and description are shown.
· Amenities are displayed in a clean grid format.
· A **Book Now** button redirects the user to the booking page.
· A back button allows navigation to the rooms list.

Rooms.jsx

Code Description

· This file displays a list of **all available hotel rooms**.
· It fetches room data from the backend API when the page loads.
· Loading state shows a spinner while rooms are being retrieved.
· If an error occurs, an error message is displayed to the user.
· Each room is rendered using the `RoomCard` component.
· If no rooms are found, a message is shown instead of the grid.
· The layout uses a responsive grid to adapt to screen size.
· The page updates automatically after rooms are loaded.
· The design keeps room browsing simple and user-friendly.

SignUp.jsx

Code Description

• This component creates a user registration form using React.
• It uses `useState` to manage user input, loading state, and error messages.
• It validates password match and minimum length before submitting.
• User data is sent to the backend using `signupUser()` API function.
• On successful signup, the user is automatically logged in using `login()`.
• `useNavigate()` redirects the user to the home page after registration.
• Error messages are shown if signup fails or validation fails.
• Loading animation appears while the account is being created.
• Provides link to login page for existing users.

Db.json

Code Description

• This file acts as a local database for the hotel booking system.
• It stores room details like price, type, capacity, and amenities.
• All booking records with guest name and dates are maintained here.
• User login details are stored under the `users` section.
• Admin login credentials are stored separately in the `admins` section.
• Rooms are linked with bookings using `roomId`.
• Helps the frontend display rooms and manage reservations easily.
• Used as a mock backend for testing and development.

App.js

Code Description

- Wraps the app with **`AuthProvider`** to manage authentication state.
- Uses **React Router (`BrowserRouter`)** to handle page navigation.
- **Public Routes:** Home, Login, Signup, Admin Login, Rooms, Room Details.
- **Protected User Routes (Private):** Booking page, Bookings list, User Profile.
- **Protected Admin Route:** Admin dashboard.
- **Route Guards:**
  - `PrivateRoute` → for logged-in users only.
  - `AuthenticatedRoute` → for any authenticated user.
  - `AdminRoute` → for admin users only.
- Navbar is displayed across all pages for easy navigation.

index.js

Code Description

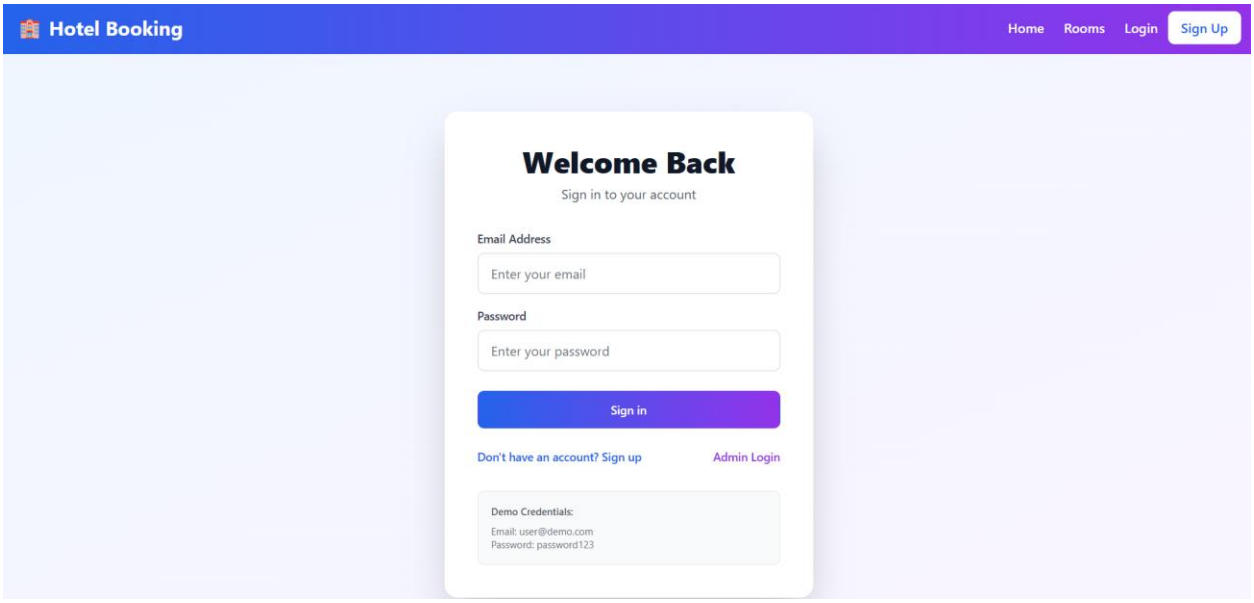
· This file is the entry point of the React application.
· It imports global CSS styles and the main `App` component.
· Uses `createRoot` from `react-dom/client` to attach the app to the DOM element with id `root`.
· Wraps the app in `StrictMode` to help detect potential issues during development.
· Renders the `App` component, which contains all routes, pages, and context providers.

