Cozy Haven Stay

Problem statement:

The **Cozy Haven Stay** Hotel Booking System is a comprehensive web application that allows users to search, book, and manage hotel reservations online. It caters to both guests and hotel administrators, providing a user-friendly experience for travelers and efficient management tools for hotel staff.

Scope

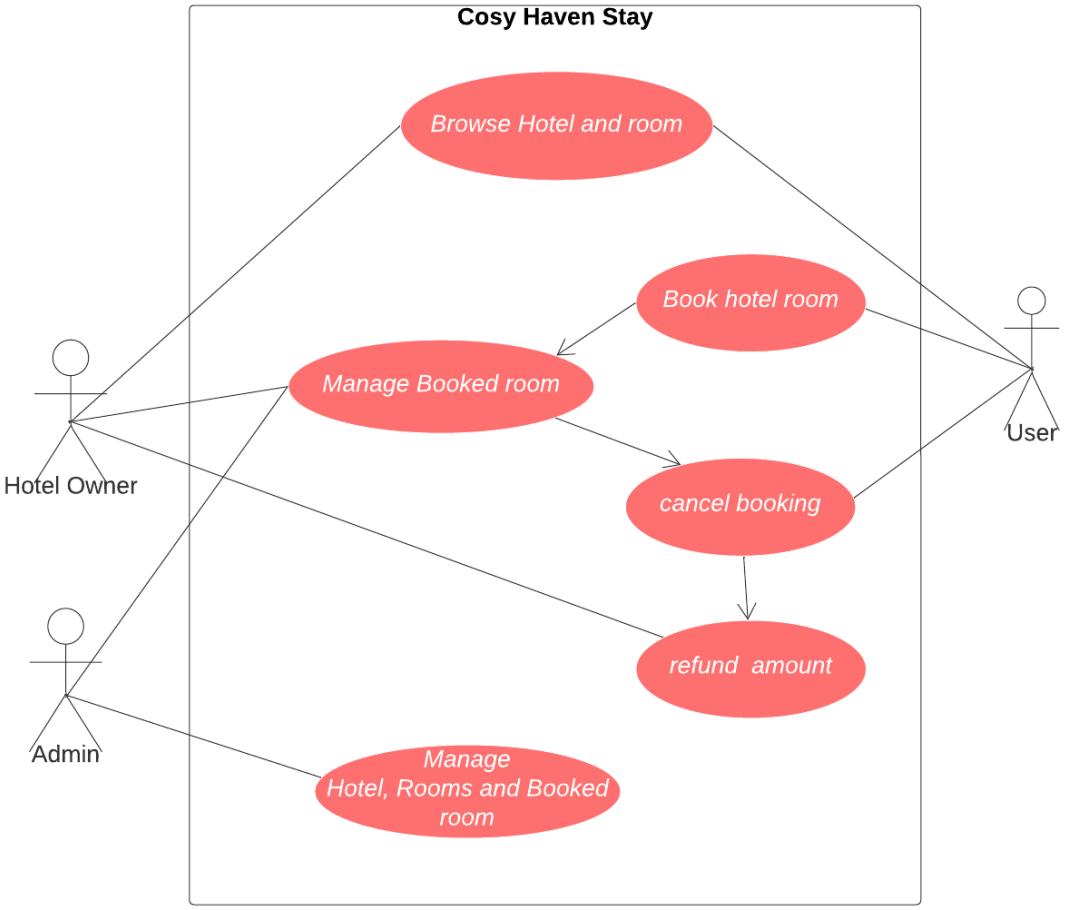
1. **User Registration and Authentication:** Provide user registration and secure login for passengers.
2. **Search and Browse Hotels:**
    - Provide users with the ability to search for hotels based on location, dates, and preferences like no. of room and no. of person includes adults and child.
    - Display detailed information about available hotels, including pricing, amenities, images, bed preference (single or double) and room size.
3. **Booking and Reservation:**
    - Enable users to select rooms, specify check-in and check-out dates, no. of room and no. of person includes adults, child and make reservations.
    - Manage bookings and view reservation details.
4. **User Reviews and Ratings:**
    - Allow guests to leave reviews and ratings for hotels they have stayed in.
5. **Hotel Management:**
    - Provide hotel administrators with tools to manage hotel listings, room availability, and bookings.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Reference Link: https://www.agoda.com/en-in

Use case Diagram:



Use Cases:

Actor: User

- Use Case: User Registration and login.
- Use Case: Search and view hotels.
- Use Case: Select hotel and make room reservation.
- Use Case: Payment Processing.
- Use Case: Cancel reservation.
- Use Case: View reservation history.

