## **Full Stack Development with MERN**

**Project Documentation format**

### 1. Introduction

**Project Title**: “ House Rent App Using MERN ”

**Team Members**:

1. J. Rithika. (Team Lead) - 310521104093

2. R.Suganthi. (Frontend) - 310521104116

3. D.Sumithra. (Backend) - 310521104117

4. Priyakumari. (Database management) - 31052110406

### 2. Project Overview

**Purpose**:

The purpose of the "House Rent Using MERN" project is to create a comprehensive web application for managing and listing rental properties. Using the MERN stack (MongoDB, Express.js, React, and Node.js), this application will streamline the rental process by providing a user-friendly interface for landlords to list properties and for tenants to browse and book available rentals.

**Goals:**

**1. Efficient Property Listing** – Develop an intuitive platform for landlords to add, update, and manage rental property listings, including details like location, price, and amenities.

**2. User-Friendly Search and Filter** – Provide renters with robust search and filtering options to easily find properties that meet their preferences.

**3. Secure Booking and Payment Integration** – Enable secure booking requests and payment options, ensuring a smooth and safe transaction process.

**4. Responsive Design** – Ensure the application is mobile-friendly, offering an optimal experience across devices.

**5. Data Analytics and Insights** – Integrate analytics for landlords to track views, bookings, and other metrics to help optimize listings.

**3. Architecture**

### Frontend: React Architecture

The frontend of the House Rent app is built using React and follows a component-based architecture, emphasizing modularity, scalability, and reusability.

**Component Structure**:

* + **Main Pages**:
    - **Home Page** – Displays featured properties and search options.
    - **Property Listings Page** – Shows a list of properties based on search and filter criteria.
    - **Property Details Page** – Displays comprehensive details about a selected property, including images, descriptions, amenities, and landlord contact information.
    - **User Profile Page** – Allows users to view and manage their listings, bookings, and personal details.
    - **Admin Dashboard** (optional) – Provides property management and analytics for landlords.
  + **UI Components**:
    - **Header and Navigation** – Includes the main navigation bar, with links to home, search, user profile, and more.
    - **Search and Filter Bar** – Enables users to filter properties by location, price, property type, and amenities.
    - **Property Card** – A reusable component displaying summary details of each property in the listing view.
    - **Booking Form** – Allows users to book properties and manage booking details.
  + **State Management**:
    - **Context API or Redux** – Manages global state for user authentication, property data, and booking details, ensuring consistency across components.
    - **API Calls** – Uses axis or the fetch API to handle data requests between the frontend and backend, primarily through a central service layer.

1. **Routing**:
   * Uses **React Router** for navigation across different pages without reloading, ensuring a smooth user experience.
2. **Authentication**:
   * Utilizes token-based authentication (JWT) to manage user sessions, storing tokens in local storage or cookies for secure access.
3. **Responsive Design**:
   * Employs CSS frameworks like Bootstrap or Material-UI to ensure mobile responsiveness and maintain consistency across devices.

### Backend: Node.js and Express.js Architecture

The backend is developed using Node.js and Express.js, providing a RESTful API to handle requests from the frontend, process business logic, and communicate with the MongoDB database.

1. **REST API Endpoints**:
   * **Authentication**:
     + POST /api/auth/register – Registers a new user (Renter or Owner).
     + POST /api/auth/login – Authenticates a user and generates a JWT token.
   * **Properties**:
     + GET /api/properties – Retrieves a list of properties, supporting search and filter options.GET /api/properties/:id – Retrieves details of a specific property.POST /api/properties – Allows landlords to add a new property (requires authentication).

**Bookings**:

* + - POST /api/bookings – Handles booking requests from tenants.
    - GET /api/bookings/user/:id – Retrieves bookings made by a specific user.
    - PUT /api/bookings/:id – Updates booking status, such as confirming or canceling.