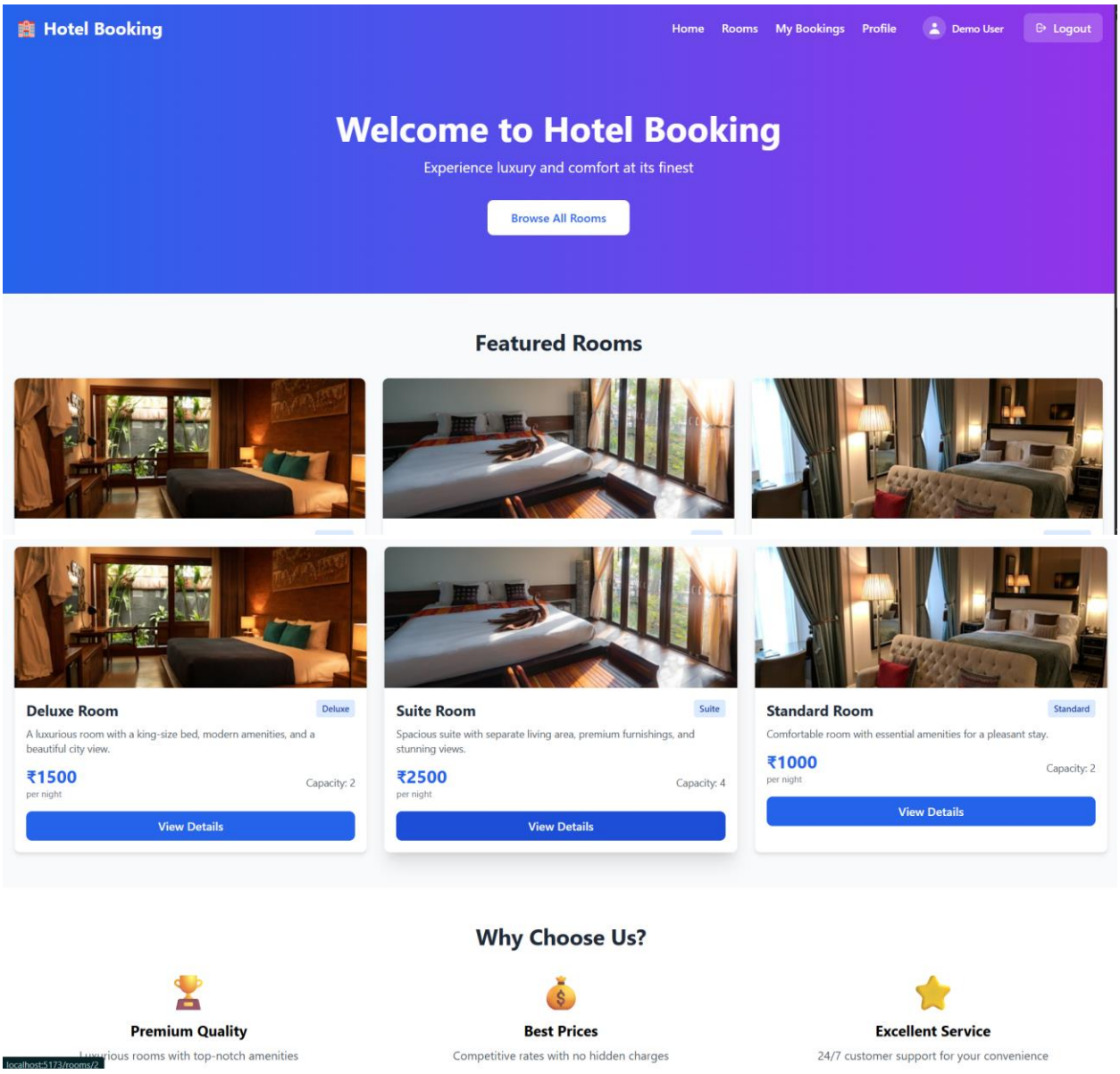
index.css

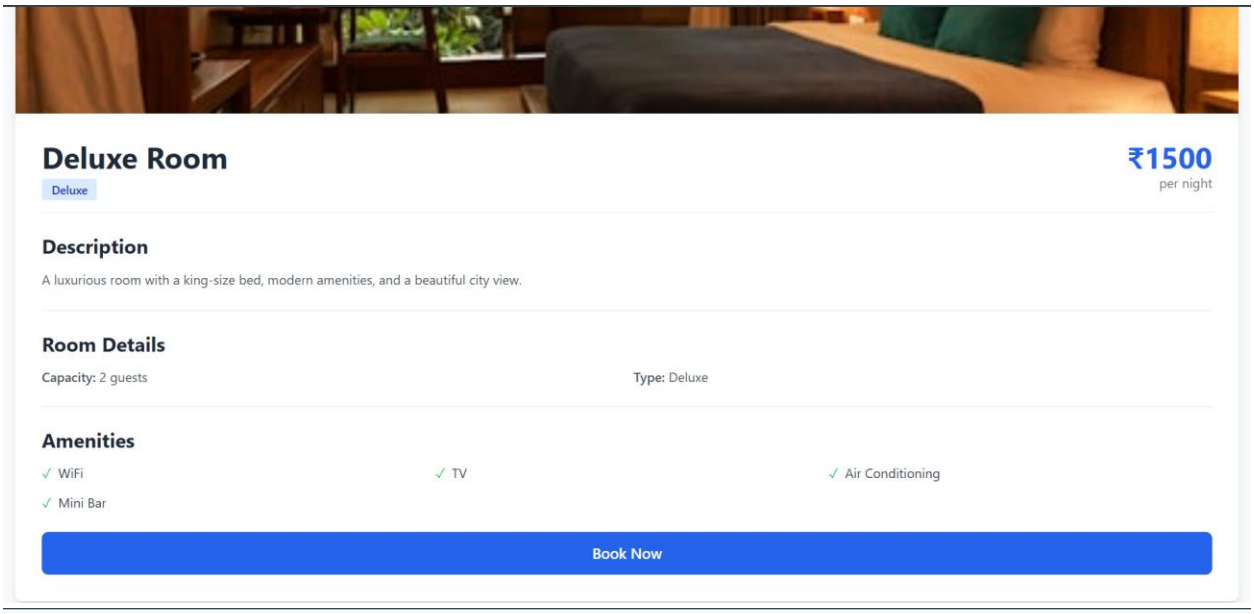
Code Description

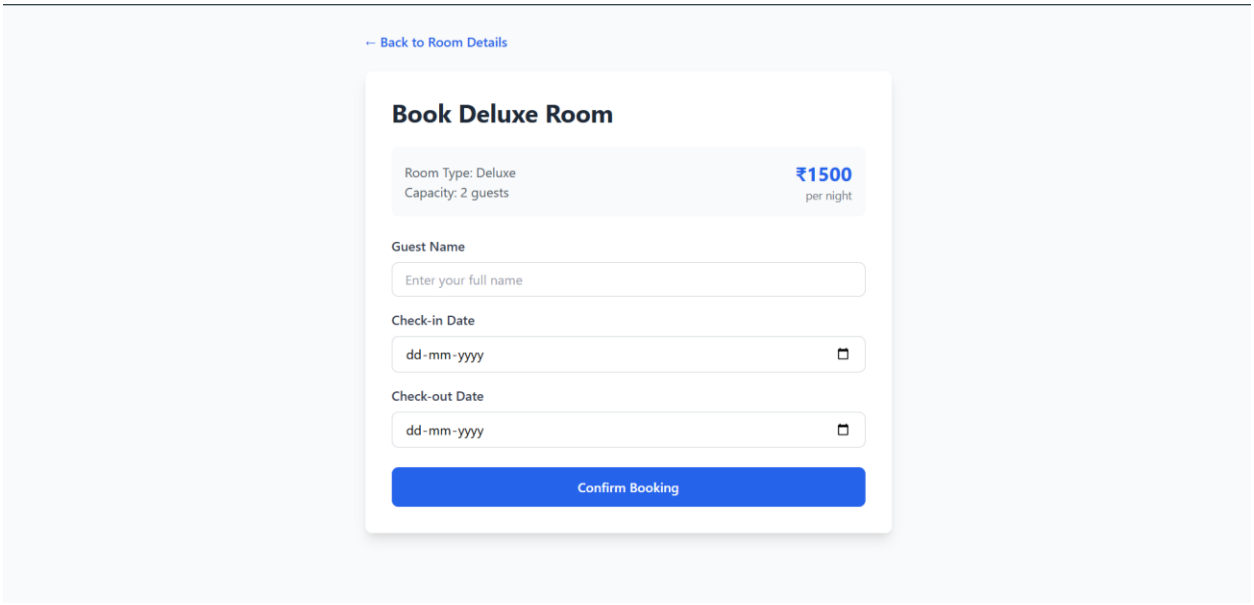
· Sets the maximum width, centering, padding, and text alignment for the `#root` container.
· Styles `.logo` images with height, padding, and smooth filter transitions on hover.

