|  | **SRI KRISHNA COLLEGE OF TECHNOLOGY**  **(An Autonomous Institution)**  **Approved by AICTE | Affiliated to Anna University Chennai|**  **Accredited by NBA - AICTE| Accredited by NAAC with ‘A’ Grade**  **KOVAIPUDUR, COIMBATORE 641042** |  |
| --- | --- | --- |

**HALL MANAGEMENT SYSTEM**

## A PROJECT REPORT

***Submitted by***

|  |  |
| --- | --- |
| **SOUBARNIKA S -** | **727822TUCS226** |
|  |  |
|  |  |

***In partial fulfilment for the award of the degree of***

## BACHELOR OF ENGINEERING

## IN

**COMPUTER SCIENCE AND ENGINEERING**

**JULY 2024**

**BONAFIDE CERTIFICATE**

Certified that this project report **HALL MANAGMENT SYSTEM** is the bonafide work **SOUBARNIKA S 727822TUCS226** who carried out the project work under my supervision.

*SIGNATURE*

**Ms. A. GOMATHY**

**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 candidate was examined by me in the Project Work Viva Voce

examination held on at Sri Krishna College of Technology,

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.

We are grateful to our beloved Head, Computing Sciences **Dr.T. Senthilnathan,** for her tireless and relentless support.

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

We thank **Mr. A. Gomathy**, Assistant Professor, 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 Hall Management System is an advanced and comprehensive web application meticulously crafted to streamline the booking and management of event spaces. By leveraging a sophisticated tech stack that includes React.js for a highly dynamic and responsive user interface, Tailwind CSS for modern, versatile, and customizable styling, Spring Boot for a robust and scalable backend, REST APIs for seamless and efficient communication, and MySQL for dependable and secure data management, the system is designed to elevate both user experience and operational efficiency. Users are provided with a powerful platform that allows them to explore and view a diverse range of halls, filtered by various criteria such as location, size, rating, and amenities. The ability to bookmark preferred halls and make reservations effortlessly enhances the convenience and functionality of the application. The system’s intuitive and user-friendly design ensures accessibility for individuals with varying levels of technical expertise, while the backend infrastructure, supported by Spring Boot and REST APIs, guarantees reliable performance, scalability, and efficient data handling. MySQL underpins robust data management and security, ensuring the integrity and protection of user information. In conclusion, the Hall Management System represents a seamless integration of cutting-edge technologies and user-centric design principles, delivering a comprehensive and efficient solution for managing event spaces, ultimately simplifying the process of discovering and booking the ideal venue and significantly enhancing the overall user experience.

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **CHAPTER.NO** | **TITLE** | **PAGE NO** |
| 1 | INTRODUCTION | 1 |
|  | 1.1 PROJECT OVERVIEW | 1 |
|  | 1.2 SCOPE OF THE PROJECT | 1 |
|  | 1.3 OBJECTIVE | 1 |
| 2 | SYSTEM SPECIFICATIONS | 2 |
| 3 | SOFTWARE DEVELOPMENT LIFE CYCLE | 4 |
|  | 3.1 PHASES OF SDLC | 4 |
|  | 3.2 OVERVIEW OF SDLC | 4 |
| 4 | METHODOLOGIES | 6 |
| 5 | IMPLEMENTATION AND RESULT | 15 |
| 6 | CONCLUSION AND FUTURE SCOPE | 38 |
|  | 6.1 CONCLUSION | 38 |
|  | 6.2 FUTURE SCOPE | 39 |
| 7 | REFERENCES | 40 |

## LIST OF FIGURES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Figure No** | | | **TITLE** | **Page No** | |
|  | | |  |  | |
| 4.1 | Process-Flow Diagram | | | 6 | |
| 4.2 | Use Case Diagram | | | 8 | |
| 4.3 | Class Diagram | | | 9 | |
| 4.4 | Sequence Diagram | | | 10 | |
| 4.5  4.6  4.7  4.8 | Collaboration Diagram  Activity Diagram  State Diagram  Package Diagram | | | 11  12  14  14 | |
| 5.1 | Login page | | | 15 | |
| 5.2 | User Registration | | | 16 | |
| 5.3 | Exam Registration | | | 16 | |
| 5.4 | Database | | | 17 | |
| 5.5 | Payment page | | | 18 | |

## LIST OF ABBREVIATIONS

**Abbreviation Acronym**

**HTML** HYPERTEXT MARKUP LANGUAGE

**CSS** CASCADING STYLESHEET

**JS** JAVASCRIPT

**SDLC** SOFTWARE DEVELOPMENT LIFE CYCLE

**PHP** HYPER TEXT PREPROCESSO

# CHAPTER 1 INTRODUCTION

This project aims to offer a seamless and efficient solution for users to manage hall bookings through an online platform. In this chapter, we will explore the problem statement, provide an overview, and outline the main objectives of the hall management system.

## PROBLEM STATEMENT

How can we develop a Hall Management System that allows users to efficiently view, explore, bookmark, and reserve halls, considering factors such as location, size, rating, and amenities, while ensuring an intuitive user interface and minimizing booking conflicts?

## OVERVIEW

In the realm of hall management, users often face challenges such as finding suitable halls based on specific criteria, managing bookings, and accessing accurate information about hall features. Traditional methods can be cumbersome, with issues such as outdated information, inefficient search processes, and difficulties in managing reservations. To address these challenges, we propose the development of a Hall Management System. This system will utilize modern web technologies to deliver a comprehensive, user-friendly platform for viewing, exploring, bookmarking, and reserving halls. By incorporating advanced search functionalities, detailed hall information, and intuitive booking features, the system aims to streamline the reservation process, reduce conflicts, and enhance overall user satisfaction.

## OBJECTIVE

The primary objective of this project is to develop a Hall Management System that offers users a seamless and efficient platform for viewing, exploring, bookmarking, and reserving halls. The system aims to enhance the booking experience by providing comprehensive search capabilities, accurate and up-to-date hall information, and an intuitive user interface. By integrating features such as detailed hall descriptions, availability tracking, and streamlined reservation processes, the system seeks to improve user satisfaction and operational efficiency while minimizing booking conflicts and ensuring a smooth reservation experience.

# CHAPTER 2 SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website. This chapter gives you a small description about the software used in the project.

## VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting support to their projects.

## LOCAL STORAGE

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user’s computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request. Local storage is a key-value pair storage mechanism, meaning it stores data in the form of a key and corresponding value. It is similar to a database table in that it stores data in columns and rows, except that local storage stores the data in the browser rather than in a database. Local storage is often used to store user information such as

preferences and settings, or to store data that is not meant to be shared with other websites. It is also used to cache data to improve the performance of a website. Local storage is supported by all modern web browsers, including Chrome,

Firefox, Safari, and Edge. It is accessible through the browser’s JavaScript API. Local storage is a powerful tool for websites to store data on the client side. It is secure, efficient, and can be used to store data that does not need to be shared with other websites.

Local Storage is a great way to improve the performance of a website by caching data. Local storage in web browsers allows website data to be stored locally on the user’s computer. It is a way of persistently storing data on the client side, which is not sent to the server with each request. This allows users to store data such as preferences, login information, and form data without needing to send it to a server. It is typically stored in a browser’s cookie file, but it can also be stored in other locations such as HTML5 Local Storage and Indexed DB. The data stored in local storage is persistent and can be accessed by the website even if the user closes the browser or navigates to another page. It is a great way for websites to store user-specific data, as it is secure, reliable, and fast. It is also a great way for developers to store data that does not need to be sent to the server with each request.

One of the key benefits of using local storage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storage is stored locally on the user’s machine, and so is not affected by these issues. Another advantage of local storage is its speed. Because the data is stored locally, it is accessed quickly, as there is no need to send requests to a server. This makes it ideal for storing data that needs to be accessed quickly, such as user preferences or session data. Local storage is also secure, as the data is stored on the user’s machine and not on a server. This means that the data is not accessible by anyone other than the user, making it a good choice for storing sensitive information.

# CHAPTER 3

# PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

## PROPOSED SYSTEM

The proposed Hall Management System aims to revolutionize the way users interact with hall booking processes, providing a streamlined and efficient solution for managing reservations. Upon logging in, users gain access to a sophisticated platform that allows them to check the availability of halls in proximity to their location. This feature simplifies the search process, enabling users to find suitable venues quickly and conveniently. The system goes beyond basic search functionality, offering users the ability to explore a wide range of halls, each with detailed descriptions, images, and essential information about their size, amenities, and ratings.

Users can bookmark their preferred halls, creating a personalized list of potential venues for future events. This feature is particularly useful for users planning multiple events or who wish to compare different options before making a final decision. The reservation process is made straightforward through an intuitive interface that allows users to select available dates and complete their bookings with minimal effort.

On the other side, hall owners benefit from a well-organized management system that streamlines the handling of reservation requests. The system provides a centralized view of all incoming requests, enabling hall owners to efficiently review, approve, or reject bookings based on availability and other criteria. This capability not only reduces administrative workload but also ensures that the scheduling of halls is managed effectively, avoiding potential conflicts and optimizing the use of available spaces.

In summary, the Hall Management System provides a comprehensive and user-friendly solution for hall bookings. It simplifies the process for users by offering convenient search, bookmarking, and reservation features, while empowering hall owners with efficient request management tools. The integration of these functionalities ensures a smooth and effective booking experience, ultimately leading to increased satisfaction for all stakeholders involved.

## ADVANTAGES

**Efficiency:** The Hall Management System allows users to quickly and easily search for, bookmark, and reserve halls from any location with internet access. This eliminates the need for time-consuming manual processes, reducing the administrative workload for hall owners and streamlining the booking process for users.