1. **Middleware**:
   * **Authentication Middleware** – Validates JWT tokens to protect endpoints requiring user login.
   * **Error Handling Middleware** – Handles errors gracefully, ensuring clear error messages and proper HTTP status codes.
2. **Business Logic**:
   * **Booking Management** – Includes logic to handle availability checks, booking validation, and payment status.
   * **User Roles and Permissions** – Differentiates access levels between tenants and landlords

### Database: MongoDB Schema and Interactions

MongoDB is the primary database, storing information about users, properties, and bookings.

1. **Schemas**:
   * **User Schema**:
     + Fields: userId, name, email, password, role (tenant/landlord), createdAt
     + Relationships: Properties listed by landlords, bookings made by tenants.
   * **Property Schema**:
     + Fields: propertyId, ownerId, title, description, location, price, images, amenities, createdAt, availability
     + Relationships: Linked to the landlord (User) who owns the property.
   * **Booking Schema**:
     + Fields: bookingId, propertyId, userId, startDate, endDate, status (pending/confirmed/canceled), paymentStatus
     + Relationships: Linked to the Renter(User) making the booking and the property being booked.

**Database Interactions**:

* + **CRUD Operations**:
    - **Properties**: Handled through property CRUD operations, allowing for seamless management of listings.
    - **Users**: Managed through registration and profile updates.
    - **Bookings**: Supports creation, updating, and querying for bookings based on user and property.
    - Indexes on key fields like location, price, and availability for faster search and filtering of properties.

This architecture leverages the strengths of the MERN stack to create a seamless, scalable, and user-friendly rental platform. The separation between the frontend, backend, and database allows for easy maintenance and future expansion of features.

## **4. Setup Instructions**

### Prerequisites:

Before setting up the project, make sure the following software dependencies are installed:

1. **Node.js** (version 14 or higher) – [Download and install from Node.js website](https://nodejs.org/)
2. **MongoDB** – [Download and install MongoDB](https://www.mongodb.com/try/download/community) or use a hosted MongoDB service like MongoDB Atlas.
3. **Git** – [Download and install Git](https://git-scm.com/downloads)

*Optional*:

* **npm** (Node Package Manager, usually comes with Node.js) or **yarn** (alternative package manager) – to install dependencies.

### Installation:

**Clone the Repository**: Open a terminal or command prompt, and clone the repository using:

bash

Copy code

git clone <repository-url>

Replace <repository-url> with the actual URL of your project’s repository.

**Install Dependencies for Backend and Frontend**:

Backend:

Navigate to the backend directory:

Copy code

cd backend

Install the required packages:

Copy code

npm install

Frontend:

Open a new terminal tab or window, navigate to the frontend directory:

Copy code

cd frontend

Install the required packages:

Copy code

npm install

**Set Up Environment Variables**:

In the **Backend** directory, create a .env file to store environment variables:

Copy code

touch .env

Open the .env file and add the following environment variables:

plaintext

Copy code

PORT=5000

MONGO\_URI=<Your\_MongoDB\_URI>

JWT\_SECRET=<Your\_JWT\_Secret>

Replace <Your\_MongoDB\_URI> with the MongoDB connection string, and <Your\_JWT\_Secret> with a strong secret for JWT authentication.

In the **Frontend** directory, create a .env file to store environment variables for the React app:

Copy code

touch .env

Open the .env file and add the following environment variable:

REACT\_APP\_API\_URL=http://localhost:5000/api

**Start the Development Servers**:

Backend:

In the backend directory, start the Node.js server:

Copy code

npm start

This will start the backend server on http://localhost:5000.

Frontend:

In the frontend directory, start the React development server:

Copy code

npm start

This will start the frontend on

**5. Folder Structure**

### Overall Project Structure

The project is typically organized into two main folders: client for the frontend (React) and server for the backend (Node.js and Express).

### Client: React Frontend Structure

The React frontend is organized to keep components, pages, utilities, and assets modular and easy to manage

Server: Node.js and Express.js Backend Structure

The backend structure in Node.js uses Express to handle routes, controllers, models, and middleware, with MongoDB as the database.

