

Project Title

House Hunt: Finding Your Perfect Rental Home

Team Members:

Shaik Mohammed Ali

Thokala Bhanu Prakash

Mulam Reddy Venkata RaviTeja Reddy

Peram Surendra Reddy

Gudimetla Siva

Table of contents

Introduction

Project Overview

Architecture

Setup Instructions

Folder Structure

Running the Application

API Documentation

Authentication

User Interface

Testing

Known Issues

Future Enhancements

Introduction

In today's fast-paced urban landscape, finding the perfect rental home can be a daunting challenge. "House Hunt: Finding Your Perfect Rental" is an innovative application designed to simplify the rental search process for individuals and families. This project aims to connect potential renters with a diverse range of properties tailored to their specific needs, preferences, and budget.



With an intuitive user interface, House Hunt allows users to explore listings based on various criteria, including location, price, number of bedrooms, and amenities. The application provides detailed property information, high-quality images, and user-friendly navigation, making the search for a new home both efficient and enjoyable.

In addition to facilitating property searches, the platform offers valuable resources, such as neighbourhood insights, pricing trends, and tips for securing rental agreements. By leveraging technology and data-driven insights, House Hunt seeks to empower renters, helping them make informed decisions and ultimately find their ideal living space.

Project Overview

Purpose and Features

The objective of the **House Hunt** project is to develop a comprehensive and user-friendly platform that fosters seamless interactions among property owners, renters, and administrators. By streamlining essential processes, this system aims to significantly enhance the overall rental experience.

Key features of the **House Hunt** platform include:

1. Efficient User Registration and Login:

- Simple onboarding process for renters and property owners, allowing quick access to the platform's features.

2. Property Management:

- Comprehensive tools for property owners to create, update, and manage their listings, including detailed descriptions and high-quality images.

3. Booking Functionalities:

- Streamlined booking process for renters to reserve properties easily, with real-time availability updates.

4. History Tracking Feature:

- Enables users to view their booking history and transaction details, promoting transparency and accountability.

5. Admin Oversight:

- Robust admin capabilities to monitor user accounts, property listings, and rental transactions, ensuring smooth operations and quick resolution of issues.

6. Enhanced Accessibility:

- The platform aims to make rental properties more accessible, helping renters find suitable homes effortlessly.



7. **Transparency in Pricing:**

- Clear and upfront pricing information for all listings, ensuring renters understand costs associated with each property, fostering trust and informed decision-making.

8. **Organized Transactions:**

- Implementation of organized transaction records to build trust among all participants.

9. **Community Building:**

- Foster a trustworthy environment where property owners and renters can interact confidently and constructively.

By integrating these features, **House Hunt** seeks to create a collaborative and efficient ecosystem for all stakeholders involved, ultimately enhancing the overall rental experience and ensuring satisfaction for users, property owners, and administrators alike.

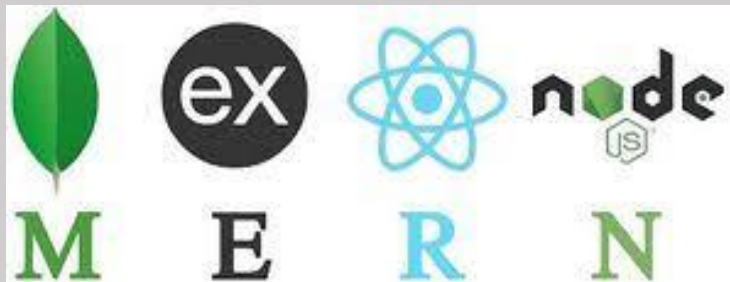
Architecture

Frontend Architecture

- **Component-Based Architecture:** The frontend is built using a component-driven approach with React. Each page and functionality (e.g., login, rental listings) is split into modular, reusable components.
 - **Container Components:** Handle state management and business logic.
 - **Presentation Components:** Focus solely on the UI.
- **State Management:** Redux is used for managing global application state, such as user authentication and rental data. Each state is divided into slices, following the Redux Toolkit convention.
- **Routing:** React Router is used to handle client-side routing. Routes are divided into public (for customers) and private routes (for admin). Admin-specific pages like rental approvals are secured by JWT-based authentication checks.
- **Error Handling and Loading States:** Error boundaries and loading indicators are implemented for a smoother user experience during API calls and page transitions.

