CHAPTER 1

INTRODUCTION

1.1 PERSPECTIVE

The Venue Booking Management System is a comprehensive platform designed to streamline the process of managing venues for various events and gatherings. It provides a user-friendly interface for venue owners and event organizers to efficiently handle venue bookings and operations. Users can register and log in to the system to access its features, including viewing available venues, booking them for their events, and managing their bookings. Advanced search and filtering options enable users to find venues based on specific criteria such as location, capacity, facilities, and availability. Secure payment integration allows for online payments, ensuring a smooth booking process. Notifications keep users informed about booking confirmations, reminders, and updates. The system also generates reports and analytics on venue bookings, revenue, and occupancy rates, providing valuable insights for decision-making. An admin panel is available for administrators to manage venues, user accounts, bookings, and system settings. Overall, the Venue Management System enhances the efficiency, transparency, and convenience of booking and managing venues for events, benefiting both users and venue owners alike.

1.2 OBJECTIVE

The objective of the Venue Management System is to streamline venue booking processes and enhance user experience for both venue owners and event organizers, ultimately improving efficiency and convenience in managing venues for events. Through advanced features like user-friendly interfaces, secure payment integration, and comprehensive analytics, the system aims to optimize venue utilization and facilitate seamless event planning and execution. By providing a centralized platform for venue management, the system enables users to easily search for and book venues that meet

their specific requirements, reducing the time and effort involved in the booking process. Additionally, the system offers features such as real-time availability updates, automated notifications, and customizable booking options to cater to the diverse needs of users. With its intuitive design and robust functionality, the Venue Management System empowers venue owners to effectively showcase their properties, attract potential clients, and maximize revenue generation opportunities. Moreover, event organizers benefit from simplified venue selection, transparent pricing models, and efficient communication channels, enabling them to focus on delivering memorable events while ensuring a seamless experience for attendees. Through continuous refinement and innovation, the system aims to evolve with changing user preferences and industry trends, providing a scalable solution that meets the evolving needs of the venue management landscape.

1.3 SCOPE

The scope of the Venue Management System includes providing a platform for venue owners to list their properties, allowing users to search and book venues for various events. It involves managing booking requests, handling payments securely, and generating reports on venue utilization and revenue. Additionally, the system offers features for administrators to oversee venue operations, user accounts, and system settings. It aims to streamline the entire process of venue management, from booking to event execution, ensuring a seamless experience for both venue owners and event organizers.

CHAPTER 2
REQUIREMENT DESCRIPTION

2.1 FUNCTIONAL REQUIREMENTS

The functional requirement in Venue Management System is the collective information about what are the operations available in the system.

- Authentication: The system should offer authentication functionalities to ensure that only authorized users, such as venue owners and event organizers, can access the system.
- Venue Listing: It should provide venue owners with the capability to list their properties on the platform, including details such as location, capacity, facilities, and availability.
- Venue Search and Filtering: Users should be able to search and filter venues based on specific criteria such as location, capacity, facilities, availability, and price range.
- Booking Management: The system should facilitate the booking process, allowing users to view available dates, make bookings, manage reservations, and receive booking confirmations.

2.2 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirement describes about platform and physical resource required for building the student management system.

- Data Security: Venue details and user credentials will be stored securely to ensure data integrity and privacy.
- Performance: The system should be responsive and able to handle multiple users concurrently without significant slowdowns.
- Scalability: The application should be scalable to accommodate a growing number of venues and users over time.
- Accessibility: The system interface should be accessible to users with disabilities, following accessibility standards.
- Reliability: The system should have minimal downtime and be available for use whenever required by users.

CHAPTER 3

SYSTEM DESIGN

3.1 ARCHITECTURE DESIGN

The architectural diagram outlines the flow of interactions between users and the system. The process begins with users accessing the VMS platform, where they have the option to log in if they already have an account or proceed without logging in. If the user chooses to log in, they are directed to the login page where they can enter their credentials.

Upon successful login, the user gains access to the full features of the VMS platform, including venue search, booking management, and other administrative tasks. They can search for venues based on their criteria such as location, capacity, amenities, and availability.

For administrators, the system provides a dashboard where they can manage venue listings, update venue details, and handle booking requests. Administrators have the authority to approve or reject booking requests, communicate with venue owners, and oversee the overall operation of the VMS platform.

For users without accounts, they can’t browse available venues and view basic information without logging in.

Overall, the architectural diagram illustrates a user-centric approach, providing a seamless experience for both administrators and users interacting with the Venue Management System.