**Flexibility:** Users can explore a variety of halls, check availability, and make reservations at their convenience. This flexibility accommodates different event planning needs and preferences, improving user satisfaction and ensuring that hall bookings align with their schedules.

**Conflict Resolution:** The system automatically manages booking availability and prevents double-bookings by ensuring that halls are reserved only for available dates. This feature minimizes the chances of scheduling conflicts and ensures smooth operation.

**Accessibility:** Both users and hall owners can access the Hall Management System from any location with internet access. This accessibility facilitates remote management of bookings and reservations, enhancing coordination and planning without geographical constraints.

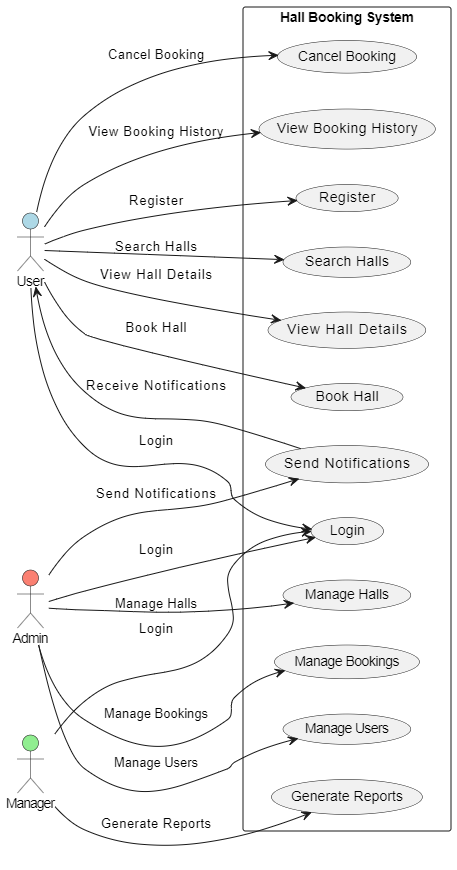
**Transparency:** The system provides clear and detailed information about hall availability, features, and reservations. This transparency helps users make informed decisions and allows hall owners to efficiently manage and track reservation requests, reducing confusion and improving communication.

# CHAPTER 4 METHODOLOGIES

# Fig 4.1.Process flow diagram

## Use Case Diagram:

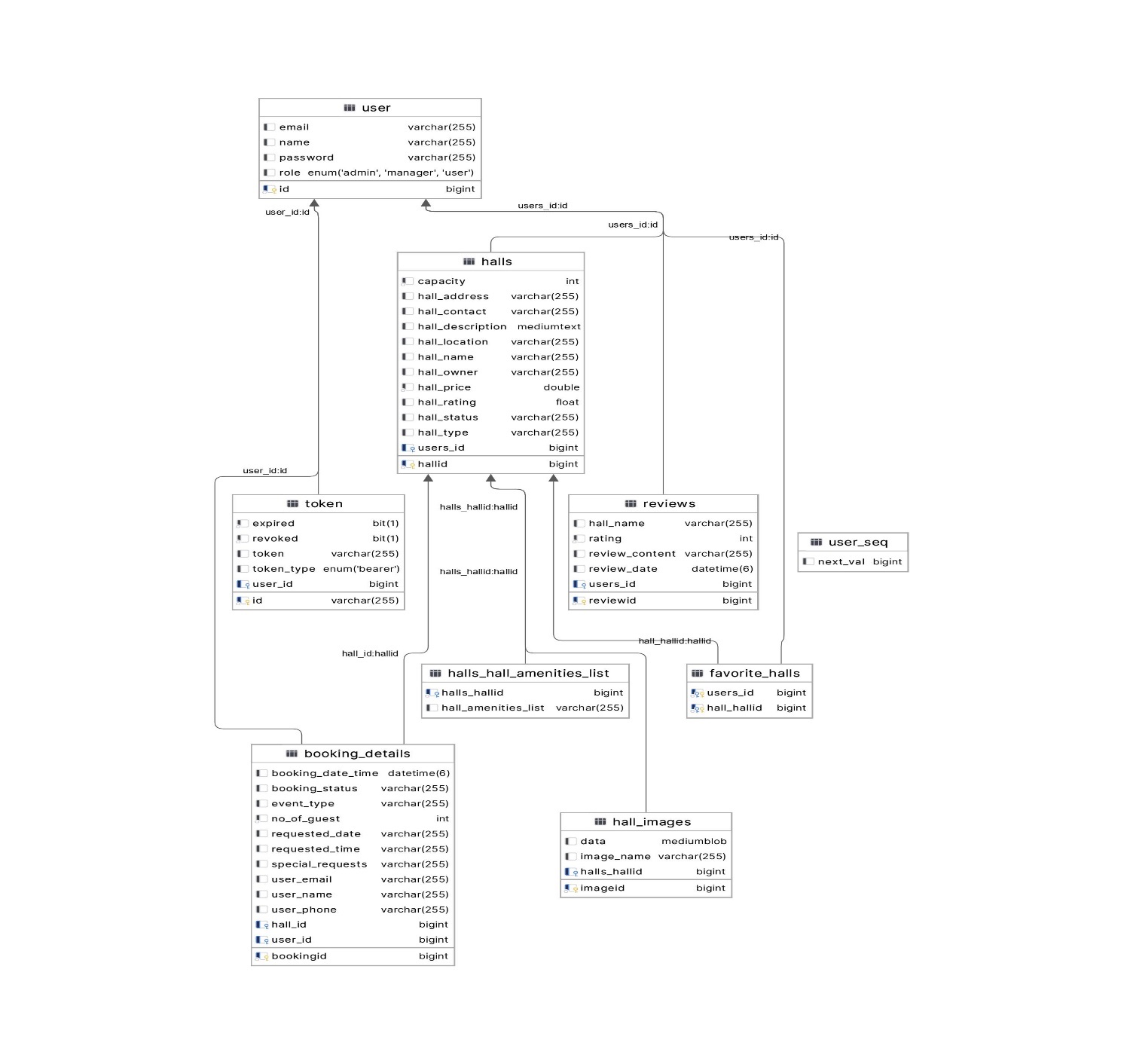
## A use case diagram is a visual representation that illustrates the interactions between users and the Hall Management System. It captures the functional requirements of the system by detailing the various scenarios in which users engage with the system. In this diagram, different types of users, such as event organizers and hall owners, are depicted as actors interacting with the system’s use cases. For instance, an event organizer may search for available halls, view detailed information, bookmark preferred options, and make reservations, while a hall owner manages reservation requests, approves or rejects bookings, and updates hall availability. The use case diagram provides a clear and structured overview of how different users interact with the system, helping to ensure that all functional requirements are considered and integrated into the design of the Hall Management System.



4.2 Use Case Diagram

**Class Diagram:**

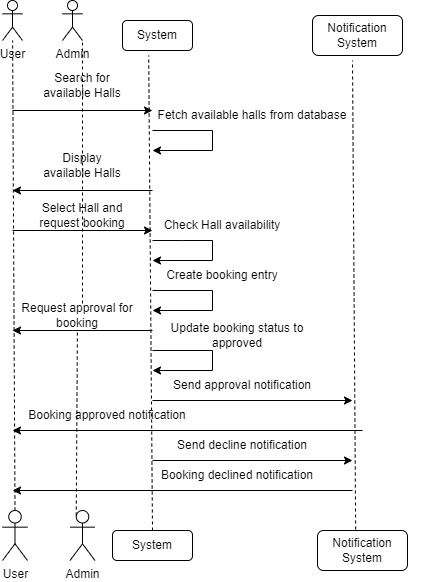
The class diagram for the Hall Management System visually represents the system’s structure and the interactions between various entities. Key classes include **BookingDetails**, which manages reservation information; Halls, which details venue attributes; and User, which captures user information and roles. Additionally, **FavoriteHalls** tracks user’s favorite venues, **Reviews** handles user feedback, **HallImages** manages images of halls, and **HallsHallAmenitiesList** lists amenities. The Token class deals with authentication. The diagram outlines how these classes are interconnected, providing a clear blueprint for the system’s architecture and development.



4.3 Class Diagram

**Sequence Diagram:**

The Hall Booking Sequence Diagram illustrates the process for a user to request a hall booking. It details the interaction between the User, Hall Management System, and Hall Owner components. The diagram shows how the user searches for available halls, selects a preferred hall, and submits a booking request. The Hall Management System processes the request by checking hall availability and sending the booking details to the Hall Owner. The Hall Owner then reviews the request, approves or rejects it, and updates the system with the booking status. This sequence ensures that the user is informed of the booking outcome and that the hall reservation process is managed efficiently.



4.4 Sequence Diagram

**CHAPTER 5 IMPLEMENTATION AND RESULT**

This chapter gives a description about the output that we produced by developing the website of our idea.

## 5.1 LANDING PAGE

When a user visits the landing page of our website, they are greeted with a welcoming interface that provides an overview of the available features. The landing page allows users to quickly access key functionalities, such as searching for available halls, exploring detailed information about various venues.