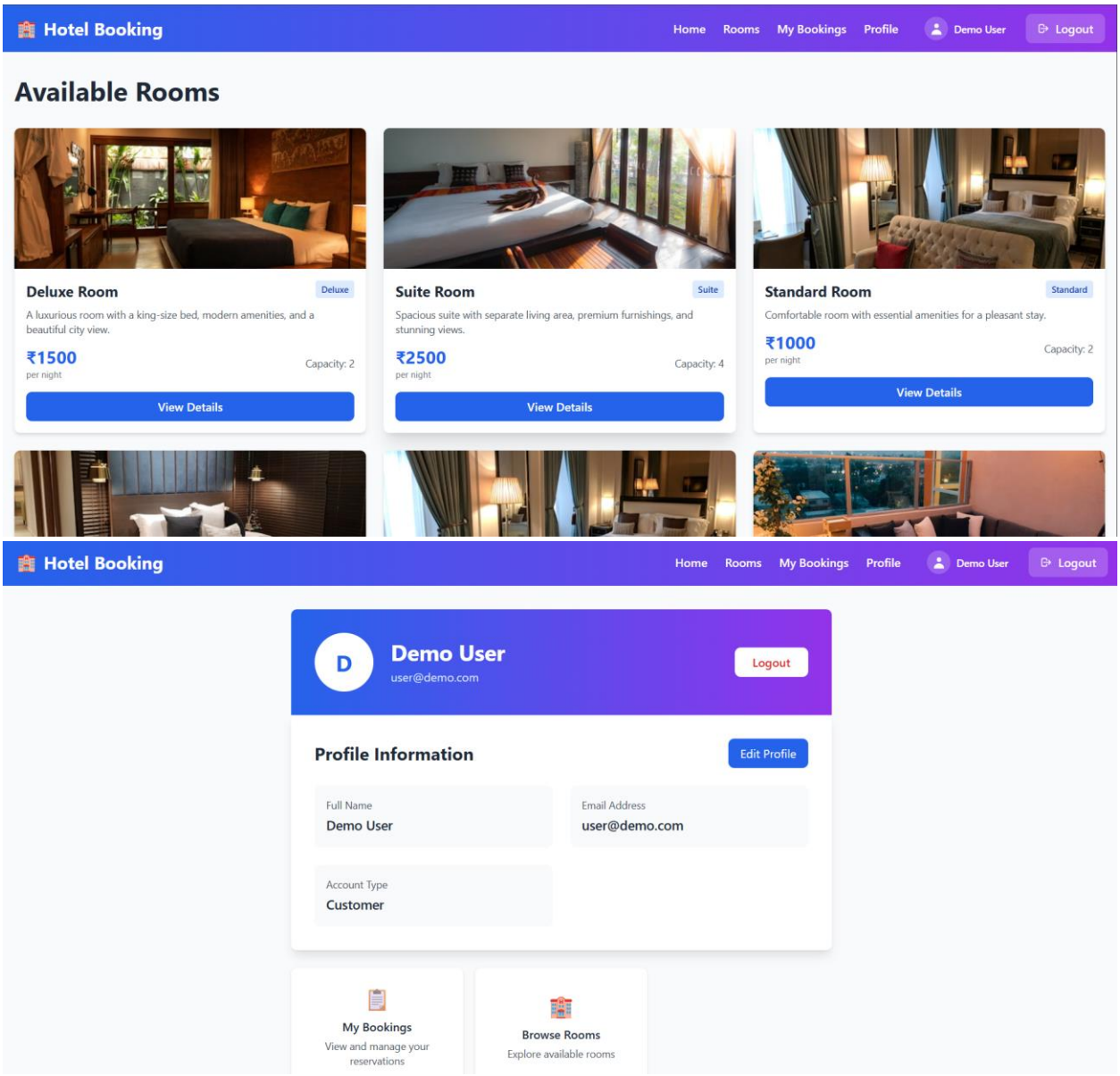
· Adds specific hover effects for logos, including a drop-shadow with color variations for React logos.
· Defines a `@keyframes` animation `logo-spin` for rotating elements from 0° to 360°.
· Uses a media query `prefers-reduced-motion: no-preference` to apply the spinning animation to the second logo element only if the user allows animations.
· `.card` class adds padding for content sections.
· `.read-the-docs` class sets text color for documentation links or notes.