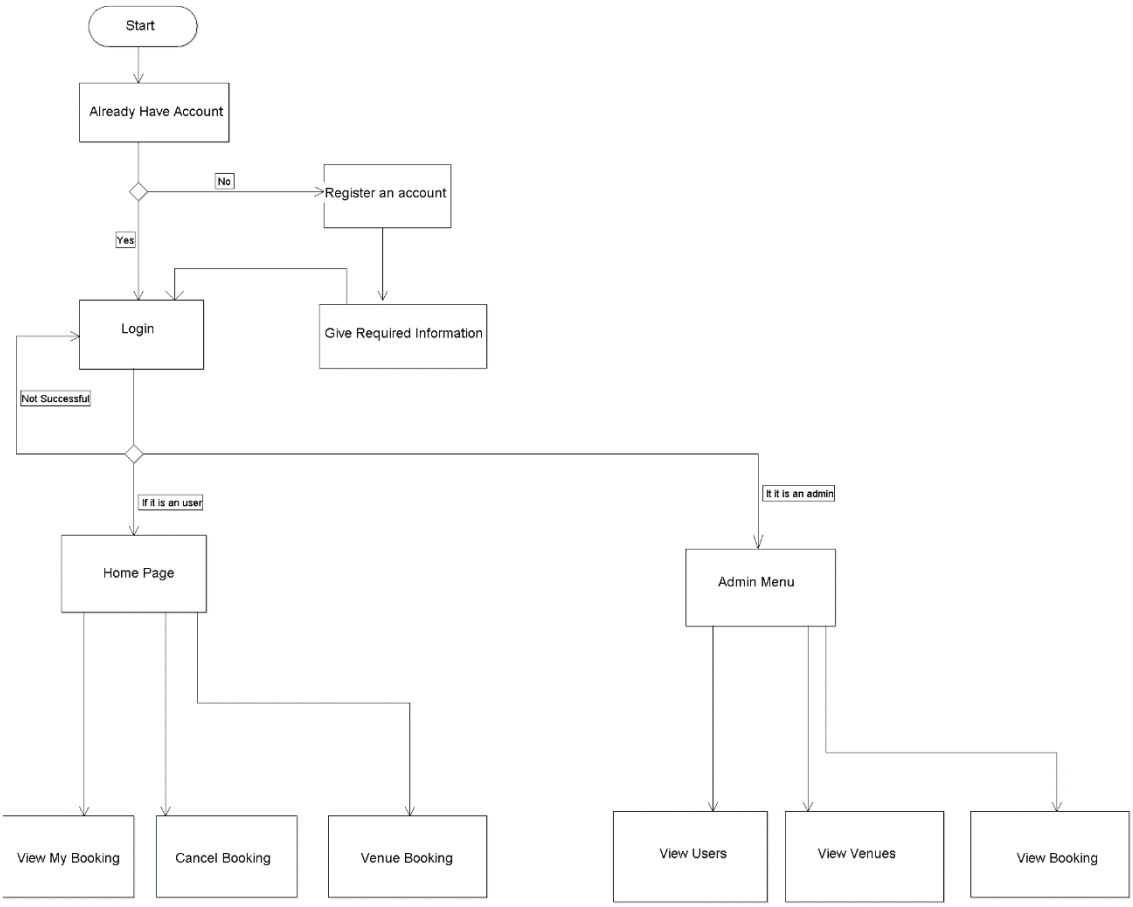

Figure 3.1: Architecture Diagram of Venue Management System

3.2 DESIGN COMPONENTS

3.2.1 Front End:

The Venue Management System uses Angular for developing interactive pages.

3.2.2 Back End:

Uses MongoDB for back end to store data.

3.3 DATABASE DESCRIPTION

Listed below gives a description of database document schemas used for Venue Management System.

3.3.1 User Structure

As shown in table 3.1, user structure contains the details of the donors.

Table 3.1: User Description

Attribute Name	Type	Width	Constraint(s)	Description
Username	String	20		Username
Gender	String	6		Gender of the user
Phone Number	Integer	10		Phone number of the user
Email-id	String	20		Mail id of the user
Password	String	20		Password of the user used for login details

3.3.2 Venue Structure

As shown in table 3.2, venue structure contains the details of the venues

Table 3.2: Venue Description

Attribute Name	Type	Width	Constraint(s)	Description
Name	String	20		Name of the venue
Seating Capacity	Integer	5		Seating Capacity of the venue
City	String	20		City of the venue
AC	String	3		AC availability
Parking Capacity	Integer	5		Parking capacity for the venue
Room Availability	Integer	5		Rooms available in the venue

3.3.3 Booking Structure

As shown in table 3.3, Booking structure contains the details of the bookings

Table 3.3: Venue Description

Attribute Name	Type	Width	Constraint(s)	Description
Username	String	20		Username of the users
Hall Name	String	20		Hall name of the venue
Date	String	10		Date of the booking

3.4 LOW LEVEL DESIGN

The following section illustrates the functionalities of the system. This includes login to the registration, forget password, delete my account, booking.

3.4.1 Login Details

Table 3.4 shows the login details of the application.

Table 3.4 Login Details

Files used	TypeScript/login.component.ts
Short Description	Allows the user to login to the application
Arguments	Username, Password
Return	Success/Failure in login
Pre-Condition	The user must have an account
Post-Condition	The home page will be displayed
Exception	Invalid Username and password
Actor	User

3.4.2 Registration Details

Table 3.5 shows the registration details of the application.

Table 3.5 Registration Details

Files used	TypeScript/register.component.ts
Short Description	Allows the user to register to the application

Arguments	Username, Password, Gender, Email, Phone Number
Return	Success/Failure in registration
Pre-Condition	The user must have a email id and phone number
Post-Condition	The login page will be displayed
Exception	Username already exists
Actor	User

3.4.1 Forget Password Details

Table 3.6 shows the forget password details of the application.

Table 3.6 Forget Password Details

Files used	TypeScript/forget.component.ts
Short Description	Allows the user to change the password
Arguments	Username, New Password
Return	Success/Failure in forget password
Pre-Condition	The user must have an account
Post-Condition	The login page will be displayed
Exception	Invalid Username
Actor	User

3.4.1 Delete My Account Details

Table 3.7 shows the delete my account details of the application.