Backend Architecture

- **Layered Architecture:** The backend follows a layered architecture pattern:
 - **Route Layer:** Defines endpoints such as `/api/rentals`, `/api/houses`, and `/api/admin`. Each route is protected using JWT middleware for admin actions like approvals.
 - **Controller Layer:** Handles the core business logic. For example, the `approveRental` function checks the admin's credentials before updating the rental's approval status.
 - **Service Layer:** This handles the interaction with the database using Mongoose. It abstracts away the direct database logic and provides reusable services, like `RentalService` for managing rentals.
 - **Model Layer:** Mongoose schemas and models define the structure of the documents in MongoDB, ensuring validation and consistency across the app.
- **Middleware:** Middleware's such as authentication (JWT verification), error handling, and input validation (using libraries like `express-validator`) are used.
- **Security:** `Helmet.js` is used for securing HTTP headers. All API endpoints are secured, and rate-limiting is applied to critical routes like login.



Database Architecture (MongoDB with Mongoose)

- **Document-Oriented Architecture:** The MongoDB database is organized using collections for each entity:
 - **Admin:** Stores admin credentials, JWT tokens, and activity logs.
 - **Rentals:** Stores information related to rental properties, including status (approved or pending).
 - **Houses:** Stores data about the houses available for rent.
- **Data Relationships:**
 - Rentals reference houses using MongoDB ObjectIDs, ensuring a clear relationship between available houses and their rental status.
 - The admin collection references approvals made by the admin for auditing purposes.
- **Schema Design:**
 - **House Schema:** Contains fields like address, owner, price, and isApproved.
 - **Rental Schema:** Contains fields like rentalDate, renter, house, and approvalStatus.
- **Database Interactions:** Mongoose is used for interacting with MongoDB. For complex operations, such as filtering rentals by approval status, Mongoose aggregation pipelines are used.

Setup Instructions

Prerequisites:

To successfully develop a full-stack house rental web application using React.js, Node.js, Express.js and MongoDB, several essential prerequisites should be considered. Below are the key requirements for undertaking this project:

Node.js and npm:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

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

Installation instructions: <https://nodejs.org/en/download/package-manager/>

npm init

Express.js:

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

npm install express

MongoDB:

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: <https://www.mongodb.com/try/download/community>

Installation instructions: <https://docs.mongodb.com/manual/installation/>

Moment.js:

Moment.js is a JavaScript package that makes it simple to parse, validate, manipulate, and display date/time in JavaScript. Moment.js allows you to display dates in a human-readable

format based on your location. Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://momentjs.com/>

React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

Antd:

Ant Design is a React.js UI library that contains easy-to-use components that are useful for building interactive user interfaces. It is very easy to use as well as integrate. It is one of the smart options to design web applications using react.

Follow the installation guide: <https://ant.design/docs/react/introduce>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Reactjs to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard.

For making better UI we have also used some libraries like material UI and bootstrap.

Install Dependencies:

- Navigate into the cloned repository directory:

```
cd Backend
```

- Install the required dependencies by running the following commands:

```
cd House
```

```
npm install
```

```
cd ../backend
```

```
npm install
```

Start the Development Server:

- To start the development server, execute the following command:

npm start

- The house rent app will be accessible at <http://localhost:4000>

You have successfully installed and set up House Hunt :Finding Your Perfect Rental Home on your local machine. You can now proceed with further customization, development, and testing as needed.