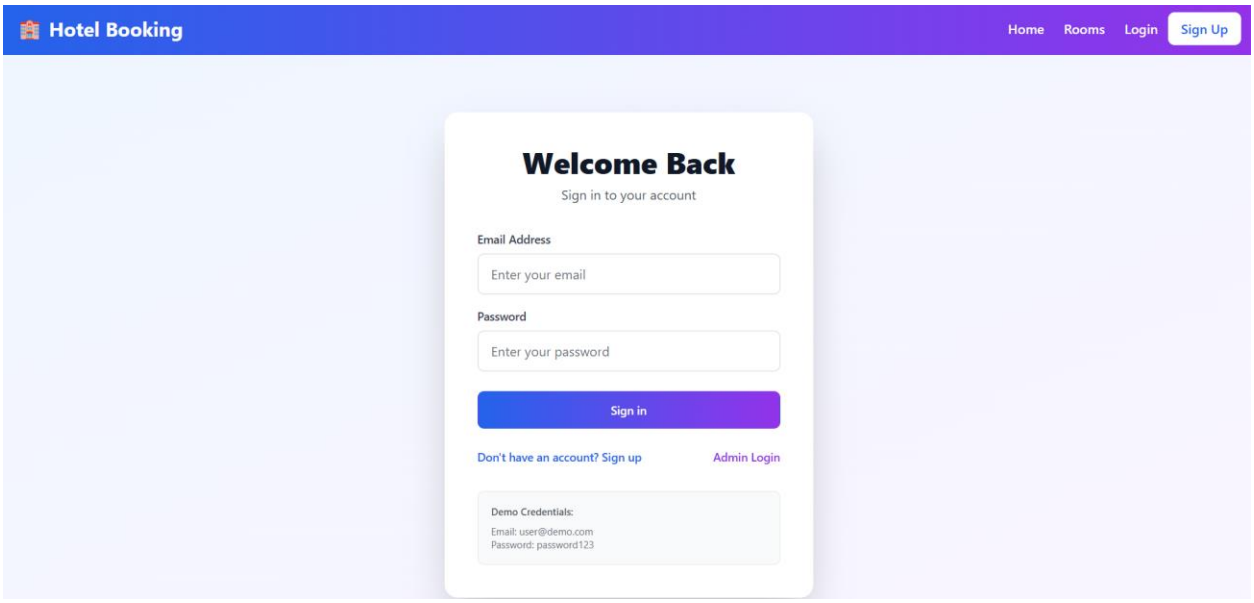


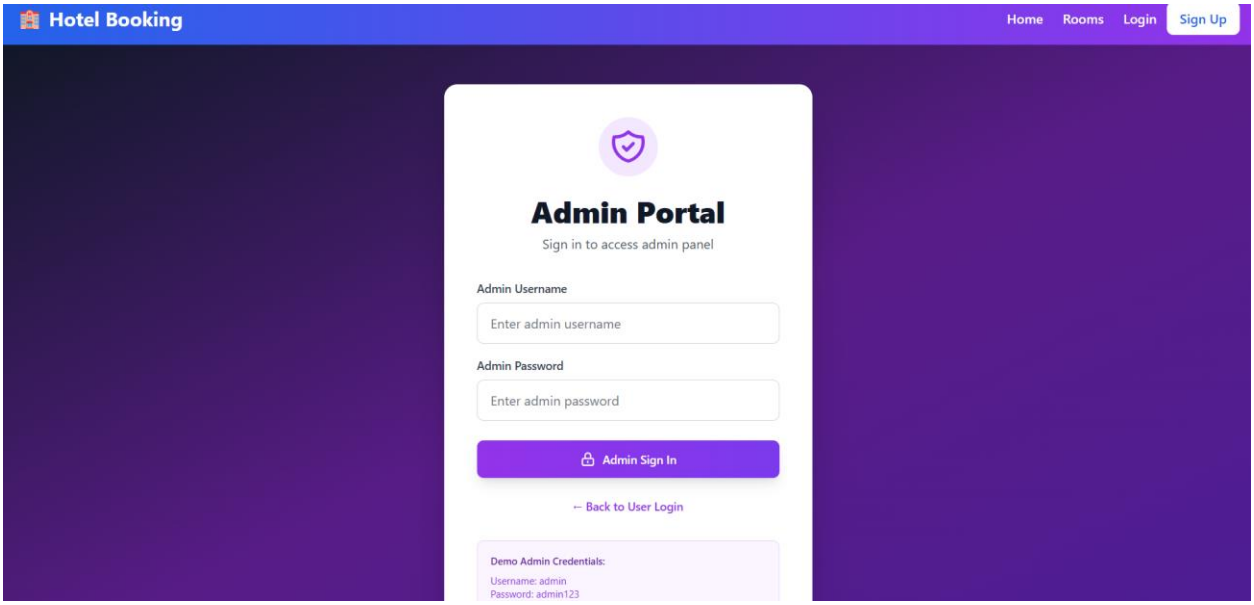

Hotel Booking
Home
Rooms
My Bookings
Profile
Demo User
Logout
Welcome to Hotel Booking
Experience luxury and comfort at its finest
Browse All Rooms
Featured Rooms
Deluxe Room
Deluxe
A luxurious room with a king-size bed, modern amenities, and a beautiful city view.
₹1500
per night
Capacity: 2
View Details
Suite Room
Suite
Spacious suite with separate living area, premium furnishings, and stunning views.
₹2500
per night
Capacity: 4
View Details
Standard Room
Standard