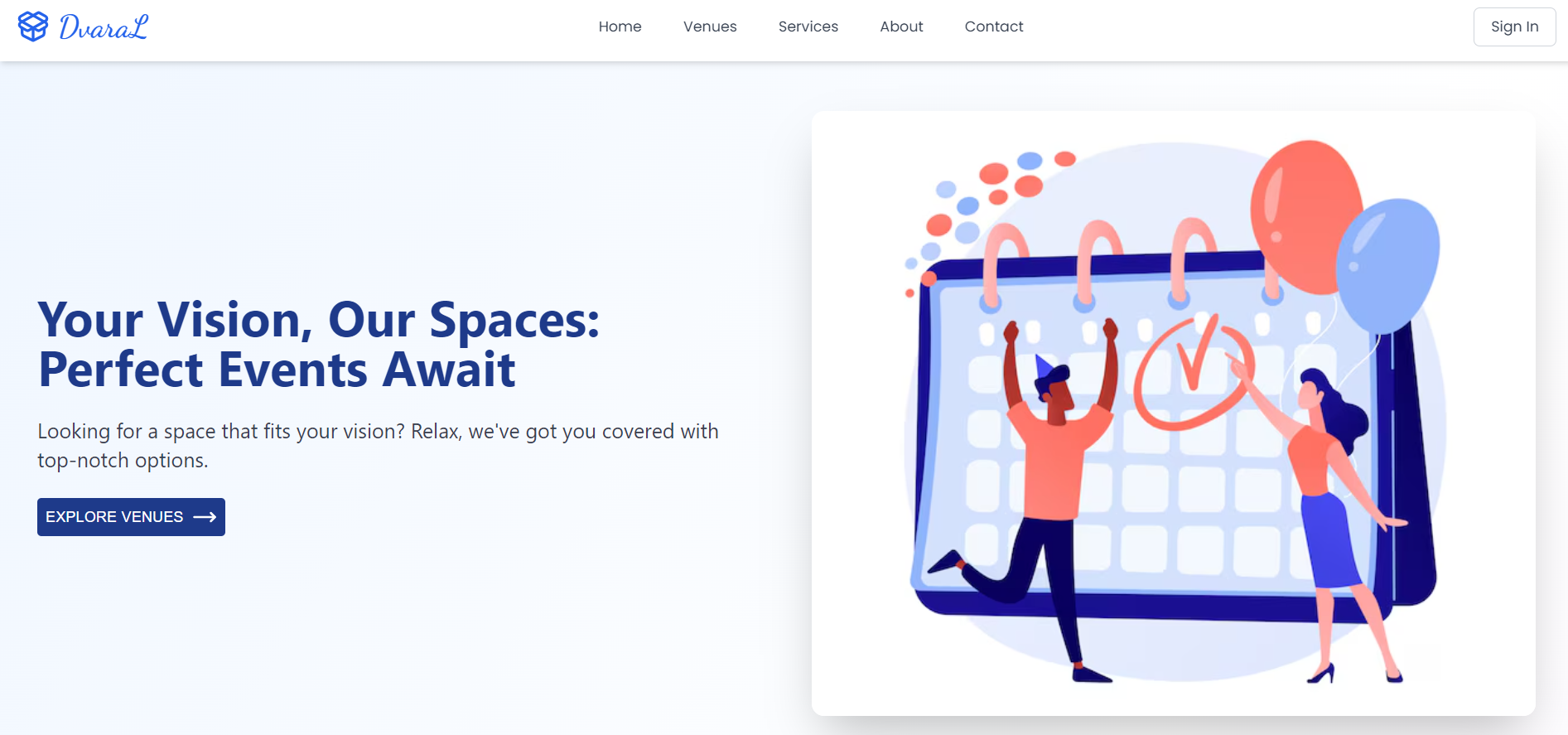


Fig 5.1 LANDING PAGE

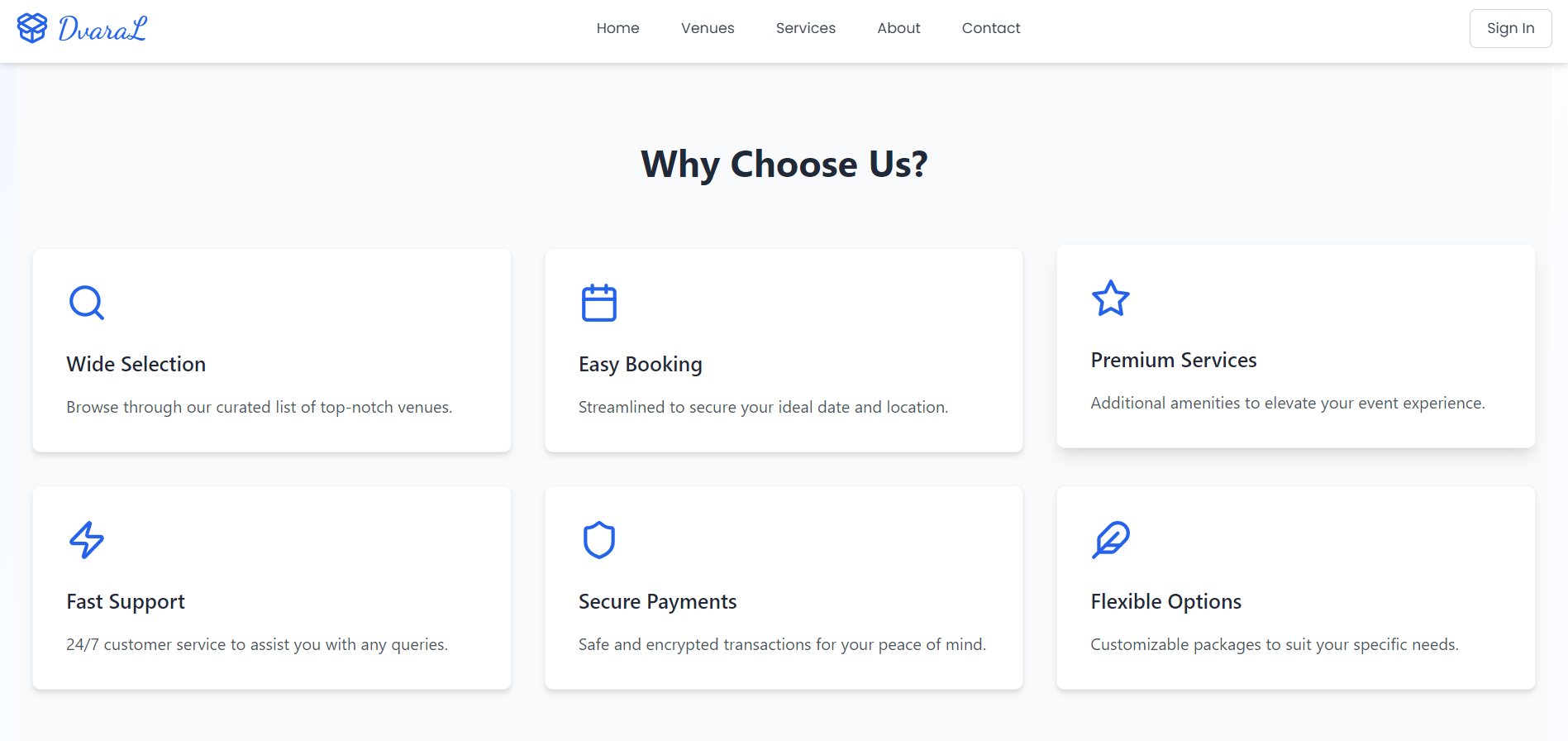


Fig 5.1 LANDING PAGE

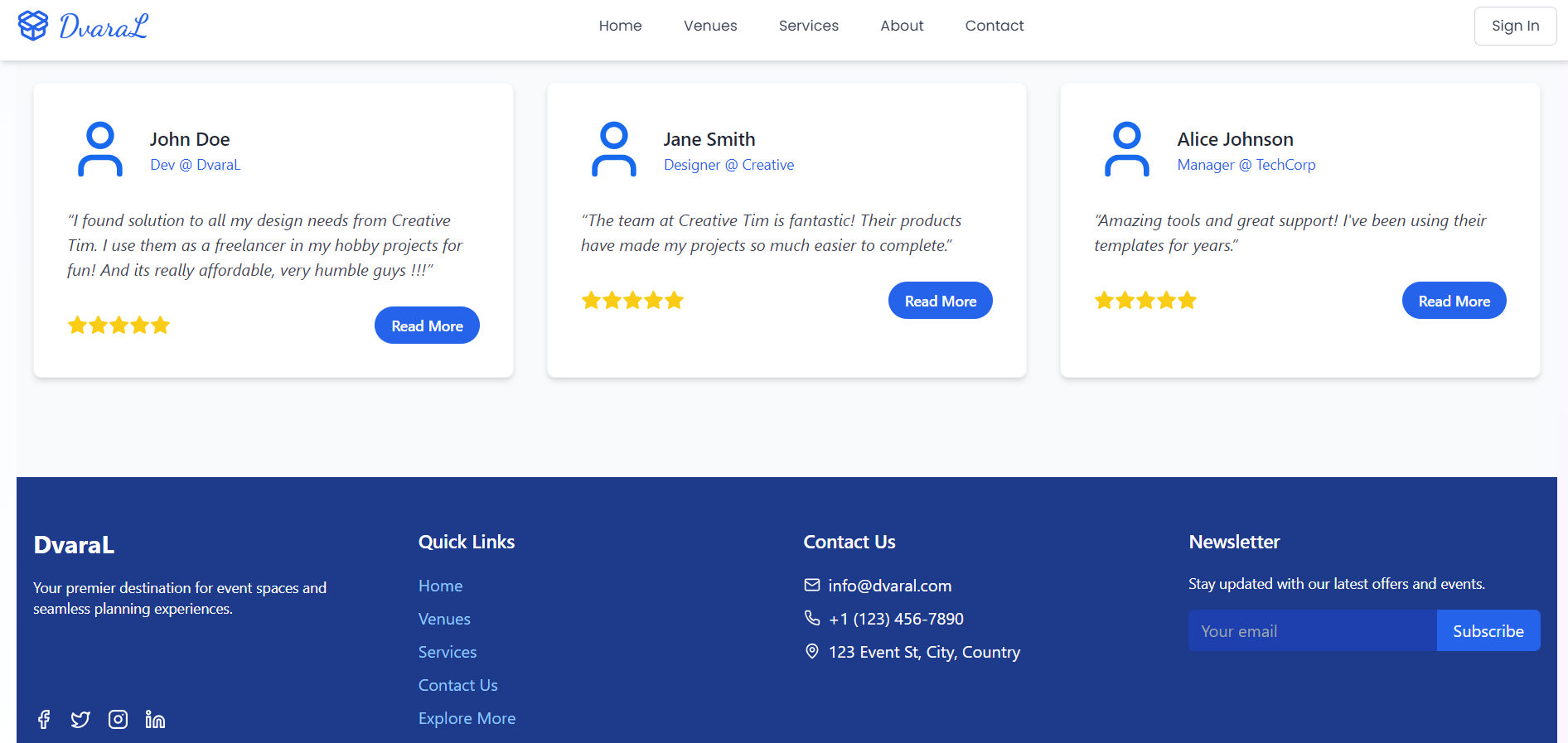


Fig 5.1 LANDING PAGE

## LOGIN

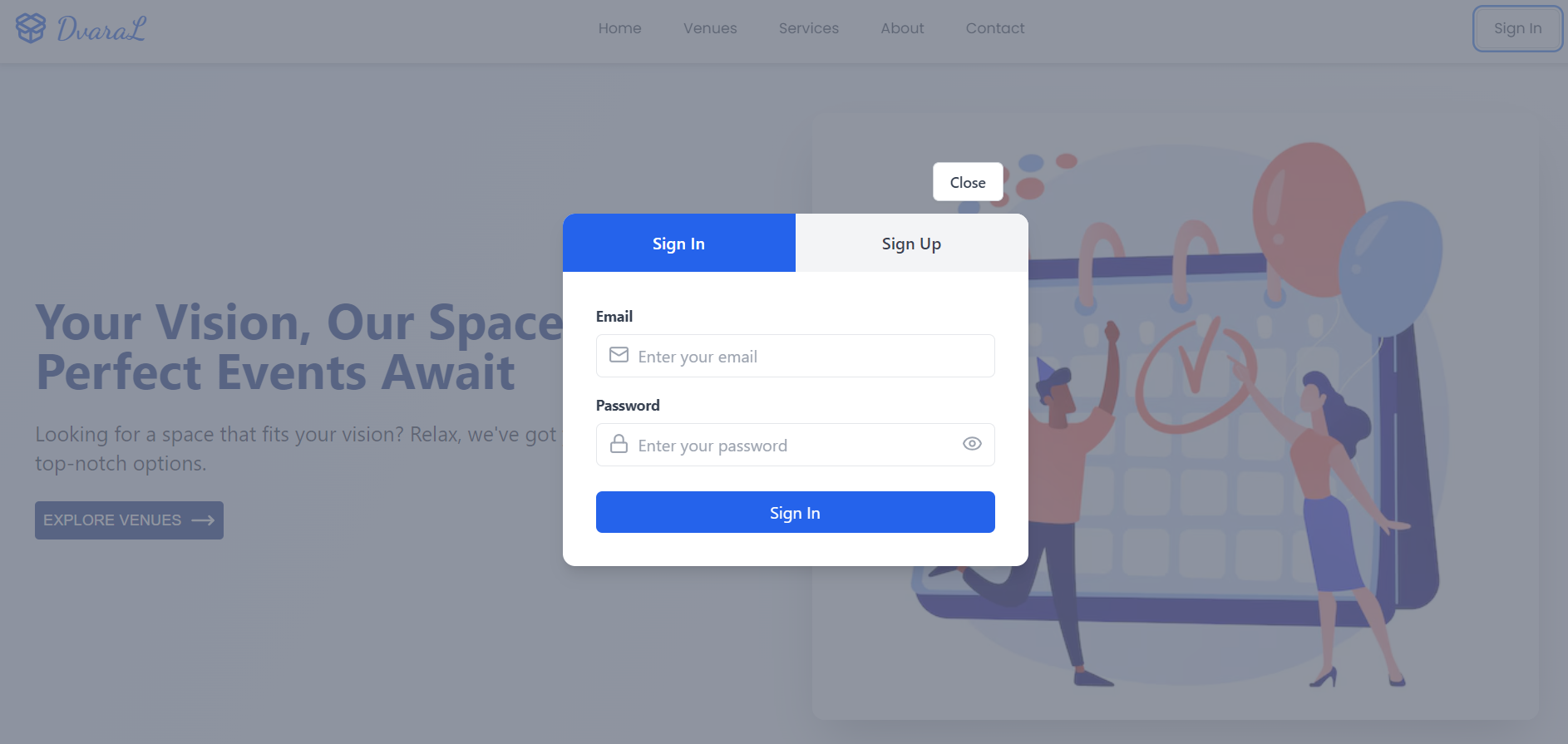


Fig 5.2 LOGIN PAGE

* 1. **REGISTER PAGE**

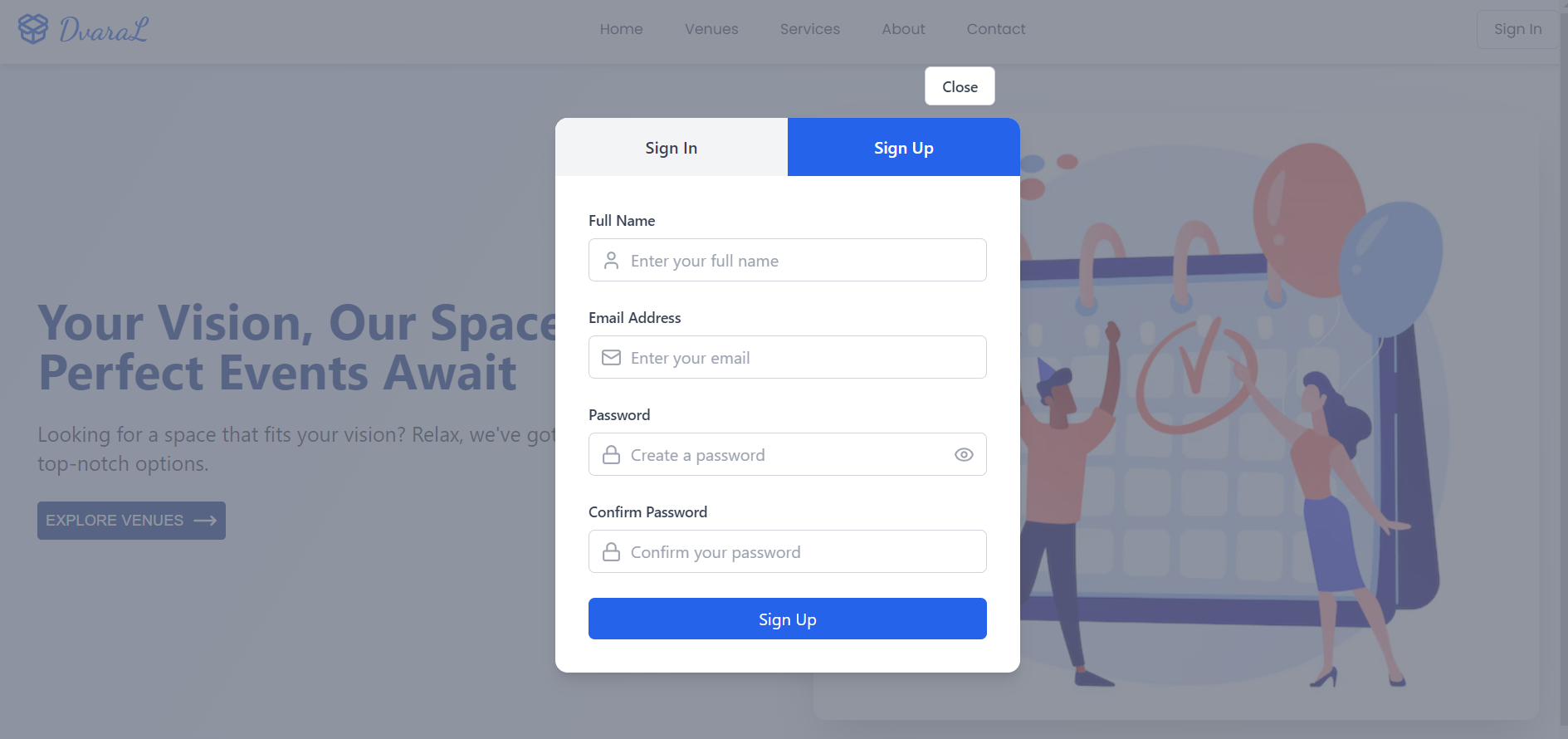
****

Fig 5.3 USER REGISTER

## USER DASHBOARD

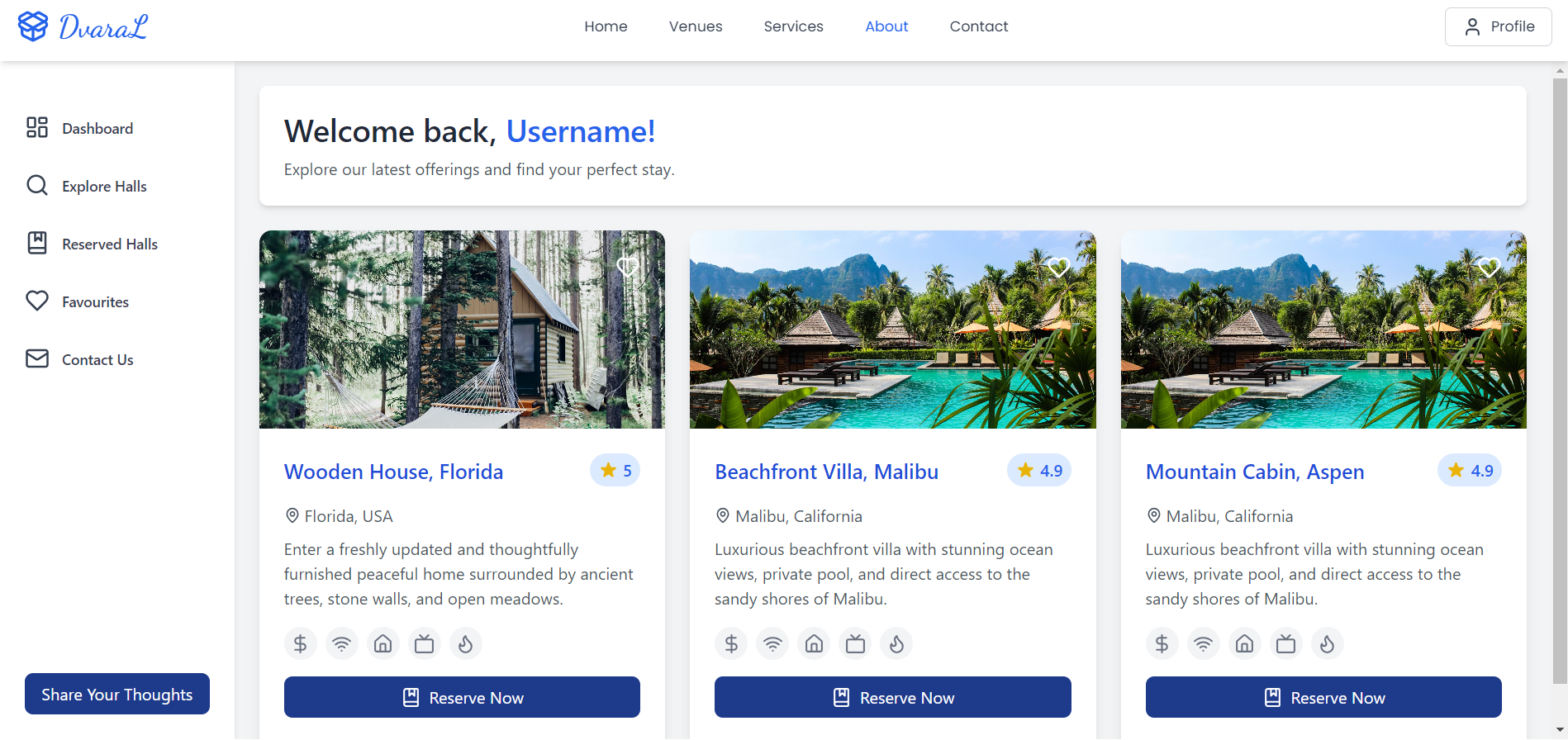


Fig 5.4 User Dashboard

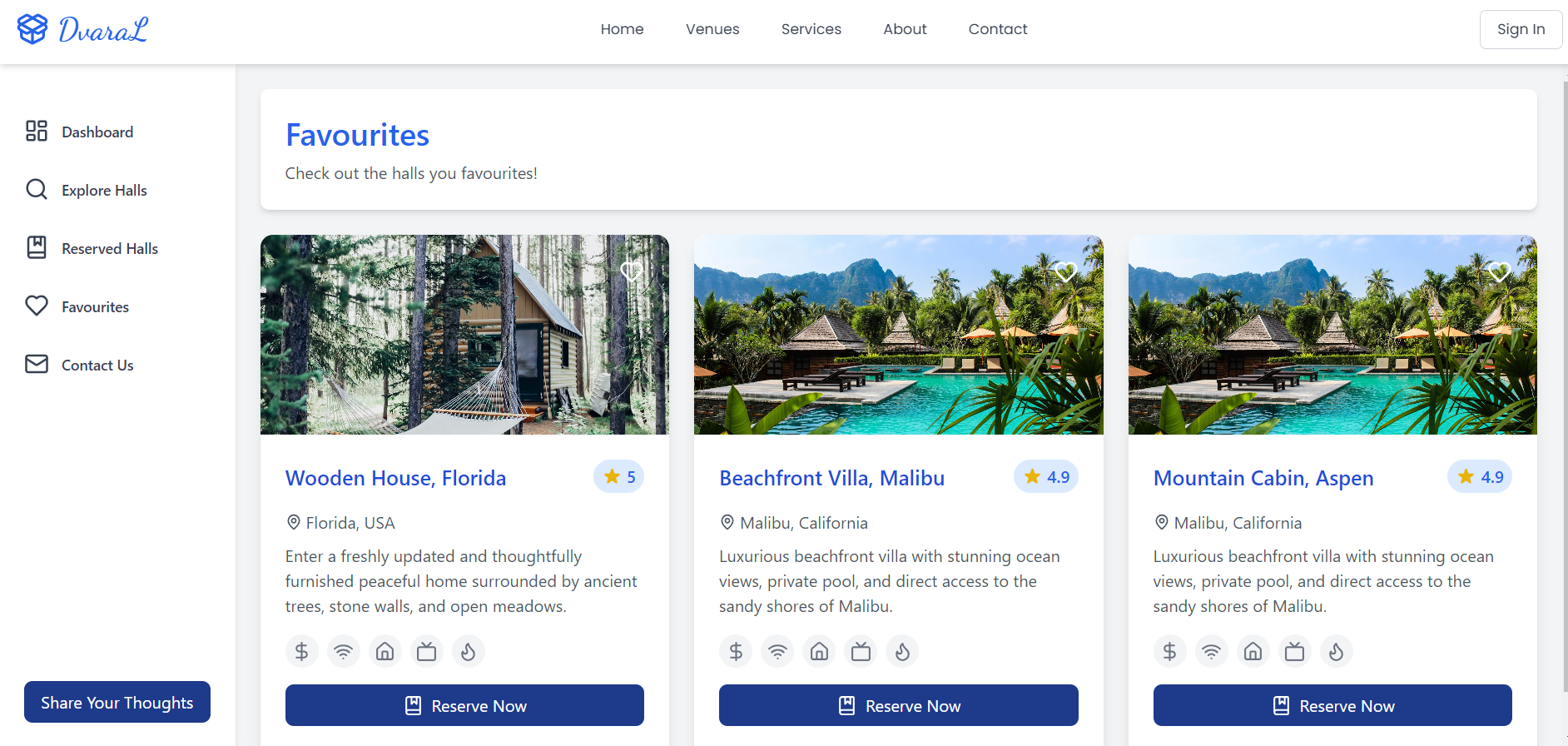


Fig 5.4 User Dashboard

## EXPLORE VENUES

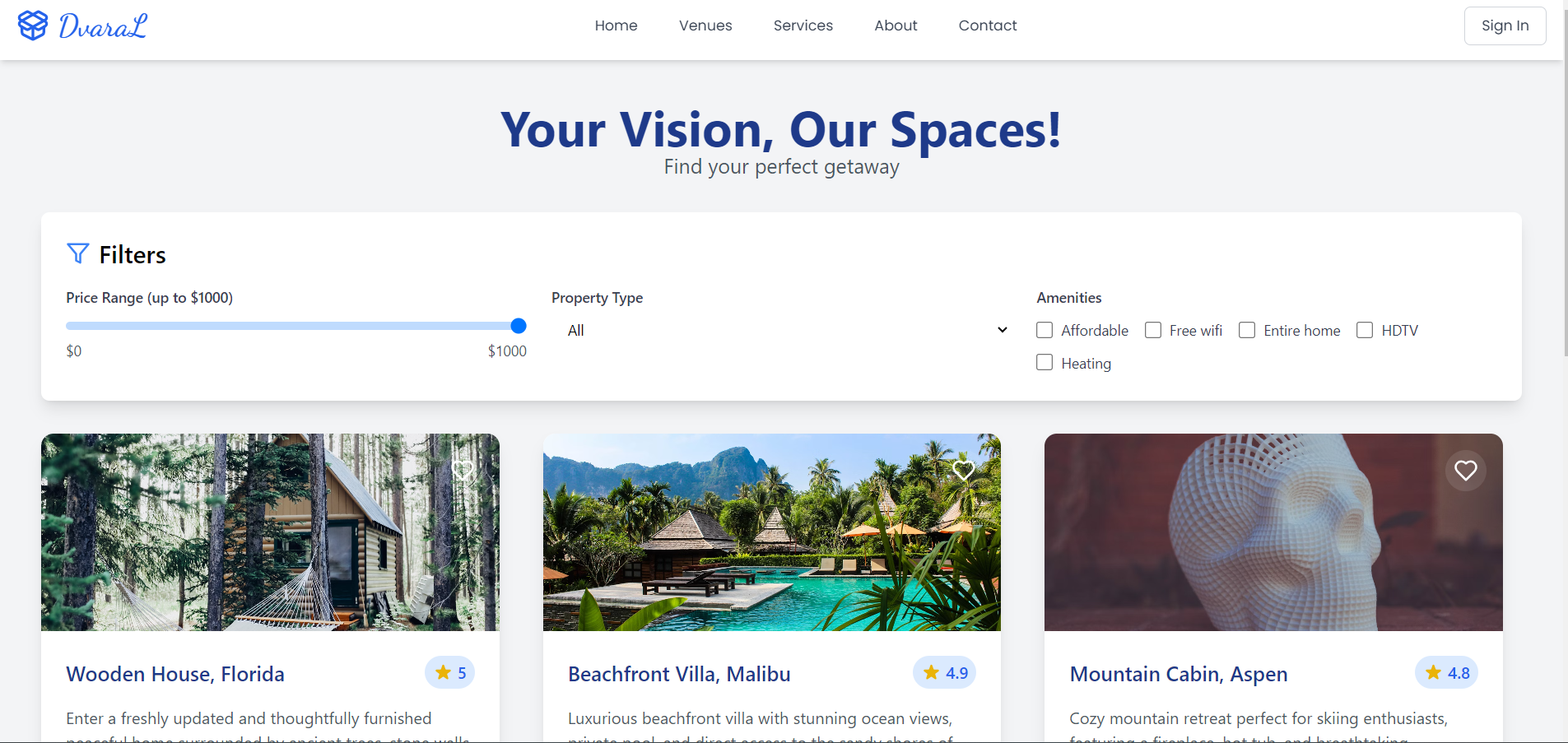


Fig 5.5 Explore Venues

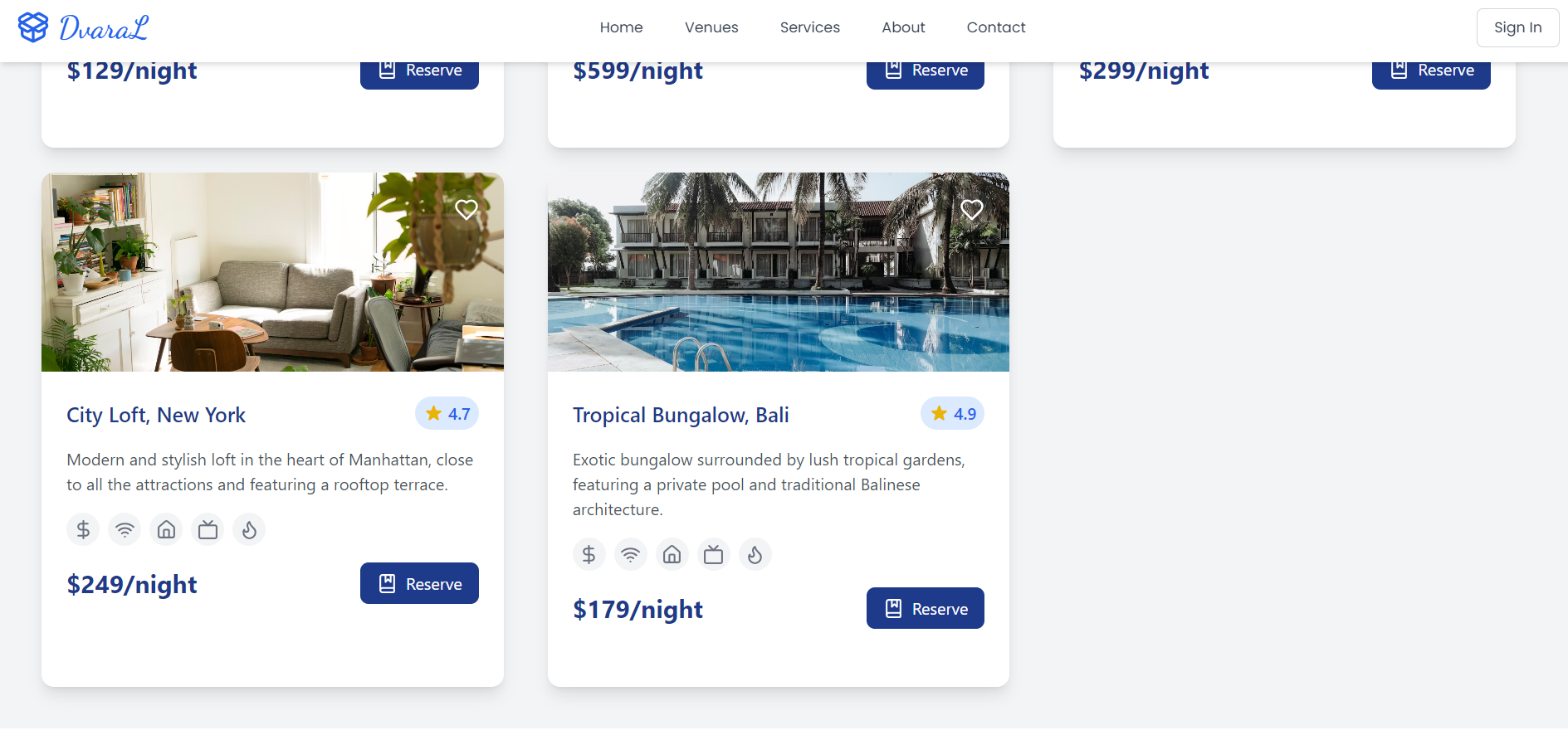


Fig 5.5 Explore Venues

## ADMIN DASHBOARD

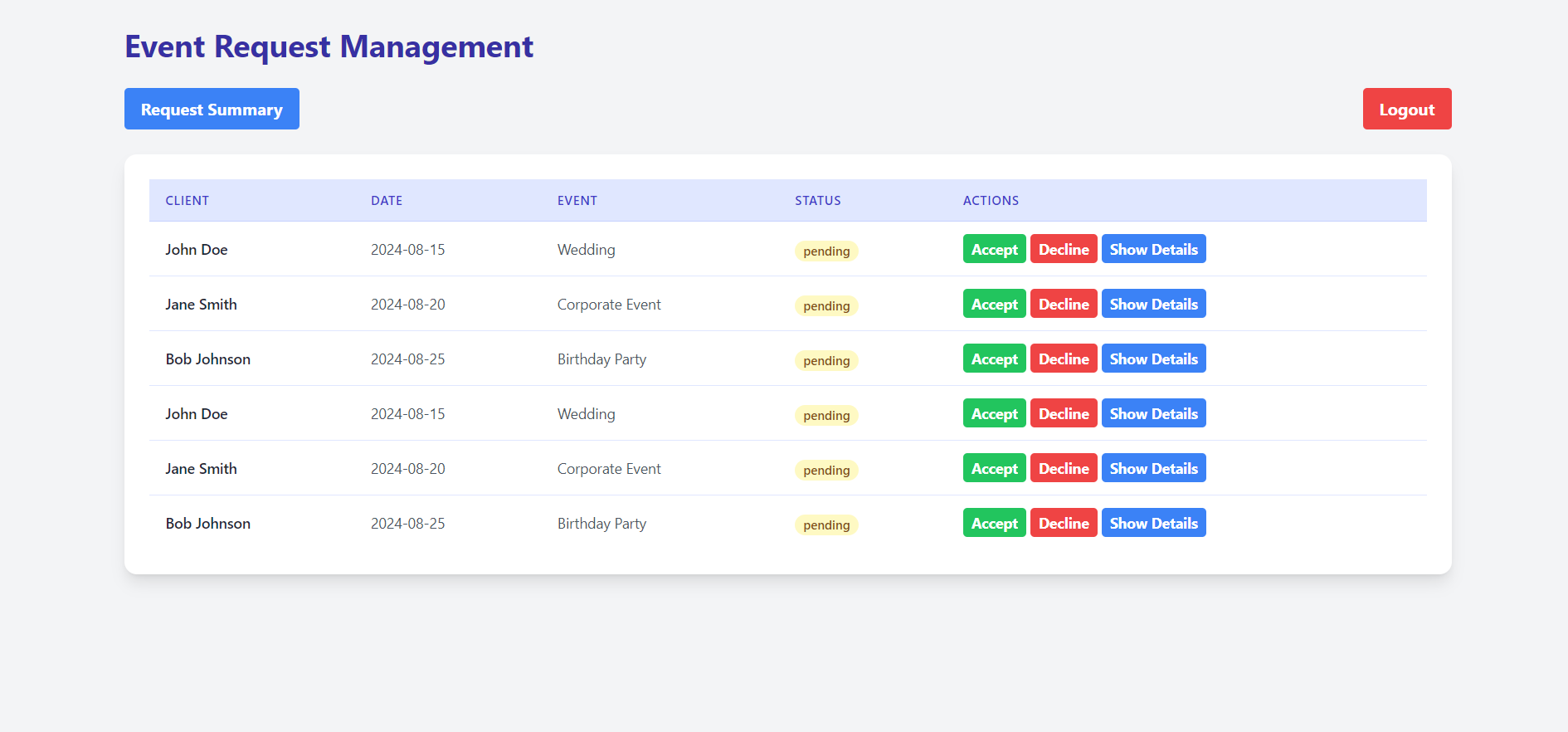


Fig 5.6 Admin Dashboard

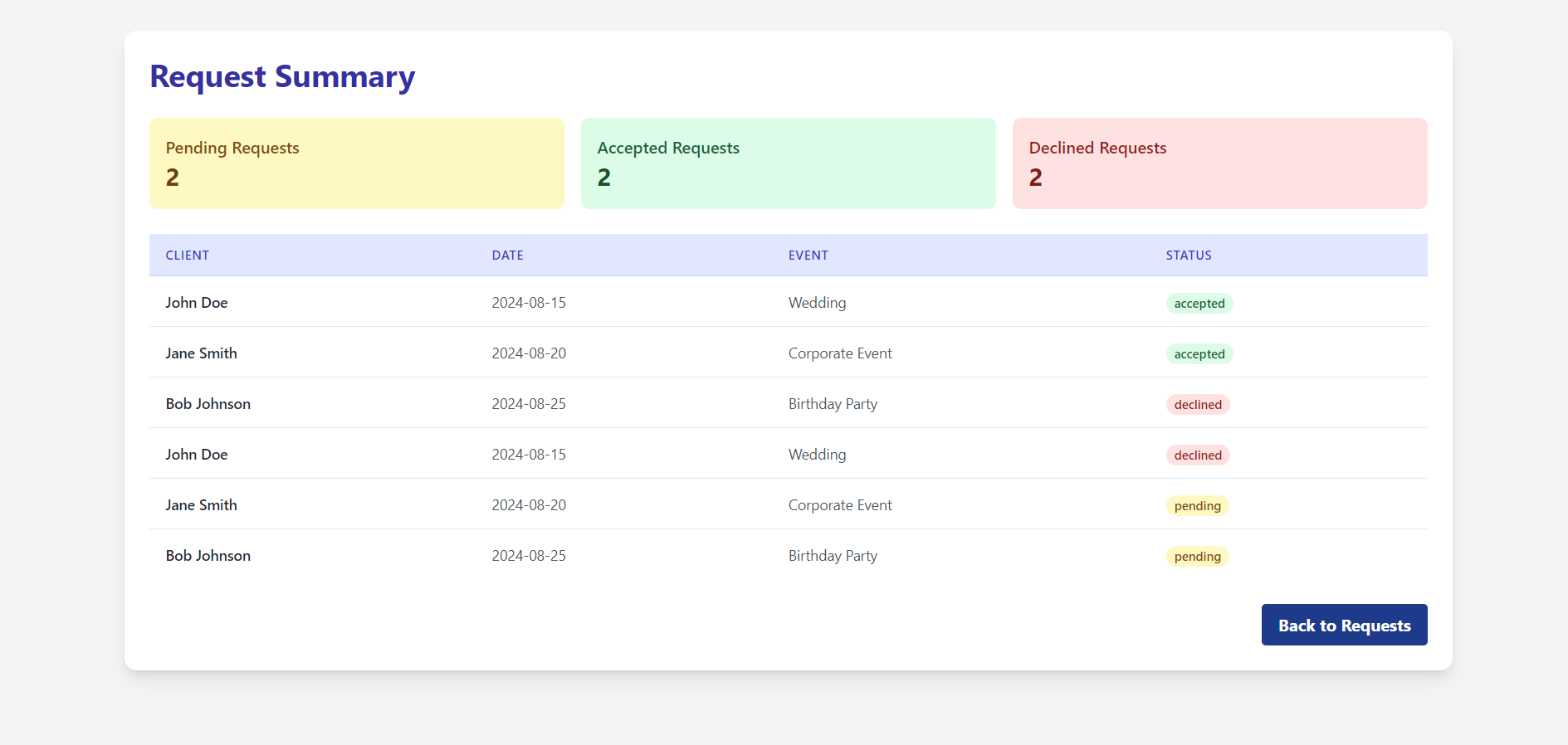


Fig 5.6 Admin Dashboard

## CODING

### Config:

### @Configuration @RequiredArgsConstructor public class ApplicationConfig { private final UserRepo userRepo; @Bean public UserDetailsService userDetailsService(){ return username -> userRepo.findByEmail(username) .orElseThrow(() -> new UsernameNotFoundException("User not Found")); } @Bean public AuthenticationProvider authenticationProvider(){ DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider(); authProvider.setUserDetailsService(userDetailsService()); authProvider.setPasswordEncoder(passwordEncoder()); return authProvider; } @Bean public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception { return config.getAuthenticationManager(); } @Bean public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); } }

### @Configuration public class CorsConfig { @Bean public WebMvcConfigurer corsConfigurer() { return new WebMvcConfigurer() { @Override public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/\*\*") .allowedOrigins("localhost:5173") .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS") .allowedHeaders("\*") .allowCredentials(true); } }; } }

### @Component @RequiredArgsConstructor public class JwtAuthenticationFilter extends OncePerRequestFilter { private final JwtService jwtService; private final UserDetailsService userDetailsService; @Override protected void doFilterInternal( @NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain filterChain) throws ServletException, IOException { final String authHeader =request.getHeader("Authorization"); final String jwtToken; final String userEmail; if(authHeader == null || !authHeader.startsWith("Bearer ")) { filterChain.doFilter(request,response); return; } jwtToken = authHeader.substring(7); userEmail = jwtService.extractUserName(jwtToken);if(userEmail != null && SecurityContextHolder.*getContext*().getAuthentication() == null){ UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail); if(jwtService.isTokenValid(jwtToken, userDetails)){ UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken( userDetails, null, userDetails.getAuthorities() ); authToken.setDetails( new WebAuthenticationDetailsSource().buildDetails(request) ); SecurityContextHolder.*getContext*().setAuthentication(authToken); } } filterChain.doFilter(request,response); }}

### @Service public class JwtService { private static final String *SECRET\_KEY* = "EbeEsh7VhXpHMAkLz7Xb3TYm7a4KLMlYn0Kr1NJEhTIOeU9HJsv3t2bMa5OjoiaD"; public String extractUserName(String token) { return extractClaim(token, Claims::getSubject); } public <T> T extractClaim(String token, Function<Claims, T> claimsResolver){ final Claims claims = extractAllClaims(token); return claimsResolver.apply(claims); } public String generateToken(UserDetails userDetails){ return generateToken(new HashMap<>(), userDetails); } public String generateToken( Map<String, Object> extraClaims, UserDetails userDetails ){ return Jwts .*builder*() .claims(extraClaims). subject(userDetails.getUsername()) .issuedAt(new Date(System.*currentTimeMillis*())) .expiration(new Date(System.*currentTimeMillis*() + 1000 \* 60 \* 24)) .signWith(getSignInKey(), SignatureAlgorithm.*HS256*) .compact(); } public Boolean isTokenValid(String token ,UserDetails userDetails) { final String username = extractUserName(token); return (username.equals(userDetails.getUsername()) && !isTokenExpired(token)); } private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); } private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); } private Claims extractAllClaims(String token) { return Jwts .*parser*() .setSigningKey(getSignInKey()) .build() .parseClaimsJws(token) .getBody(); } private Key getSignInKey() { byte[] keyByte = Decoders.