#### **Key Folders and Files in client/src/**:

* **components/** – Holds reusable UI components like the Navbar, Footer, and PropertyCard to keep the code modular.
* **pages/** – Contains the main pages (e.g.,
* Home, Listings, PropertyDetails, Profile, and AdminDashboard) used for routing and navigation.
* **services/** – Centralized API calls, often using Axios for making HTTP requests to the backend.
* **context/** – Manages global state for the app, such as user authentication and property data
* **App.js** – Sets up React Router for page navigation.
* **index.js** – Main entry point that renders the App component.

**6. Running the Application**

* Provide commands to start the frontend and backend servers locally.

**Frontend:** o run the House Rent application locally, you need to start both the frontend (React) and backend (Node.js) servers.

### Commands to Start the Servers Locally

1. **Start the Backend Server**:
   * Open a terminal, navigate to the server directory
   * **Backend:** npm start in the server directory.

**7. API Documentation**

* Document all endpoints exposed by the backend.
* Include request methods, parameters, and example responses.

**8. Authentication**

* Explain how authentication and authorization are handled in the project.
* Include details about tokens, sessions, or any other methods used.

**9. User Interface**

**1. Home Page**

* **Features**: Showcases featured properties, search bar for locations, property types, and price range.
* **UI Elements**: A hero image with a search bar overlay, property cards displaying image, price, location, and a brief description.
* **Example Layout**:
  + Navigation bar at the top (logo, links to listings, login, and signup).
  + Featured properties section with cards.
* **Screenshots**:
  + Full-width hero section.
  + Property cards with high-quality images and essential info.

**2. Property Listings Page**

* **Features**: Lists all available properties with options to filter and sort (by price, location, size, etc.).
* **UI Elements**:
  + Filters panel (checkboxes or dropdowns for price range, location, and property type).
  + Grid layout of property cards with thumbnail images, pricing, location, and a "View Details" button.
* **Screenshots**:
  + Filter panel on the left.
  + Grid of property cards, highlighting property info and quick access buttons.

**3. Property Details Page**

* **Features**: Displays detailed information about a selected property.
* **UI Elements**:
  + Image carousel for property photos.
  + Property details section with description, amenities, location map, and landlord information.
  + Contact button or form for inquiries.
* **Screenshots**:
  + Image carousel for property pictures.
  + "Contact Landlord" form or button.

**4. Login and Signup Pages**

* **Features**: Allow users to create accounts and log in.
* **UI Elements**:
  + Input fields for email, password, and name (for signup).
  + Login button, "Forgot Password" link, and social login options (optional).
* **Screenshots**:
  + Login form with a clean and simple design.
  + Signup form showing input validation.

**5. User Dashboard**

* **Features**: Provides account settings, saved properties, and rental application status (if applicable).
* **UI Elements**:
  + Tabs or sidebar for "Profile," "Saved Properties," and "Applications."
  + List of saved properties with links to their details.
* **Screenshots**:
  + Profile tab with editable personal information.
  + Saved properties section with quick access links.

**6. Admin Dashboard (Optional)**

* **Features**: Allows property managers to list, edit, and remove properties.
* **UI Elements**:
  + Dashboard with "Add New Property," "Manage Listings," and "View Applications."
  + Form for adding new property details (title, description, photos, location, etc.).
* **Screenshots**:
  + Listings management page with action buttons (edit, delete).

**Additional Tips**

* Use **React** for the UI components and manage state efficiently.
* Ensure **responsive design** so the app works on mobile and desktop.
* Style using **CSS frameworks** like Bootstrap or Tailwind for faster prototyping.

If you'd like specific screenshots or mockups generated, I can create visual representations based on these layouts.