Comfortable room with essential amenities for a pleasant stay.
₹1000
per night
Capacity: 2
View Details
Why Choose Us?
Premium Quality
Luxurious rooms with top-notch amenities
Best Prices
Competitive rates with no hidden charges
Excellent Service
24/7 customer support for your convenience
localhost:5173/rooms/2

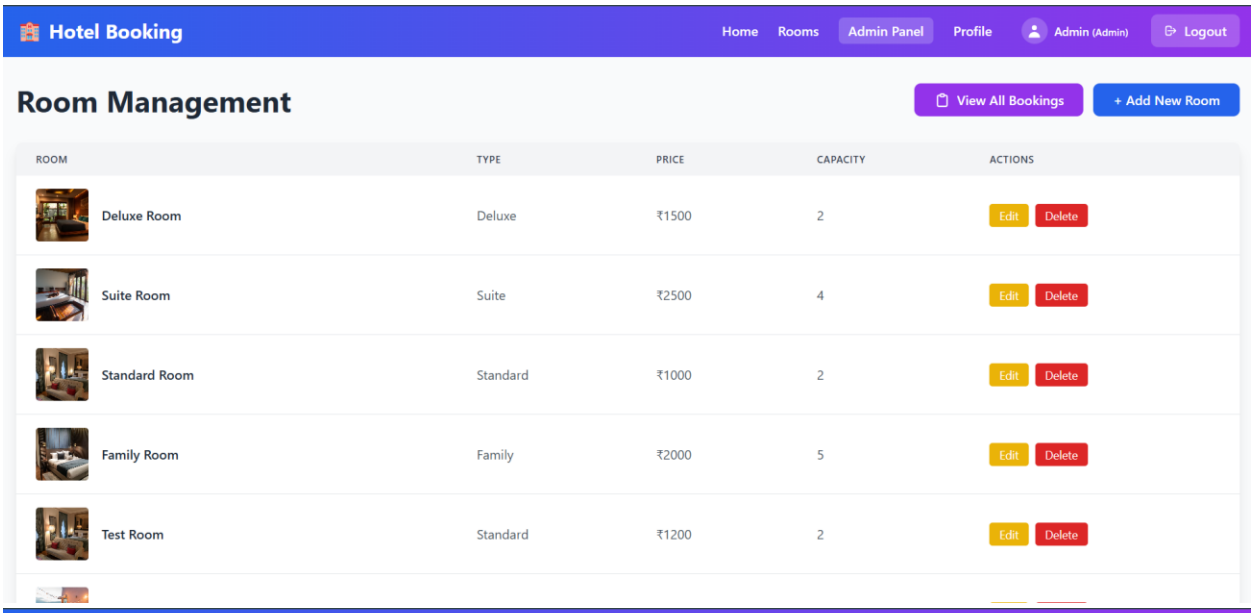

Deluxe Room
Deluxe
₹1500
per night
Description
A luxurious room with a king-size bed, modern amenities, and a beautiful city view.
Room Details
Capacity: 2 guests
Type: Deluxe
Amenities
✓ WiFi
✓ TV
✓ Air Conditioning
✓ Mini Bar
Book Now