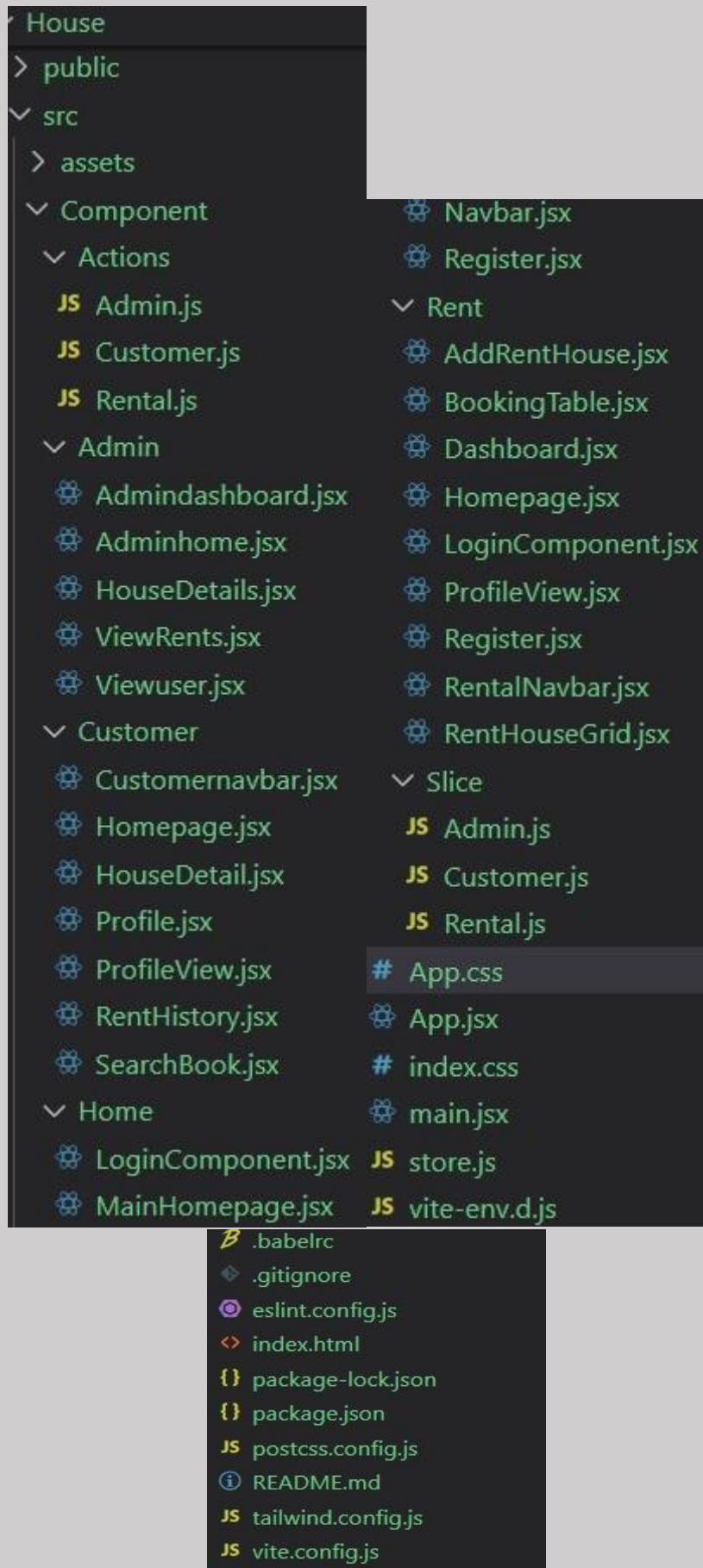
Folder Structure

Backend Folders

```

  Backend
  |
  |__ src
  |   |__ Controller
  |   |   |__ Admin.js
  |   |   |__ Customer.js
  |   |   |__ Rental.js
  |   |__ modal
  |   |   |__ Apartment.js
  |   |   |__ BookModal.js
  |   |   |__ Customer.js
  |   |   |__ Rentals.js
  |   |__ uploads
  |   |__ Validation
  |   |   |__ Customer.js
  |   |   |__ Rental.js
  |   |__ index.js
  |   |__ .babelrc
  |   |__ .env
  |   |__ jsconfig.json
  |   |__ package-lock.json
  |   |__ package.json