Table 3.7 Delete My Account Details

Files used	TypeScript/delete.component.ts
Short Description	Allows the user to delete his account
Arguments	Username, Password
Return	Success/Failure
Pre-Condition	The user must have an account
Post-Condition	The landing page will be displayed
Exception	Invalid Username and password
Actor	User

3.5 USER INTERFACE DESIGN

3.5.1 Landing Activity

Figure 3.2 provide the interface for main activity.

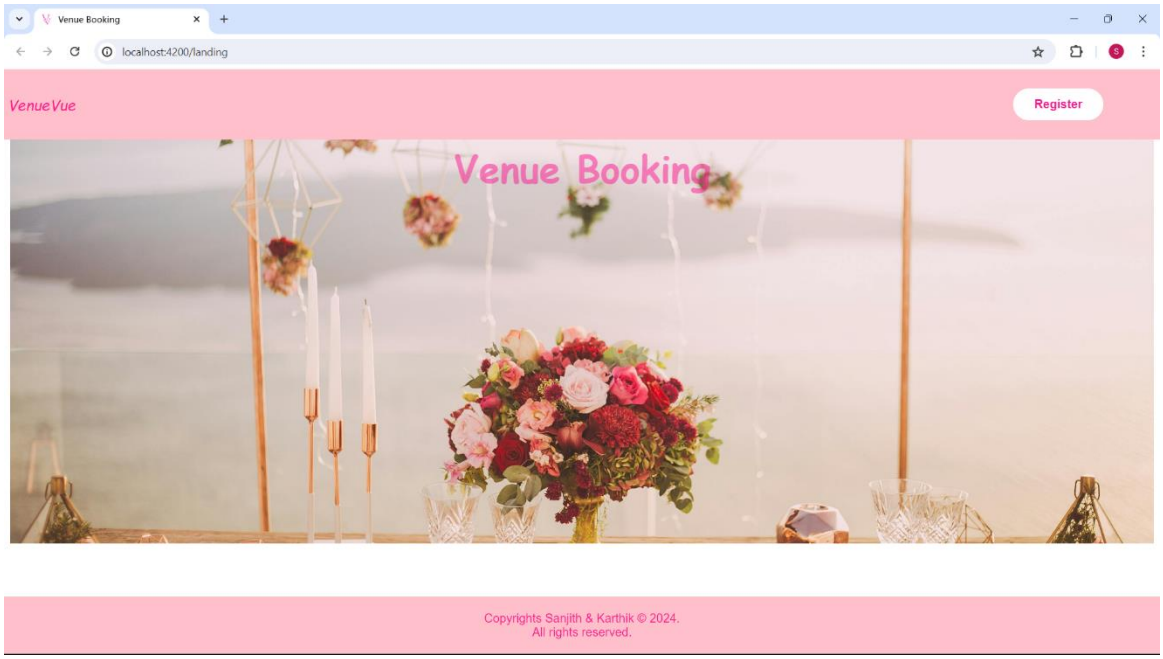

Figure 3.2: Landing layout of Venue Management System

3.5.2 Registration page

Figure 3.3 provide the interface for Registration page

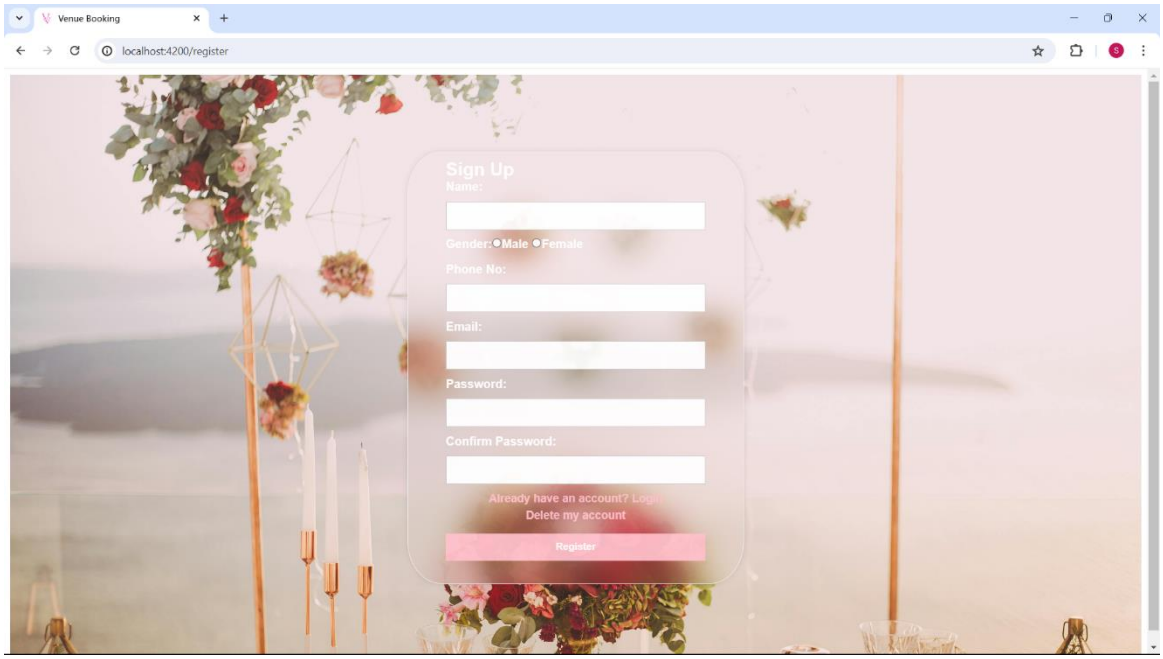

Figure 3.3: Registration Page

3.5.3 Login page

Figure 3.4 provide the interface for Login page

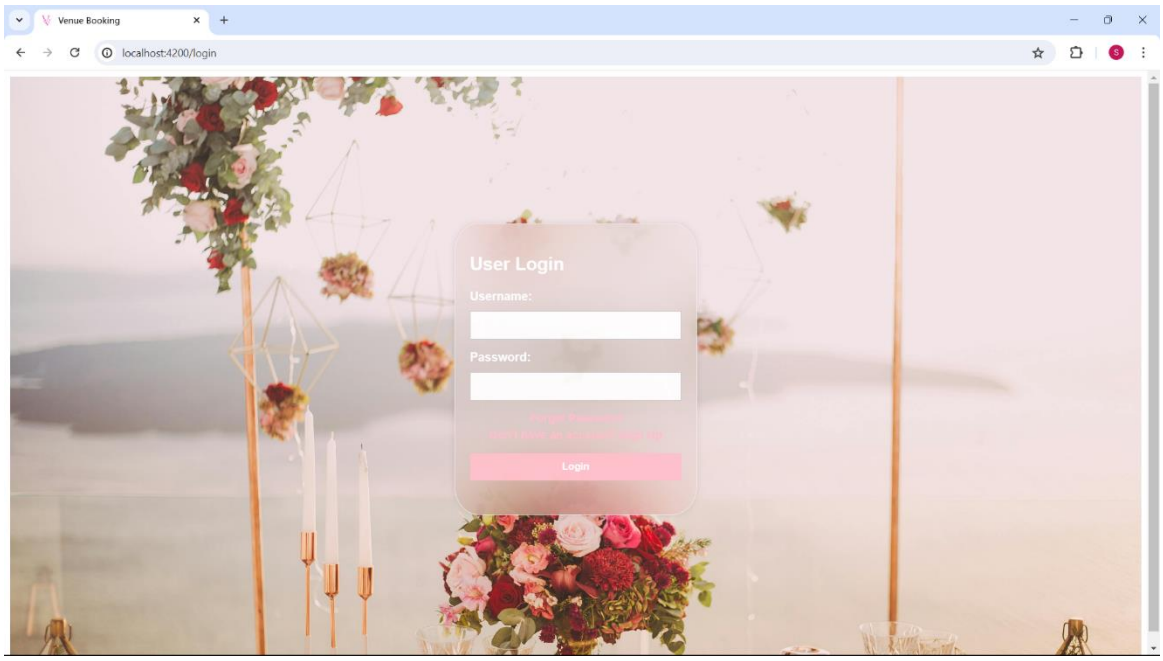