Actor: Hotel Owner (Good to have)

- Use Case: Log in as hotel owner.
- Use Case: Manage the rooms.
- Use Case: Manage Bookings.
- Use Case: Logout.

Actor: Administrator

- Use Case: Log In.
- Use Case: User Management.
- Use Case: Manage Hotel Listings.

- Use Case: Manage Bookings of all hotels.
- Use Case: Logout.

System: Security and Authentication

- Use Case: Authenticate User.

System: Database Management

- Use Case: Store Hotel, room, and schedule Information.
- Use Case: Store reservation Information.
- Use Case: Store User and Flight Operator Information.

Development Process:

1. **User and Flight Owner, Admin Registration:**
   - User, Hotel Owner, Admin can create accounts, providing personal details (name, gender, contact number, address, etc.)
   - The system validates the information and creates user profiles.
   - Users and Hotel Owner, Admin log in using their credentials (username/email and password) and navigate to their corresponding dashboard.
2. **User's Dashboard:**
   - Guests can search for hotel room by specifying the duration of stay, location of stay, and number of rooms.
     - To fill location name should be typed and filled with auto suggestions or selected from the dropdown list.
     - To fill duration of stay date should be selected from calendar. Date must be selected from current date to vacate date and it should be validated.
   - The user can select the hotel from the list. The system should display a list of hotels with facilities in hotel like Dining, parking, free Wi-Fi, Room Service, Swimming pool, Fitness center.
   - By selecting a hotel from the list, system list out all rooms in hotel with following details Room size, bed sizes (single bed, double bed, king size), maximum number of people accommodate in room, base fare, and calculate total fare of room by number people stay in room. Should show status of available on your date.
     - If a room contains a single bed maximum 2 people can stay in room, from 2nd person additionally charged with 40% of base fare, if person age is above 14 else additionally charged with 20%.
     - If a room contains a double bed maximum 4 people can stay in room, from 3rd person additionally charged with 40% of base fare, if person age is above 14 else additionally charged with 20%.
     - If a room contains a double bed maximum 6 people can stay in room, from 5th person additionally charged with 40% of base fare, if person age is above 14 else additionally charged with 20%.
   - Guests can book the rooms as they selected and can be viewed in manage booking.
   - Guests can view the history of booking and manage their booking history.
   - They can cancel bookings and request refunds as needed.

3. **Hotel owner Dashboard:**
    - Hotel owners can add, edit, or remove rooms. adding new room should have following details:
        - Room size (Room size: 70 m²/753 ft²), bed sizes (single bed, double bed, king size), maximum number of people accommodate in room, base fare, AC or NonAC.
    - Hotel owners can view the tickets booked by the user.
    - Hotel owners can refund amount of booked tickets which are cancelled by the user.
4. **Administrators Dashboard**
    - Administrator can manage(delete) user accounts and Hotel owner accounts.
    - Administrator can manage the room reservation in hotel by the user.
    - Administrators can access a dashboard to manage Flight routes.
5. **Security and Compliance:**
    - User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

- User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
- Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
- Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
    1. Extract the token from the request header.
    2. Verify the token's signature using the server's secret key.
- Payload Verification:
    1. Decode the token and extract user information.
    2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI) to develop the services. • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning. • Implement security to allow/deny Create, Update, Delete operations. • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical). • Database connections and web service URLs should be configurable. • Implement Unit Test Project for testing the API. • Implement JWT for Security. • Implement Logging. • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for

		readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, add Hotel, Room and room booking by user.**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate user password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, data binding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features:

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application.
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.