```



Frontend Folders

Running The Application

1. **cd (Change Directory)**

This command is used to navigate between directories. For example, to move into the backend folder, you would run

```
cd backend
```

2. **npm install (npm i)**

This command installs all the necessary dependencies listed in the package.json file for your project. You need to run this in both the backend and House (frontend) directories

```
npm install
```

3. **npm run build**

This command builds the React application for production, optimizing it for better performance. The build files are created in a /build directory inside the House folder

```
npm run build
```

4. **npm run dev**

This command is used to run the backend in development mode. It watches for changes and automatically restarts the server

```
npm run dev
```

5. **npm start**

This command starts both the React frontend (in House) and the backend. For React, it runs the development server and opens your app in a browser. For the backend, it starts the Node.js server

```
npm start
```

These commands will help streamline development and deployment for both the backend and frontend components of project

API Documentation

Admin

1. Admin Dashboard

- Method: GET
- Endpoint: /api/admin/dashboard
- Description: Fetches data for the admin dashboard.

2. Manage Users

- Method: GET
- Endpoint: /api/admin/viewuser
- Description: Retrieves the list of users for admin management.

3. Review Rental Applications

- Method: GET
- Endpoint: /api/admin/viewrent
- Description: Retrieves rental applications for admin review.

Customer

1. Homepage Data

- Method: GET
- Endpoint: /api/homepage
- Description: Fetches the homepage content for customers.

2. Search for Properties

- Method: GET
- Endpoint: /api/searchbook
- Query Parameters:
 - location: Location to search for properties (e.g., Cityville)
 - priceRange: Price range (e.g., 0-1000)
- Description: Search for available rental properties.

3. Manage Customer Bookings

- Method: GET
- Endpoint: /api/cart
- Description: Retrieves customer bookings in the cart.

]

4. View House Details

- Method: GET
- Endpoint: /api/housedetail/{id}
- Path Parameter:
 - id: ID of the house (e.g., 1)
- Description: Fetches details for a specific house.

Rental

1. Rental Dashboard

- Method: GET
- Endpoint: /api/rents/dashboard
- Description: Retrieves the dashboard data for rental users.

2. Add New House for Approval

- Method: POST
- Endpoint: /api/rents/addhouse

3. Rental User Registration

- Method: POST
- Endpoint: /api/rent/register
- Description: Registers a new rental user.

4. Rental User Login

- Method: POST
- Endpoint: /api/rent/login
- Request Body:
- Description: Rental user login.

Authentication

In the **House Hunt** project, authentication and authorization are key to ensuring that users, property owners, and administrators access only the features and data they are authorized to use. Here's how authentication and authorization are typically handled:

1. Authentication:

Authentication verifies the identity of a user (admin, customer, or rental user) before allowing them access to the platform. In the House Hunt project, authentication can be handled using JWT (JSON Web Tokens).

- **Token-based Authentication:**
 - Upon successful login, the backend generates a JWT token that contains the user's identity and role (e.g., admin, customer, or rental user).
 - This token is sent back to the client and stored in the client-side (usually in local storage or cookies).
 - For every subsequent request, the client sends the token in the Authorization header as a Bearer token.
 - The backend validates the token to ensure the user is authenticated.
- **Login Example:**
 - Method: POST
 - Endpoint: /api/rent/login
- **Token Validation:**
 - Every request to a protected route (e.g., admin dashboard, user profile) includes the token.
 - The backend middleware checks if the token is valid and if the user's role matches the required permissions.

2. Authorization:

Authorization controls access to specific resources based on the user's role (admin, customer, rental user). Once a user is authenticated, their role (stored in the JWT) determines what resources they can access.

- **Role-based Authorization:**
 - Admins can access routes like /admin/viewuser or /admin/viewrent, while customers and rental users cannot.