Figure 3.4: Login Page

3.5.4 Forget Password page

Figure 3.5 provide the interface for Forget Password page

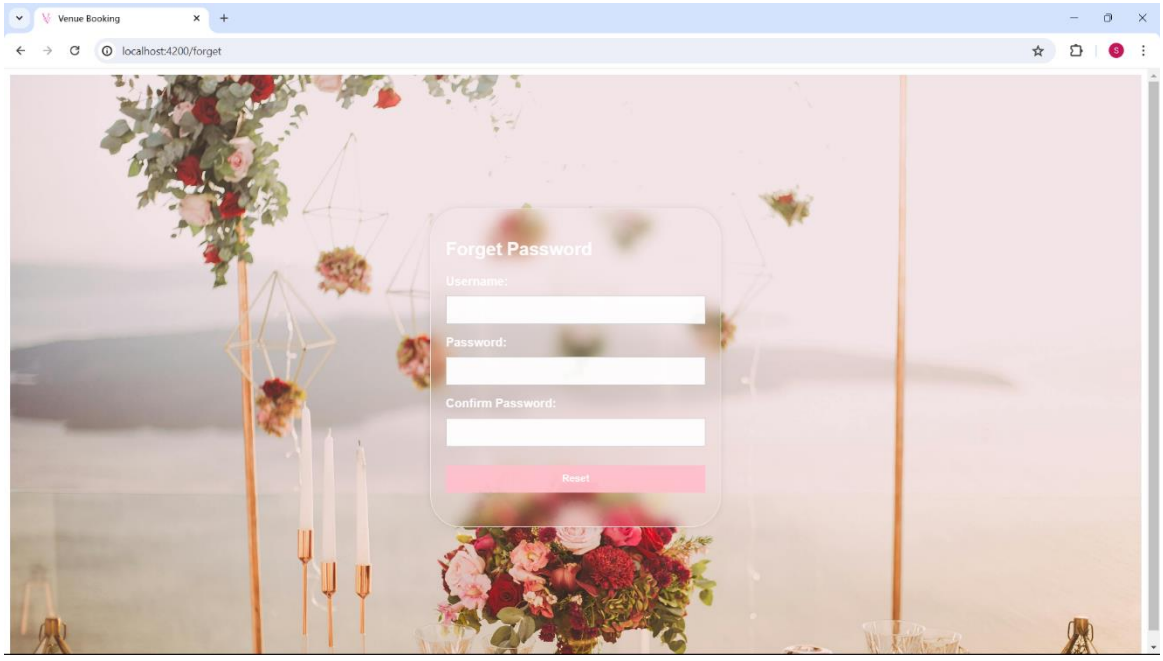

Figure 3.5: Forget Password Page

3.5.5 Delete My Account page

Figure 3.6 provide the interface for Delete My Account page

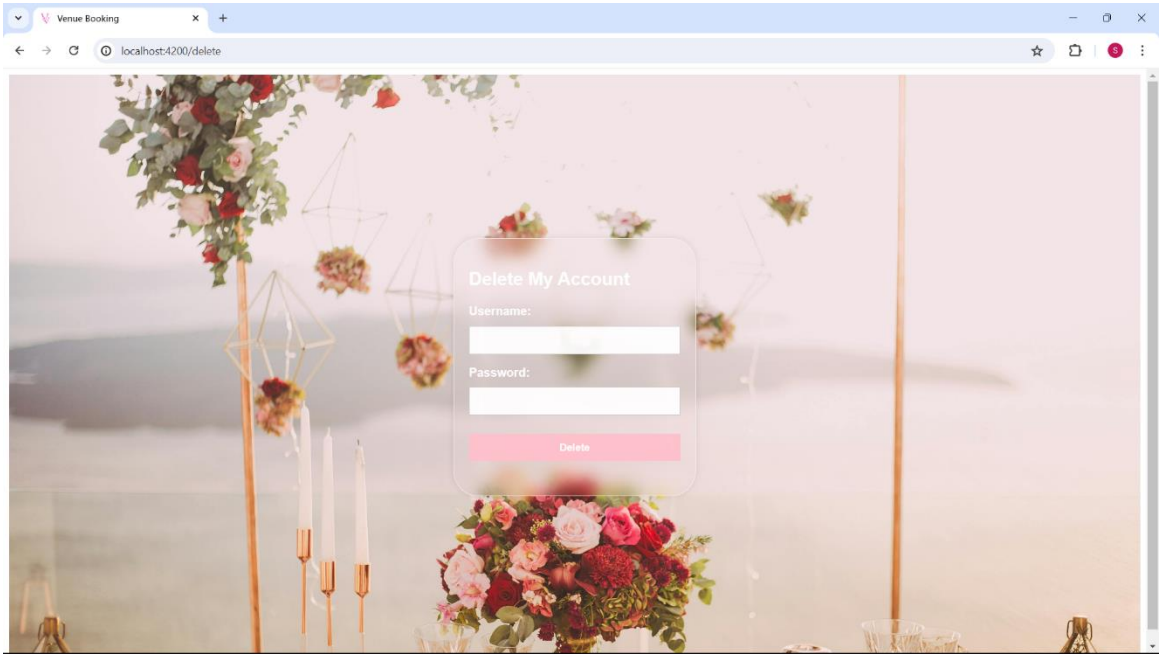

Figure 3.6: Delete My Account Page

3.5.6 Home page

Figure 3.7 provide the interface for Home page

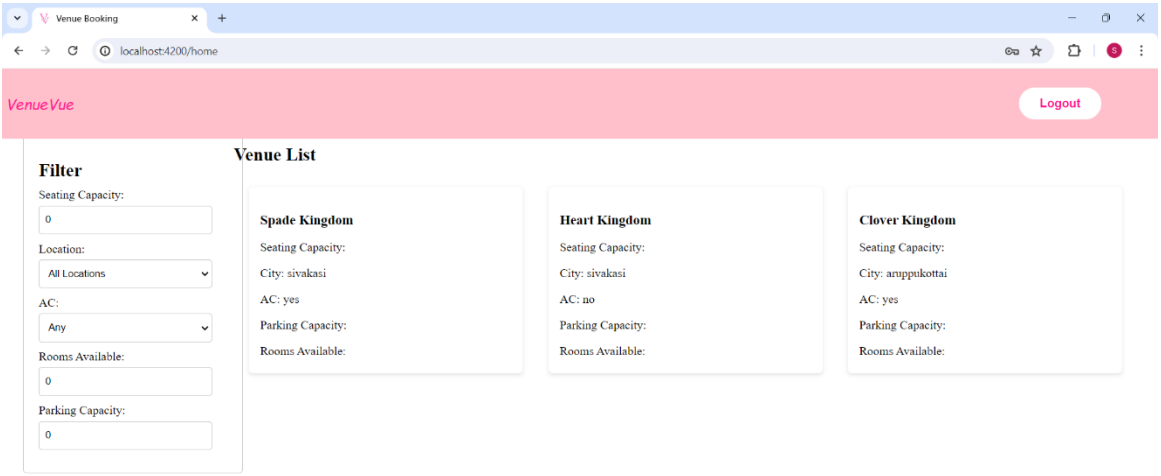

Figure 3.7: Home Page

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 LOGIN IMPLEMENTATION

The login credentials are obtained. If the credentials are OK, then the user is redirected to the homepage

GET username, password

IF username, password valid

RETURN homepage

ELSE

TOAST Invalid Credential

4.2 REGISTRATION IMPLEMENTATION

The form fields are obtained. If they are valid, then the user is added to the database.

GET requestFields

IF requestFields valid

RETURN added to db

ELSE

TOAST enter valid details

4.3 HOME IMPLEMENTATION

The form fields are obtained. If they are valid, then the venueDetails are listed.

GET venueDetails

IF filterApplied valid

RETURN details of the venues

ELSE

TOAST default list to be displayed

4.4 BOOKING IMPLEMENTATION

The form fields are obtained. If they are valid, then the details are stored in database.

GET venueAvailability

IF available

RETURN booking status

ELSE

TOAST other venue list to be displayed

4.5 CANCELLATION IMPLEMENTATION

The form fields are obtained. If they are valid, then the details are remove from database.