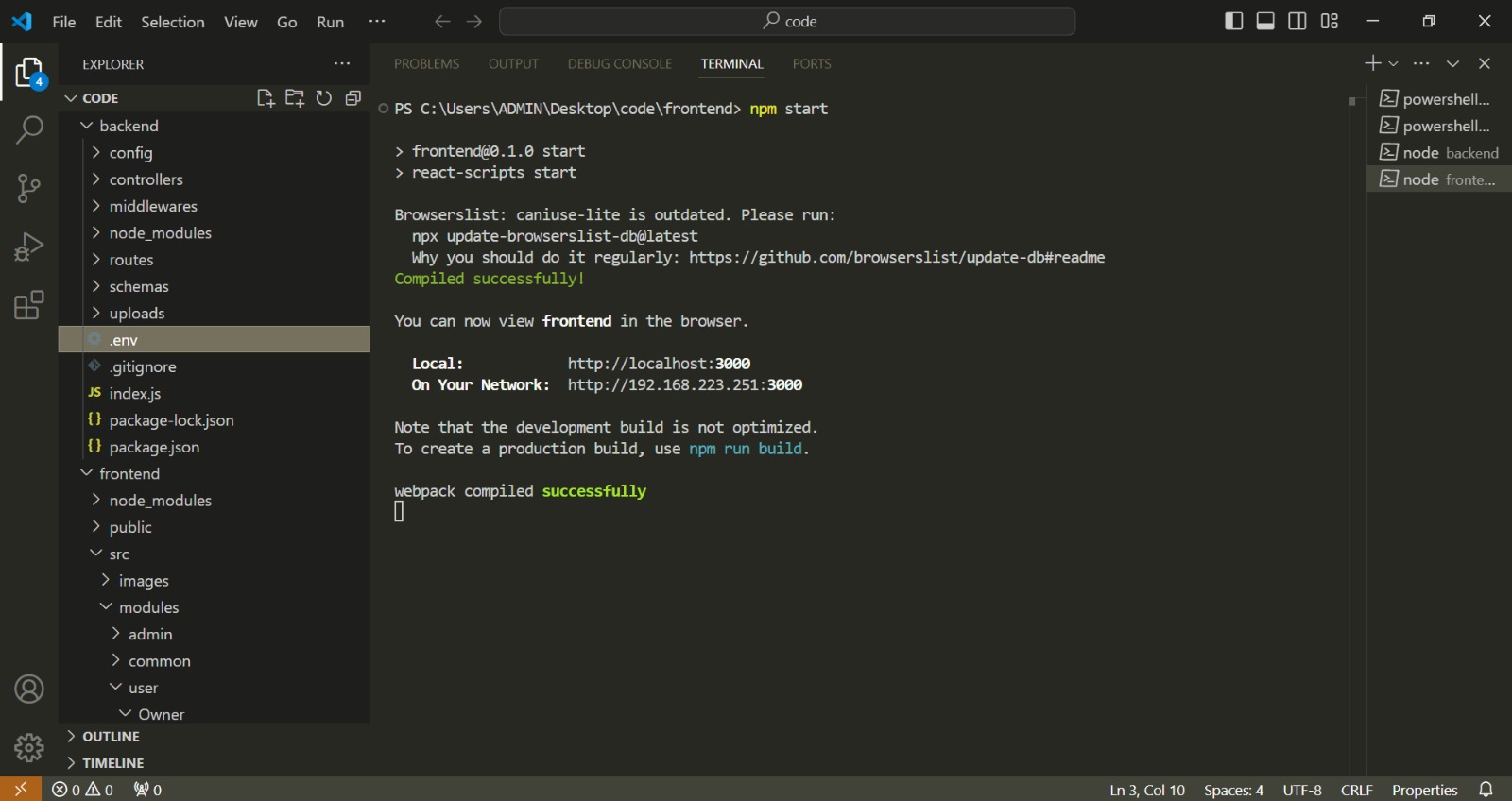
**10. Testing**

Testing a house rental app built with the MERN stack involves covering various testing strategies to ensure that each component functions correctly, the API endpoints are reliable, the UI is responsive, and data is correctly stored and retrieved from the database. Here’s a comprehensive testing strategy and tools you can use to cover all critical aspects of the application.