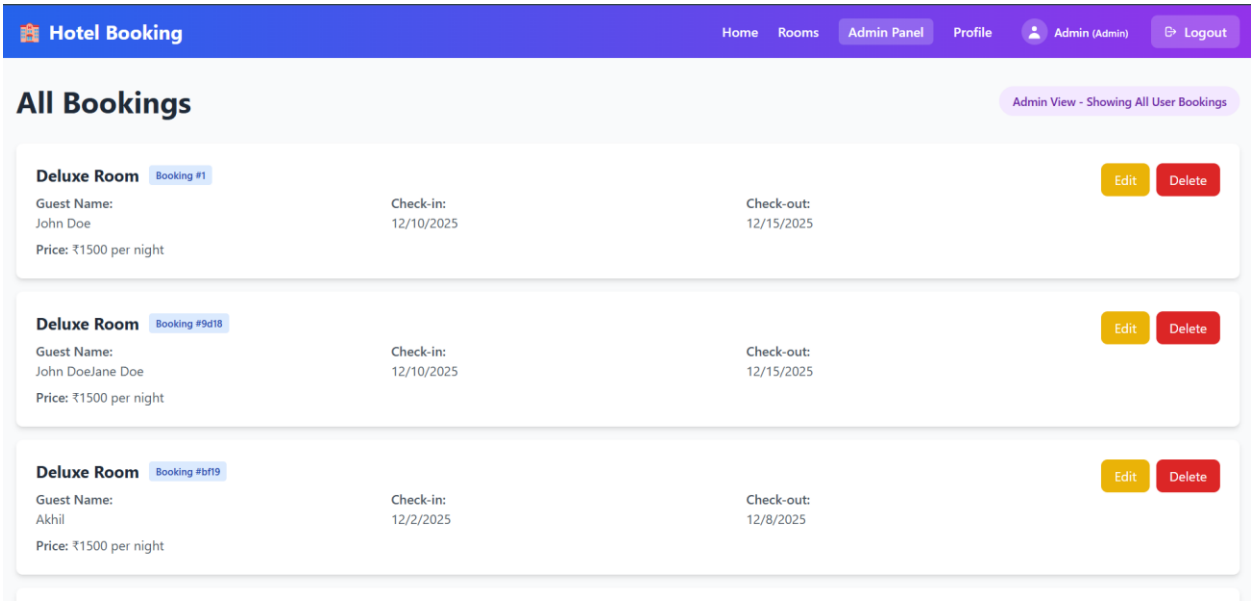

← Back to Room Details
Book Deluxe Room
Room Type: Deluxe
Capacity: 2 guests
₹1500
per night
Guest Name
Enter your full name
Check-in Date
dd-mm-yyyy
Check-out Date
dd-mm-yyyy
Confirm Booking