GET bookedDetails

IF available

RETURN cancellation status

ELSE

TOAST message is displayed

CHAPTER 5

RESULTS AND DISCUSSION

5.1 TEST CASES AND RESULTS

5.1.1 Test Cases and Results for Login function:

The Table 5.1, Table 5.2 shows that the possible test data for the both positive and negative test case given below, if the user is already having account then the output is true otherwise false.

Table 5.1: Positive Test Case and result for Login

Test Case ID	TC1
Test Case Description	It tests whether the given login details are valid or not
Test Data	sanjith,pass@1234
Expected Output	TRUE
Result	PASS

Table 5.2: Negative Test Case and result for Login

Test Case ID	TC2
Test Case Description	It tests whether the given login details are valid or not
Test Data	Sanjith,pass
Expected Output	FALSE
Result	PASS

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT(S)

The Venue Management System represents a significant step forward in streamlining venue booking processes and enhancing the overall user experience. By providing a centralized platform for venue owners and event organizers to connect and collaborate, the system fosters efficiency and convenience in managing venues for various events. Moving forward, potential enhancements could include the integration of advanced analytics capabilities to provide insights into venue utilization and user preferences, allowing for more informed decision-making and strategic planning. Additionally, future iterations of the system could incorporate features such as automated scheduling algorithms to optimize venue allocation and minimize conflicts. Furthermore, enhancing the system's communication capabilities, such as real-time messaging and notification functionalities, could further improve collaboration and coordination between venue owners and event organizers. Overall, the Venue Management System holds great potential for continued growth and evolution, offering opportunities to further streamline venue management processes and deliver enhanced value to users across the board.

APPENDIX – A

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENT:

Processor:	Intel Core i5 or equivalent AMD
RAM:	8GB or higher
Storage:	SSD with at least 256GB of free space
Network:	Ethernet or Wi-Fi connectivity

SOFTWARE REQUIREMENT:

Operating System:	Any (64bits)
DBMS:	MongoDB (v7.0.6)
Text Editor:	Visual Studio Code
Angular Version:	15.1 and above
Node.js Version	v20.11.1

APPENDIX – B

Commands and Comments

Angular:

Create a new Angular project:

ng new project-name

Create a new component:

ng generate component component-name

Create a new service:

ng generate service service-name

Run the Angular development server:

ng serve

Build the Angular project for production:

ng build --prod

Generate a new module:

ng generate module module-name

Create a new directive:

ng generate directive directive-name

Create a new pipe:

ng generate pipe pipe-name

Node.js:

Initialize a new Node.js project:

npm init

Install dependencies:

npm install package-name

Run a Node.js server:

node server.js

Install Node.js packages globally:

npm install -g package-name

Install Angular CLI globally:

npm install -g @angular/cli

Run tests in Angular:

ng test

Create a new Angular project with routing:

ng new project-name –routing

Setting Host IP for Angular Development Server

ng serve --host your-ip-address

APPENDIX – C

SOURCE CODE

landing.component.html

```
<header class="header">
    <nav>
      <a href="/landing" class="logo">VenueVue</a>
      <ul>
        <li>
          <a href="/register" class="header-items">Register</a>
        </li>
      </ul>
    </nav>
  </header>
  <section class="picture">
    <div class="text">
      <br /><br />
      <h1>Venue Booking</h1>
      <br />
      <h2><span class="t1"></span></h2>
    </div>
  </section>
  <section class="form">
    <form name="f1" class="intro">
      <table>
        <tr>
          <td class="box">
            <select name="city" id="city">
              <option value="City">City</option>
              <option value="sivakasi">Sivakasi</option>
              <option value="virudhunagar">Virudhunagar</option>
              <option value="aruppukottai">Aruppukottai</option>
```

```
            </select>
          </td>
          <td class="box">
            <input type="date" name="date" id="dateInput" />
          </td>
          <td class="box"><input type="button" value="Search" /></td>
        </tr>
      </table>
    </form>
  </section>
  <footer>
    <p>Copyrights Sanjith & Karthik &amp;copy; 2024.<br />All rights reserved.</p>
  </footer>
```

register.component.html

```
<form
    #form="ngForm"
    (ngSubmit)="onSubmit(form)"
    name="f1"
    autocomplete="off"
  >
    <h2>Sign Up</h2>
    <div>
      <label for="username">Name:</label>
      <input
        type="text"
        id="username"
        name="username"
        ngModel
        #username="ngModel"
        required
      />
    </div>
    <div>
      <label for="gender">Gender:</label>
      <input
        type="radio"
        id="gender"
        name="gender"
        value="Male"
        ngModel
```

```
    #gender="ngModel"
    required
  />Male
  <input
    type="radio"
    id="gender"
    name="gender"
    value="Female"
    ngModel
    #gender="ngModel"
    required
  />Female
</div>
<div>
  <label for="phone">Phone No:</label>
  <input
    type="text"
    id="phone"
    name="phone"
    ngModel
    #phone="ngModel"
    required
  />
</div>
<div>
  <label for="email">Email:</label>
  <input
    type="text"
    id="email"
    name="email"
    ngModel
    #email="ngModel"
    required
  />
</div>
<div>
  <label for="password">Password:</label>
  <input
    type="password"
    id="password"
    name="password"
    ngModel
    #password="ngModel"
    required
  />
</div>
```

```
      <div>
        <label for="cpassword">Confirm Password:</label>
        <input
          type="password"
          id="cpassword"
          name="cpassword"
          ngModel
          #cpassword="ngModel"
          required
        />
      </div>
      <a href="/login">Already have an account? Login</a>
      <a href="/delete">Delete my account</a>
      <div *ngIf="show" class="validate-msg">{{ validateMsg }}</div>
      <div>
        <button type="submit" (click)="validate()">Register</button>
      </div>
    </form>
```