- Rental users can add and manage their listings but cannot access the admin dashboard or other users' bookings.
- **Example of Authorization:**
 - Admin Role: Only users with the "admin" role can access /api/admin/* routes.
 - Customer Role: Only customers can view /api/ make bookings.
 - Rental Role: Rental users can manage their own properties through /api/rents/*.

3. Token-based Sessions:

The token serves as a session identifier, and since it is stateless, it doesn't require server-side session storage. However, the backend can use middleware to:


- Validate the token on every request to ensure it's still valid.
- Check if the token has expired (JWT tokens usually have an expiration time).

Security Considerations:


- Token Expiration: JWT tokens are generally issued with an expiration time (e.g., 24 hours). Once expired, the user must log in again to receive a new token.
- Refresh Tokens: For long sessions, a refresh token can be used to issue a new access token without requiring the user to log in again.
- Secure Storage: Tokens should be securely stored (e.g., in HTTP-only cookies) to prevent XSS (Cross-Site Scripting) attacks.
- Role-based Access Control (RBAC): Ensures that users can only perform actions permitted by their role (e.g., customers can't delete other users' listings).
- **Middleware Flow:**
 1. The backend middleware checks for the presence of a token in the request header.
 2. It verifies the token using a secret key to ensure it's not tampered with.
 3. If valid, the user proceeds to access the requested route.
 4. If invalid or missing, the user receives a 401 Unauthorized error.

This structure ensures that only authenticated and authorized users can perform actions based on their roles in the **House Hunt** project.

User Interface

 **HOUSE HUNT**

Customer RegistrationLogin



Login

Email


Enter your email

Password


Enter your password

Login

[Forgot your password?](#)