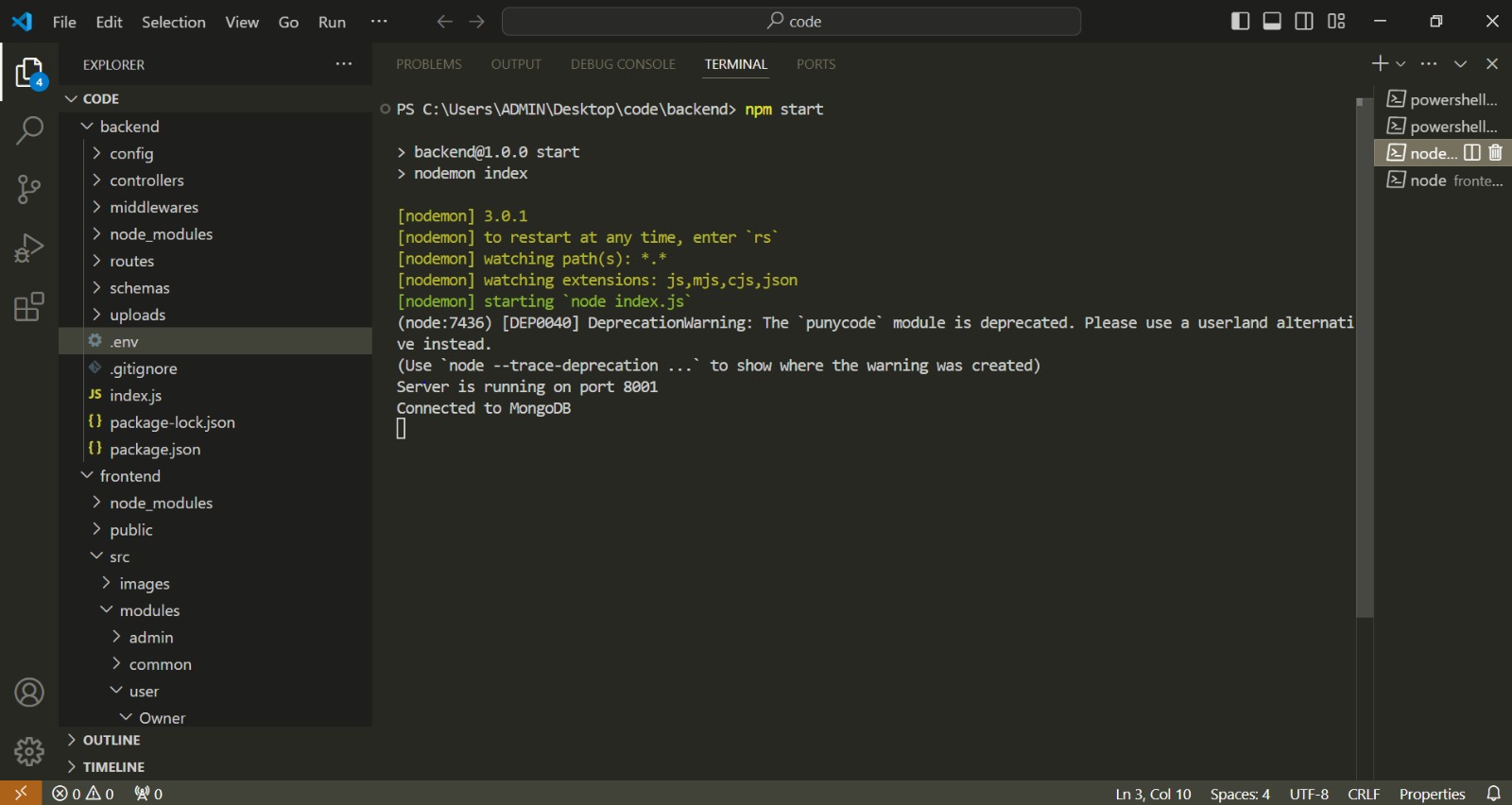
**Unit Testing :**

fronts (e.g., property cards, search bar, forms)..

**Frontend**: Test React comp



**Backend**: Test helper functions, controllers, and database operations.



**3. End-to-End (E2E) Testing**

* **Objective**: Simulate real user actions and verify the entire flow from the frontend to backend and database.
* **Focus Areas**:
  + User interactions such as browsing listings, viewing property details, logging in, signing up, and making inquiries.
  + Responsive design testing across different devices.
* **Tools**:
  + **Cypress**: A popular E2E testing tool for automating user interactions in the browser, capturing screenshots, and debugging UI issues.
  + **Selenium**: Useful for cross-browser testing and simulating user actions across various environments.