*BASE64*.decode(*SECRET\_KEY*); return Keys.*hmacShaKeyFor*(keyByte); } }

### @Configuration public class LogoutConfiguration { @Bean public CustomLogoutHandler logoutHandler(TokenRepo tokenRepo, JwtService jwtService) { return new CustomLogoutHandler(tokenRepo, jwtService); } @Bean public LogoutSuccessHandler logoutSuccessHandler() { return new CustomLogoutSuccessHandler(); } }

### Landing page:

### import React from 'react';

### import Navbar from "../../Web/Navbar";

### import LandingPageContent from './LandingPageContent';

### import Footer from './Footer';

### import { MoveRight } from 'lucide-react';

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

### import LandingPageImg from '/src/assets/images/LP2.avif';

### import { Button } from '@mui/material';

### const LandingPage = () => {

### return (

### <div className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-purple-50">

### <Navbar />

### 

### User Dashboard:

### import React from "react";

### import Navbar from '../../Web/Navbar';

### import { Search, BookMarked, Heart, Mail, LayoutDashboard } from "lucide-react";

### import { BookingCard } from "./BookingCard";

### import UserSideBar from './UserSideBar';

### const UserDashboard = () => {

### 

### return (

### <div className="flex flex-col min---h-screen bg-gray-100">

### <Navbar />

### <div className="flex flex-grow overflow-hidden h-[90vh]">

### {/\* Sidebar \*/}

### <UserSideBar/>