 **HOUSE HUNT**

Customer RegistrationLogin



Register

Full Name

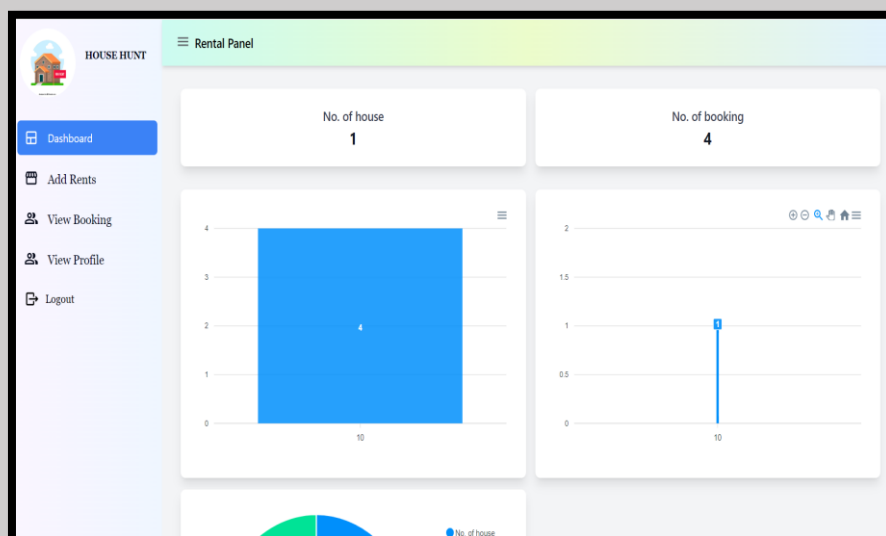
shyam@gmail.com


Phone

Address


Choose File No file chosen

Register



 **HOUSE HUNT**

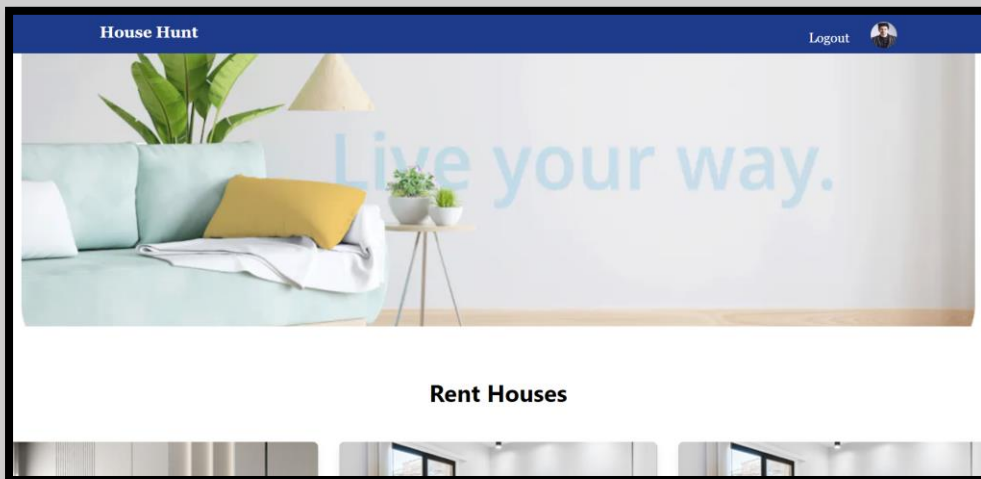
[Rental Registration](#) [Rental Login](#)




Register


| No file chosen

[Register](#)



House Hunt

Logout 

 **Ashok**

Contact Information
Email: ashok123@gmail.com
Phone: 9996543546
Address: AT Agraharam

Account Information
Password: 1234
[Change Password](#) [My Orders](#)

Testing

1. Unit Testing:

- a. Focuses on testing individual components or functions, such as user registration, property listing, or booking.
- b. Each function or method is tested in isolation to ensure it performs as expected.
- c. **Example:** Verifying that the user registration function successfully creates a new user and returns the expected response.

2. Integration Testing:

- a. Ensures that different components of the system work together seamlessly.
- b. This is especially important for testing interactions between the frontend and backend, like API calls.
- c. **Example:** Testing the interaction between the user registration form (frontend) and the user creation API (backend).

3. Black Box Testing:

- a. Focus: This type of testing focuses on the functionality of the system without knowledge of the internal code structure or logic. The tester only knows the inputs and expected outputs, not how the software processes them.

4. White Box Testing:

Focus: White box testing involves understanding the internal structure, logic, and code of the system. The tester examines how the inputs are processed within the code to produce the expected outputs.

Admin Test Cases

Test Case :1

Test Scenario: Admin Login with valid credentials

Precondition: Admin account exists

Test Steps:

Open Admin login page

Enter valid username and password

Click on "Login"

Expected Result: Admin should successfully log in to the dashboard.

Test Case :2

Test Scenario: Admin Login with invalid credentials

Precondition: None

Test Steps:

Open Admin login page

Enter invalid username/password

Click on "Login"

Expected Result: Error message should be displayed: "Invalid username or password."

Test Case: 3

Test Scenario: Admin adds new rental property

Precondition: Admin is logged in

Test Steps:

Navigate to "Add Rentals" page

Enter rental details (property name, location, price, etc.)

Click "Submit"

Expected Result: Rental property should be successfully added to the system and visible in the listings.

Test Case: 4

Test Scenario: Admin views users list

Precondition: Admin is logged in

Test Steps:

Navigate to "View Users" page

Expected Result: List of all registered users should be displayed.

Test Case :5

Test Scenario: Admin verifies a house listing

Precondition: Admin is logged in, House listing pending verification

Test Steps:

Navigate to "Verify Houses" page

Select an unverified house listing

Click "Verify"

Expected Result: The house listing should be marked as "Verified" and available to users.

Test Case: 6

Test Scenario: Admin views booking history

Precondition: Admin is logged in, Bookings exist for a property

Test Steps:

Navigate to "View Bookings" page

Expected Result: List of all current and past bookings should be displayed.

Customer Test Cases

Test Case: 1

Test Scenario: User Registration with valid details

Precondition: None

Test Steps:

Open registration page

Enter valid user details (name, email, password, etc.)

Click "Submit"

Expected Result: User account should be created successfully, and a confirmation email should be sent.