register.component.ts

```
import { Component } from '@angular/core';
import { NodeUtilityService } from '../node-utility.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrl: './register.component.css',
})
export class RegisterComponent {
  msg: string = '';
  validateMsg: string = '';
  show: boolean = false;
  constructor(private util: NodeUtilityService, private router: Router) {}
  onSubmit(form: any) {
    this.util
      .register(
        form.value.username,
        form.value.gender,
```

```
      form.value.phone,
      form.value.email,
      form.value.password
    )
    .subscribe((data) => {
      if (data.status) {
        this.msg = data.message;
        this.router.navigateByUrl('/login');
      } else {
        this.msg = data.message;
        this.router.navigateByUrl('/register');
      }
    });
}
validate() {
  const userName = <HTMLInputElement>document.querySelector('#username');
  const passWord = <HTMLInputElement>document.querySelector('#password');
  const gender = <HTMLInputElement>document.querySelector('#gender');
  const email = <HTMLInputElement>document.querySelector('#email');
  const cpassWord = <HTMLInputElement>document.querySelector('#cpassword');
  const phone = <HTMLInputElement>document.querySelector('#phone');
  var u1 = userName.value;
  var p1 = passWord.value;
  var cp1 = cpassWord.value;
  var g = gender.value;
  var e = email.value;
  var p = phone.value;
  if (u1.length == 0) {
    this.validateMsg = 'Enter a Valid Username';
    this.show = true;
    return false;
  }
  if (p.length == 0) {
    this.validateMsg = 'Phone Number Missing';
```

```
      this.show = true;
      return false;
    }
    if (p.length != 10) {
      this.validateMsg = 'Enter the Correct Mobile No';
      this.show = true;
      return false;
    }
    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(e)) {
      this.validateMsg = 'Enter a valid email address';
      this.show = true;
      return false;
    }
    if (p1.length < 8 || cp1.length < 8) {
      this.validateMsg = 'Enter a strong password';
      this.show = true;
      return false;
    }
    if (p1 != cp1) {
      this.validateMsg = "Password doesn't match";
      this.show = true;
      return false;
    }
    this.show = false;
    return true;
  }
}
```

login.component.ts

```
import { Component } from '@angular/core';
import { NodeUtilityService } from '../node-utility.service';
import { Router } from '@angular/router';
```

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.css',
})
export class LoginComponent {
  username: string = '';
  password: string = '';
  user: string = '';
  msg: string = '';
  show: boolean = false;
  validateMsg: string = '';
  constructor(private util: NodeUtilityService, private router: Router) {}
  onSubmit(form: any) {
    this.util
      .login(form.value.username, form.value.password)
      .subscribe((data) => {
        if (data.status) {
          localStorage.setItem('user', form.value.username);
          this.msg = data.message;
          this.show = false;
          this.router.navigateByUrl('/home');
        } else {
          this.msg = data.message;
          this.show = true;
          this.validateMsg = 'Invalid Credential';
          this.router.navigateByUrl('/login');
        }
      });
  }
  validate() {
    const userName = <HTMLInputElement>document.querySelector('#username');
    const passWord = <HTMLInputElement>document.querySelector('#password');
    var u1 = userName.value;
```

```
    var p1 = passWord.value;
    if (u1.length == 0) {
      this.validateMsg = 'Username is Missing';
      this.show = true;
      return false;
    }
    if (p1.length == 0) {
      this.validateMsg = 'Password is Missing';
      this.show = true;
      return false;
    }
    this.show = false;
    return true;
  }
}
```

login.component.html

```
<form
      #form="ngForm"
      (ngSubmit)="onSubmit(form)"
      name="f1"
      autocomplete="off"
    >
      <h2>User Login</h2>
      <div>
        <label for="username">Username:</label>
        <input
          type="text"
          id="username"
          name="username"
          ngModel
          #username="ngModel"
          required
```

```
      />
    </div>
    <div>
      <label for="password">Password:</label>
      <input
        type="password"
        id="password"
        name="password"
        ngModel
        #password="ngModel"
        required
      />
    </div>
    <a href="/forget">Forget Password</a>
    <a href="/register">Don't have an account? Sign Up</a>
    <div *ngIf="show" class="validate-msg">{{ validateMsg }}</div>
    <div>
      <button type="submit" (click)="validate()">Login</button>
    </div>
  </form>
```

home.component.html

```
<div class="header">
  <!-- Include the header component -->
  <app-header></app-header>
</div>

<div class="content">
  <div class="filter">
    <!-- Include the filter component -->
    <app-filter (filterChanged)="handleFilterChange($event)"></app-filter>
  </div>

  <div class="venue-list">
```

```
    <!-- Include the venue component -->
    <app-venue [filterData]="filterData"></app-venue>
  </div>
</div>
```

home.component.ts

```
user: string = '';
  constructor(private router: Router) {
    let u: any = localStorage.getItem('user');
    this.user = u;
    if (u == null || u == '') {
      this.router.navigateByUrl('landing');
    }
  }
  logout() {
    localStorage.removeItem('user');
    this.router.navigateByUrl('login');
  }
  filterData: any = {}; // Initialize an object to hold filter data

  // Define a method to handle filter changes emitted by the FilterComponent
  handleFilterChange(event: any): void {
    // Assign the filter data emitted by the FilterComponent to the filterData object
    this.filterData = event;
  }
```

header.component.html