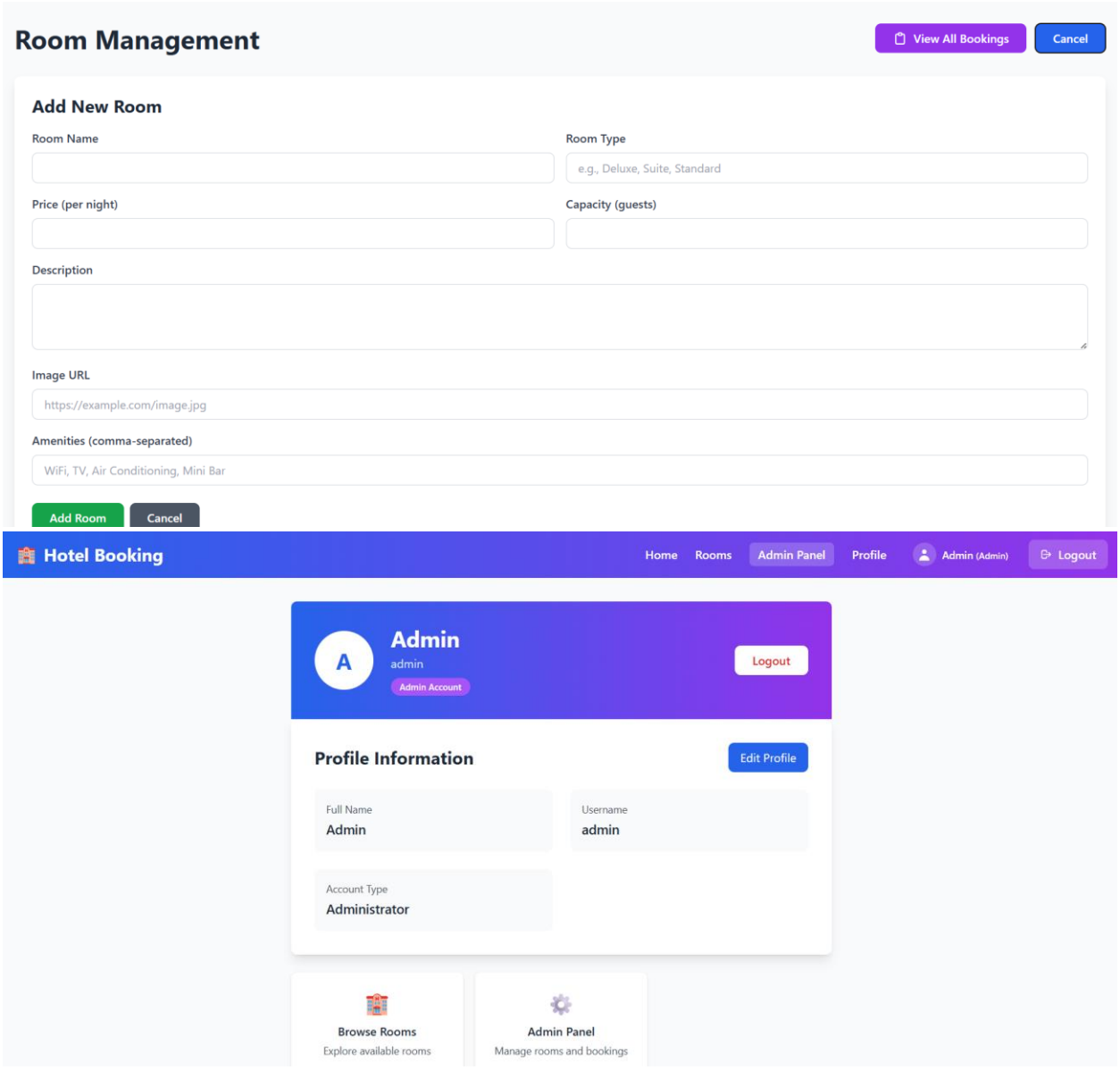

Hotel Booking
Home
Rooms
My Bookings
Profile
Demo User
Logout
Available Rooms
Deluxe Room
Deluxe
A luxurious room with a king-size bed, modern amenities, and a beautiful city view.
₹1500
per night
Capacity: 2
View Details
Suite Room
Suite
Spacious suite with separate living area, premium furnishings, and stunning views.
₹2500
per night
Capacity: 4
View Details
Standard Room
Standard
Comfortable room with essential amenities for a pleasant stay.
₹1000
per night
Capacity: 2
View Details
Hotel Booking
Home
Rooms
My Bookings
Profile
Demo User
Logout
D
Demo User
user@demo.com
Logout
Profile Information
Edit Profile
Full Name
Demo User
Email Address
user@demo.com
Account Type
Customer
My Bookings
View and manage your reservations
Browse Rooms
Explore available rooms


Hotel Booking
Home
Rooms
Login
Sign Up
Welcome Back
Sign in to your account
Email Address
Enter your email
Password
Enter your password
Sign in
Don't have an account? Sign up
Admin Login
Demo Credentials:
Email: user@demo.com
Password: password123


Hotel Booking
Home
Rooms
Login
Sign Up
Admin Portal
Sign in to access admin panel
Admin Username
Enter admin username
Admin Password
Enter admin password
Admin Sign In
← Back to User Login
Demo Admin Credentials:
Username: admin
Password: admin123


Hotel Booking
Home
Rooms
Admin Panel
Profile
Admin (Admin)
Logout
Room Management
View All Bookings
+ Add New Room
ROOM
TYPE
PRICE
CAPACITY
ACTIONS
Deluxe Room
Deluxe
₹1500
2
Edit
Delete
Suite Room
Suite
₹2500
4
Edit
Delete
Standard Room
Standard
₹1000
2
Edit
Delete
Family Room
Family
₹2000
5
Edit
Delete
Test Room
Standard
₹1200
2
Edit
Delete


Hotel Booking
Home
Rooms
Admin Panel
Profile
Admin (Admin)
Logout
All Bookings
Admin View - Showing All User Bookings
Deluxe Room
Booking #1
Guest Name:
John Doe
Check-in:
12/10/2025
Check-out:
12/15/2025
Price: ₹1500 per night
Edit
Delete
Deluxe Room
Booking #9d18
Guest Name:
John DoeJane Doe
Check-in:
12/10/2025
Check-out:
12/15/2025
Price: ₹1500 per night
Edit
Delete
Deluxe Room
Booking #bf19
Guest Name:
Akhil
Check-in:
12/2/2025
Check-out:
12/8/2025
Price: ₹1500 per night
Edit
Delete



Project demo:

Demo link: https://drive.google.com/file/d/1MO3sckNQaEGTAuFQeuz1SNHNtasOtNkz/view?usp=sharing