Test Case :2

Test Scenario: User Login with valid credentials

Precondition: User account exists

Test Steps:

Open login page

Enter valid username and password

Click "Login"

Expected Result: User should successfully log in to their account.

Test Case: 3

Test Scenario: User Login with invalid credentials

Precondition: None

Test Steps:

Open login page

Enter invalid username/password

Click "Login"

Expected Result: Error message should be displayed: "Invalid username or password."

Test Case :4

Test Scenario: User views verified houses

Precondition: User is logged in, verified house listings exist

Test Steps:

Navigate to "View Verified Houses" page

Expected Result: List of verified rental properties should be displayed.

Test Case: 6

Test Scenario: User adds a booking

Precondition: User is logged in, verified houses exist

Test Steps:

Browse house listings

Select a property

Click "Book"

Enter booking details and confirm

Expected Result: Booking should be added successfully, and a confirmation message should be displayed.

Test Case: 7

Test Scenario: User views booking history

Precondition: User is logged in, Past bookings exist

Test Steps:

Navigate to "History" page

Expected Result: List of all past bookings should be displayed.

Rental Test Cases

Test Case: 1

Test Scenario: Rental owner login with valid credentials

Precondition: Rental owner account exists

Test Steps:

Open rental owner login page

Enter valid username and password

Click "Login"

Expected Result: Rental owner should successfully log in to the dashboard.

Test Case: 2

Test Scenario: Rental owner adds a new house listing

Precondition: Rental owner is logged in

Test Steps:

Navigate to "Add Houses" page

Enter house details (property name, location, price, etc.)

Click "Submit"

Expected Result: House listing should be successfully added to the system.

Test Case: 3

Test Scenario: Rental owner views bookings

Precondition: Rental owner is logged in, Bookings exist for properties

Test Steps:

Navigate to "View Bookings" page

Expected Result: List of current and past bookings for the owner's properties should be displayed.

Known Issues

User Registration/Login Issues:

- **Session management flaws:** Sessions may not expire properly, leading to security vulnerabilities.
- **Password strength enforcement:** Weak passwords might be accepted if no validation is implemented.

2. Property Listings Management:

- **Duplicate listings:** The system may not prevent users or admins from adding duplicate properties.
- **Inconsistent property visibility:** Newly added or verified properties may not appear immediately in listings due to caching or database update issues.

3. Booking Issues:

- **Overbooking:** Lack of synchronization might allow multiple users to book the same property simultaneously.
- **Transaction issues:** Problems in handling payment gateway integration or confirming the transaction can disrupt the booking process.

4. Admin Dashboard Challenges:

- **Slow performance:** Displaying large amounts of data (e.g., users, bookings, listings) might cause lag if pagination or efficient database queries are not implemented.
- **Verification process delays:** Unoptimized workflows for house verification could cause delays in listing verification.

Future Enhancements

1. Enhanced User Experience:

- Search filters: Implement advanced search filters (e.g., price range, location, property type) to help users find properties more easily.
- Property comparison: Allow users to compare multiple properties side by side.

2. Improved Admin Features:

- Automated verification: Introduce automated property verification to reduce manual effort for admins.
- Real-time notifications: Admins should get real-time notifications for new property submissions, booking updates, or user queries.

3. Property Owner Enhancements:

- Property analytics: Provide rental owners with analytics (e.g., views, inquiries, bookings) for their properties.
- Messaging system: Enable direct communication between property owners and potential renters through an in-app messaging system.

4. Booking System Enhancements:

- Dynamic pricing: Allow property owners to set dynamic pricing based on demand, season, or occupancy.
- Refund and cancellation policy: Implement flexible booking and cancellation policies, along with automated refunds.

5. Security and Privacy Enhancements:

- Two-factor authentication (2FA): Add an extra layer of security for users and admins.
- End-to-end encryption: Enhance security by encrypting all sensitive data during transactions and communication.

6. Mobile App Development:

- Mobile application: Develop a mobile app for a more convenient user experience on the go.
- Push notifications: Send users instant notifications about new properties, booking updates, and account activity.