### {/\* Main Content \*/}

### <main className="flex-grow p-6 overflow-y-auto">

### <div className="bg-white rounded-lg shadow-md p-6 mb-6">

### <h1 className="text-3xl font-semibold text-gray-800">Welcome back, <span className="text-blue-600">Username!</span></h1>

### <p className="text-gray-600 mt-2">Explore our latest offerings and find your perfect stay.</p>

### </div>

### <BookingCard/>

### </main>

### </div>

### </div>

### <main className="container mx-auto px-4 py-6 sm:py-8 md:py-12">

### <section className="mb-10 sm:mb-16 md:mb-20">

### <div className="flex flex-col lg:flex-row items-center justify-between gap-6 sm:gap-8 md:gap-12">

### <div className="w-full lg:w-1/2 space-y-4 sm:space-y-6 p-2 sm:p-4 md:p-5">

### <h1 className="text-2xl sm:text-3xl md:text-4xl lg:text-5xl font-bold text-blue-900 leading-tight font-josefin-sans">

### Your Vision, Our Spaces: <br />

### <span className="text-blue-900">Perfect Events Await</span>

### </h1>

### <p className="text-base sm:text-lg md:text-xl text-gray-700 font-poppin">

### Looking for a space that fits your vision?

### Relax, we've got you covered with top-notch options.

### </p>

### <Link to="/explore-page">

### <div className='mt-6'>

### <Button

### style={{ color: "#fff" , backgroundColor: "#1E3A8A"}}

### className=" hover:bg-blue-900 px-4 sm:px-6 md:px-8 py-2 sm:py-3 md:py-4 rounded-full text-sm sm:text-base md:text-lg font-semibold transition-all duration-300 ease-in-out"