```
<!DOCTYPE html>
<body>
  <header class="header">
    <nav>
      <a href="/landing" class="logo">VenueVue</a>
      <ul>
        <li><a href="#" (click)="logout()" class="header-items">Logout</a></li>
```

```
      </ul>
    </nav>
  </header>
</body>
```

header.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrl: './header.component.css',
})
export class HeaderComponent {
  constructor(private router: Router) {}
  logout() {
    localStorage.removeItem('user');
    this.router.navigateByUrl('login');
  }
}
```

filter.component.html

```
<div class="filter">
  <h2>Filter</h2>
  <div class="filter-options">
    <label>Seating Capacity:</label>
    <input type="number" [(ngModel)]="seatingCapacity" />
    <label>Location:</label>
    <select [(ngModel)]="selectedLocation" (click)="applyFilters()">
      <option value="">All Locations</option>
      <option value="sivakasi">Sivakasi</option>
      <option value="aruppukottai">Aruppukottai</option>
      <!-- Add more locations as needed -->
```

```
    </select>
    <label>AC:</label>
    <select [(ngModel)]="selectedAc">
      <option value="">Any</option>
      <option value="yes">Yes</option>
      <option value="no">No</option>
    </select>
    <label>Rooms Available:</label>
    <input type="number" [(ngModel)]="selectedRoomsAvailable" />
    <label>Parking Capacity:</label>
    <input type="number" [(ngModel)]="selectedParkingCapacity" />
    <!-- Add more filter options as needed -->
  </div>
</div>
```

filter.component.ts

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-filter',
  templateUrl: './filter.component.html',
  styleUrl: './filter.component.css',
})
export class FilterComponent {
  @Output() filterData: EventEmitter<any> = new EventEmitter();

  seatingCapacity: number = 0;
  selectedLocation: string = '';
  selectedAc: string = '';
  selectedRoomsAvailable: number = 0; // Added for rooms available filter
  selectedParkingCapacity: number = 0; // Added for parking capacity filter

  constructor() {}
```

```
  applyFilters(): void {
    // Emit filter values
    this.filterData.emit({
      seatingCapacity: this.seatingCapacity,
      selectedLocation: this.selectedLocation,
      selectedAc: this.selectedAc,
      selectedRoomsAvailable: this.selectedRoomsAvailable,
      selectedParkingCapacity: this.selectedParkingCapacity,
    });
  }
}
```

venue.component.html

```
<h2>Venue List</h2>
<div class="venue-list">
  <div class="venue-card" *ngFor="let venue of filteredVenues">
    <h3>{{ venue.name }}</h3>
    <p>Seating Capacity: {{ venue.seatingCapacity }}</p>
    <p>City: {{ venue.city }}</p>
    <p>AC: {{ venue.ac }}</p>
    <p>Parking Capacity: {{ venue.parkingCapacity }}</p>
    <p>Rooms Available: {{ venue.roomsAvailable }}</p>
    <!-- Add more venue details as needed -->
  </div>
</div>
```

venue.component.ts

```
import { Component, Input } from '@angular/core';
import { NodeUtilityService } from '../node-utility.service';

@Component({
  selector: 'app-venue',
  templateUrl: './venue.component.html',
  styleUrl: './venue.component.css',
})
```

```
export class VenueComponent {
  @Input() filterData: any;
  venues: any[] = [];
  filteredVenues: any[] = [];
  seatingCapacity: number = 0;
  selectedLocation: string = '';
  selectedAc: string = '';
  selectedRoomsAvailable: number = 0;
  selectedParkingCapacity: number = 0;

  constructor(private util: NodeUtilityService) {}

  ngOnInit(): void {
    // Fetch venues from the service when component initializes
    this.getVenues();
  }
  ngOnChanges(): void {
    // Check if filterData exists and apply filters accordingly
    if (this.filterData) {
      this.applyFilters(this.filterData);
    }
  }

  getVenues(): void {
    // Call venue service to fetch venues
    this.util.getVenues().subscribe((venues) => {
      this.venues = venues;
      // Initially, display all venues
      this.filteredVenues = venues.item;
    });
  }

  applyFilters(filterData: any): void {
    // Update filter values
```

```
this.seatingCapacity = filterData.seatingCapacity;
this.selectedLocation = filterData.selectedLocation;
this.selectedAc = filterData.selectedAc;
this.selectedRoomsAvailable = filterData.selectedRoomsAvailable;
this.selectedParkingCapacity = filterData.selectedParkingCapacity;

// Apply filters to the venues list
this.filteredVenues = this.venues.filter((venue) => {
 let match = true;
 if (
  this.seatingCapacity &&
  venue.seatingcapacity < this.seatingCapacity
 ) {
  match = false;
 }
 if (this.selectedLocation && venue.city !== this.selectedLocation) {
  match = false;
 }
 if (this.selectedAc && venue.ac !== this.selectedAc) {
  match = false;
 }
 if (
  this.selectedRoomsAvailable &&
  venue.roomsavailable < this.selectedRoomsAvailable
 ) {
  match = false;
 }
 if (
  this.selectedParkingCapacity &&
  venue.parkingcapacity < this.selectedParkingCapacity
 ) {
  match = false;
 }
 // Add more conditions for other filters as needed
```

```
      return match;
    });
  }
}
```

REFERENCES

1. https://angular.io/docs

2. https://nodejs.org/docs/latest/api/

3. https://www.mongodb.com/docs/