### >

### Explore Venues <MoveRight className="ml-2" />

### </Button>

### </div>

### </Link>

### </div>

### <div className="w-full lg:w-2/4 mt-6 lg:mt-0 pr-5">

### <img

### src={LandingPageImg}

### alt="Elegant Event Space"

### className="rounded-xl shadow-2xl object-cover w-full h-48 sm:h-64 md:h-80 lg:h-96 xl:h-[80vh]"

### />

### </div>

### </div>

### </section>

### <LandingPageContent />

### <Footer />

### </main>

### </div>

### );

### }

### export default LandingPage;

### );

### };

### export default UserDashboard;

### RESERVATION PAGE

### import React from "react";

### import Navbar from "../../Web/Navbar";

### import { BookingCard } from "./BookingCard";

### import UserSideBar from './UserSideBar';

### const ReservedHall = () => {

### return (

### <>

### <div className="flex flex-col min---h-screen bg-gray-100">

### <Navbar />

### <div className="flex flex-grow overflow-hidden h-[90vh]">

### <UserSideBar />

### <main className="flex-grow p-6 overflow-y-auto">

### <div className="bg-white rounded-lg shadow-md p-6 mb-6">

### <h1 className="text-3xl font-semibold text-blue-600">Reserved Halls</h1>

### <p className="text-gray-600 mt-2">Check out the halls you reserved!</p>

### </div>

### <BookingCard />

### </main>

### </div>

### </div>

### </>

### )

### }

### export default ReservedHall;

### ADMIN DASHBOARD

### import React, { useState } from 'react';

### import { useNavigate } from 'react-router-dom';

### import RequestSummary from './RequestSidePanel';

### function RequestManagement() {

### const navigate = useNavigate();

### const [requests, setRequests] = useState([

### {

### id: 1,

### client: 'John Doe',

### date: '2024-08-15',

### event: 'Wedding',

### status: 'pending',

### hallDetails: {

### name: 'Grand Ballroom',

### capacity: 200,

### price: '$2000',

### amenities: ['Stage', 'Dance floor', 'Audio system']

### }

### },

### {

### id: 2,

### client: 'Jane Smith',

### date: '2024-08-20',

### event: 'Corporate Event',

### status: 'pending',

### hallDetails: {

### name: 'Conference Center',

### capacity: 150,

### price: '$1500',

### amenities: ['Projector', 'Whiteboard', 'Wi-Fi']

### }

### },

### {

### id: 3,

### client: 'Bob Johnson',

### date: '2024-08-25',

### event: 'Birthday Party',

### status: 'pending',

### hallDetails: {

### name: 'Party Hall',

### capacity: 100,

### price: '$1000',

### amenities: ['Decorations', 'Catering kitchen', 'Sound system']

### }

### },

### {

### id: 4,

### client: 'John Doe',

### date: '2024-08-15',

### event: 'Wedding',

### status: 'pending',

### hallDetails: {

### name: 'Grand Ballroom',

### capacity: 200,

### price: '$2000',

### amenities: ['Stage', 'Dance floor', 'Audio system']

### }

### },

### {

### id: 5,

### client: 'Jane Smith',

### date: '2024-08-20',

### event: 'Corporate Event',

### status: 'pending',

### hallDetails: {

### name: 'Conference Center',

### capacity: 150,

### price: '$1500',

### amenities: ['Projector', 'Whiteboard', 'Wi-Fi'] }

### },

### {

### id: 6,

### client: 'Bob Johnson',

### date: '2024-08-25',

### event: 'Birthday Party',

### status: 'pending',

### hallDetails: {

### name: 'Party Hall',

### capacity: 100,

### price: '$1000',

### amenities: ['Decorations', 'Catering kitchen', 'Sound system']

### }

### },

### ]);

### const [expandedId, setExpandedId] = useState(null);

### const [showSummary, setShowSummary] = useState(false);

### const handleAccept = (id) => {

### setRequests(requests.map(req =>

### req.id === id ? { ...req, status: 'accepted' } : req

### ));

### };

### const handleDecline = (id) => {

### setRequests(requests.map(req =>

### req.id === id ? { ...req, status: 'declined' } : req

### ));

### };

### const toggleExpand = (id) => {

### setExpandedId(expandedId === id ? null : id);

### };

### const handleRequestSummary = () => {

### setShowSummary(true);

### };

### const handleCloseSummary = () => {

### setShowSummary(false);

### };

### const handleLogout = () => {

### navigate('/');

### console.log("Logout button clicked");

### };

### if (showSummary) {

### return <RequestSummary requests={requests} onClose={handleCloseSummary} />;

### }

### return (

### <div className="bg-gray-100 min-h-screen p-4 sm:p-6 md:p-8">

### <div className="max-w-7xl mx-auto">

### <h2 className="text-2xl sm:text-3xl font-bold text-indigo-800 mb-6">Event Request Management</h2>

### <div className="flex flex-col sm:flex-row justify-between mb-6">

### <button

### onClick={handleRequestSummary}

### className="bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-4 rounded transition-colors duration-200 mb-2 sm:mb-0"

### >

### Request Summary

### </button>

### <button

### onClick={handleLogout}

### className="bg-red-500 hover:bg-red-600 text-white font-bold py-2 px-4 rounded transition-colors duration-200"

### >

### Logout

### </button>

### </div>

### <div className="bg-white shadow-lg rounded-xl p-4 sm:p-6 overflow-x-auto">

### <table className="w-full divide-y divide-indigo-200">

### <thead className="bg-indigo-100">

### <tr>

### <th className="px-4 py-3 text-left text-xs font-semibold text-indigo-700 uppercase tracking-wider">Client</th>

### <th className="px-4 py-3 text-left text-xs font-semibold text-indigo-700 uppercase tracking-wider">Date</th>

### <th className="px-4 py-3 text-left text-xs font-semibold text-indigo-700 uppercase tracking-wider">Event</th>

### <th className="px-4 py-3 text-left text-xs font-semibold text-indigo-700 uppercase tracking-wider">Status</th>

### <th className="px-4 py-3 text-left text-xs font-semibold text-indigo-700 uppercase tracking-wider">Actions</th>

### </tr>

### </thead>

### <tbody className="bg-white divide-y divide-indigo-100">

### {requests.map((request) => (

### <React.Fragment key={request.id}>

### <tr className="hover:bg-indigo-50 transition-colors duration-200">

### <td className="px-4 py-3 whitespace-nowrap text-sm font-medium text-gray-800">{request.client}</td>

### <td className="px-4 py-3 whitespace-nowrap text-sm text-gray-600">{request.date}</td>

### <td className="px-4 py-3 whitespace-nowrap text-sm text-gray-600">{request.event}</td>

### <td className="px-4 py-3 whitespace-nowrap">

### <span className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-full

### ${request.status === 'accepted' ? 'bg-green-100 text-green-800' :

### request.status === 'declined' ? 'bg-red-100 text-red-800' :

### 'bg-yellow-100 text-yellow-800'}`}>

### {request.status}

### </span>

### </td>

### <td className="px-4 py-3 whitespace-nowrap text-sm font-medium">

### {request.status === 'pending' && (

### <>

### <button onClick={() => handleAccept(request.id)} className="bg-green-500 hover:bg-green-600 text-white font-bold py-1 px-2 rounded mr-1 transition-colors duration-200">

### Accept

### </button>

### <button onClick={() => handleDecline(request.id)} className="bg-red-500 hover:bg-red-600 text-white font-bold py-1 px-2 rounded mr-1 transition-colors duration-200">

### Decline

### </button>

### </>

### )}

### <button onClick={() => toggleExpand(request.id)} className="bg-blue-500 hover:bg-blue-600 text-white font-bold py-1 px-2 rounded transition-colors duration-200">

### {expandedId === request.id ? 'Hide Details' : 'Show Details'}

### </button>

### </td>

### </tr>

### {expandedId === request.id && (

### <tr>

### <td colSpan="5" className="px-4 py-3">

### <div className="bg-indigo-50 p-4 rounded-lg shadow-inner">

### <h4 className="font-bold text-lg mb-3 text-indigo-800">Hall Details</h4>

### <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">

### <p><span className="font-semibold text-indigo-700">Name:</span> {request.hallDetails.name}</p>

### <p><span className="font-semibold text-indigo-700">Capacity:</span> {request.hallDetails.capacity}</p>

### <p><span className="font-semibold text-indigo-700">Price:</span> {request.hallDetails.price}</p>

### <p><span className="font-semibold text-indigo-700">Amenities:</span> {request.hallDetails.amenities.join(', ')}</p>

### </div>

### </div>

### </td>

### </tr>

### )}

### </React.Fragment>

### ))}

### </tbody>

### </table>

### </div>

### </div>

### </div>

### );

### }

### export default RequestManagement;

# CHAPTER 6

# CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project and the learning we learnt by taking over this project.

## CONCLUSION

## In conclusion, the Hall Management System is a comprehensive and advanced solution designed to revolutionize the management and booking of event spaces. This project integrates a sophisticated tech stack comprising React.js for a dynamic and responsive user interface, Tailwind CSS for modern and customizable styling, Spring Boot for a robust backend framework, REST APIs for efficient communication, and MySQL for secure and reliable data management. The system empowers users to explore a wide range of halls based on key criteria such as location, size, rating, and amenities, while also offering features to bookmark favorites and make reservations with ease. By streamlining the booking process and enhancing operational efficiency, the Hall Management System not only simplifies the task of finding and securing the ideal venue but also provides a user-friendly experience that meets both personal and professional event planning needs. Overall, this project exemplifies the effective integration of modern technologies to address the challenges of event space management and elevate the user experience.