**4. Performance Testing**

* **Objective**: Measure the app's responsiveness and speed, especially for database queries and API responses.
* **Focus Areas**:
  + API performance under heavy load (important for listing retrieval and search).
  + Database read/write operations, especially if the app has a significant number of listings or users.
* **Tools**:
  + **Artillery**: For load testing and simulating concurrent users to test the backend.
  + **Lighthouse**: Integrated into Chrome DevTools, great for assessing frontend performance, such as page load times and mobile responsiveness.
  + **JMeter**: Another powerful tool for load testing, especially for APIs and backend performance.

**5. Security Testing**

* **Objective**: Identify vulnerabilities in authentication, data handling, and user interactions.
* **Focus Areas**:
  + Testing authentication (login and signup) to prevent unauthorized access.
  + SQL and NoSQL injection attempts, especially in the backend.
  + Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) testing on the frontend.
* **Tools**:
  + **OWASP ZAP**: Open-source security scanner that can identify common vulnerabilities.
  + **Postman**: Useful for manually testing API security by simulating malicious inputs.
  + **Helmet.js**: While not strictly a testing tool, it helps secure Express apps by setting appropriate HTTP headers.

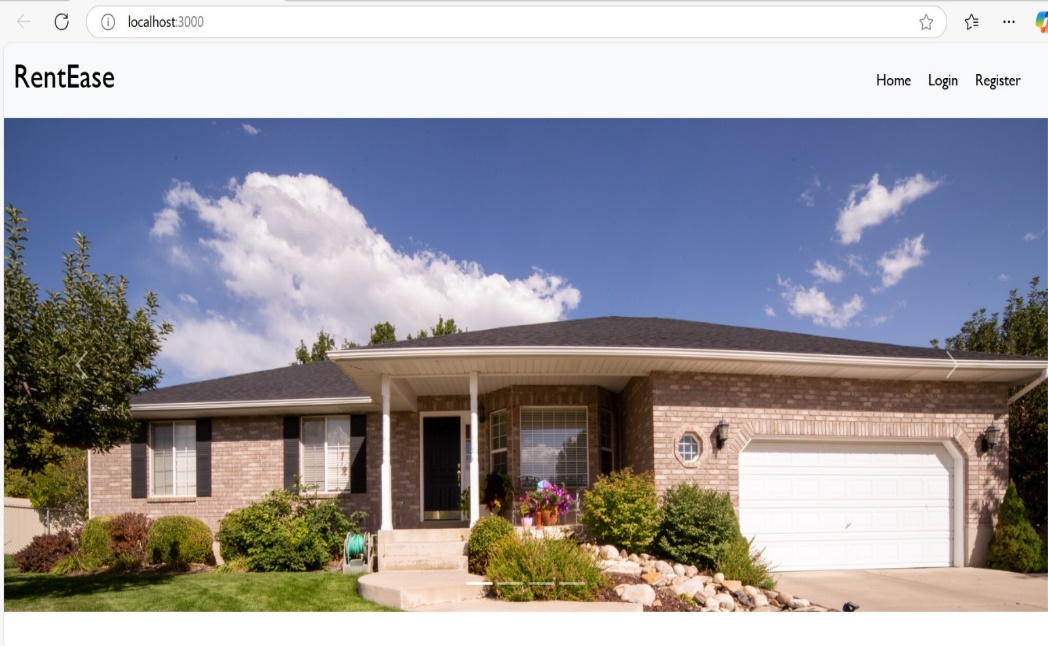
**6. UI and Usability Testing**

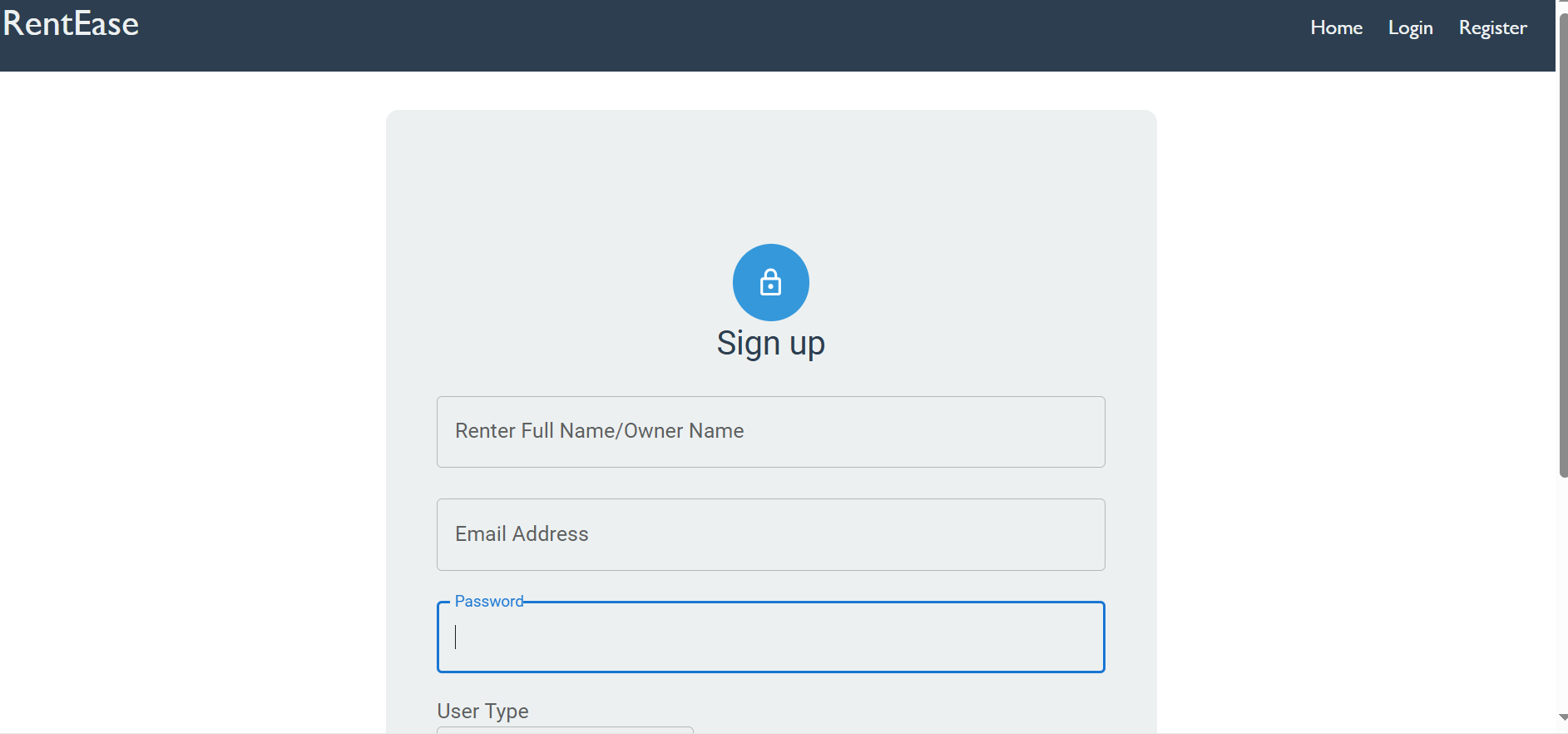
* **Objective**: Ensure the user interface is user-friendly, visually appealing, and bug-free.
* **Focus Areas**:
  + Component rendering and consistency in design (layout, colors, fonts).
  + Responsiveness on various screen sizes and browsers.
  + Accessibility to ensure that the app is usable for people with disabilities.
* **Tools**:
  + **Storybook**: Useful for isolated component testing, allowing developers to visually inspect each UI component.
  + **Axe by Deque**: Accessibility testing tool for identifying issues in the UI.
  + **BrowserStack or LambdaTest**: Cross-browser testing to validate the UI on multiple browsers and devices.

**7. Database Testing**

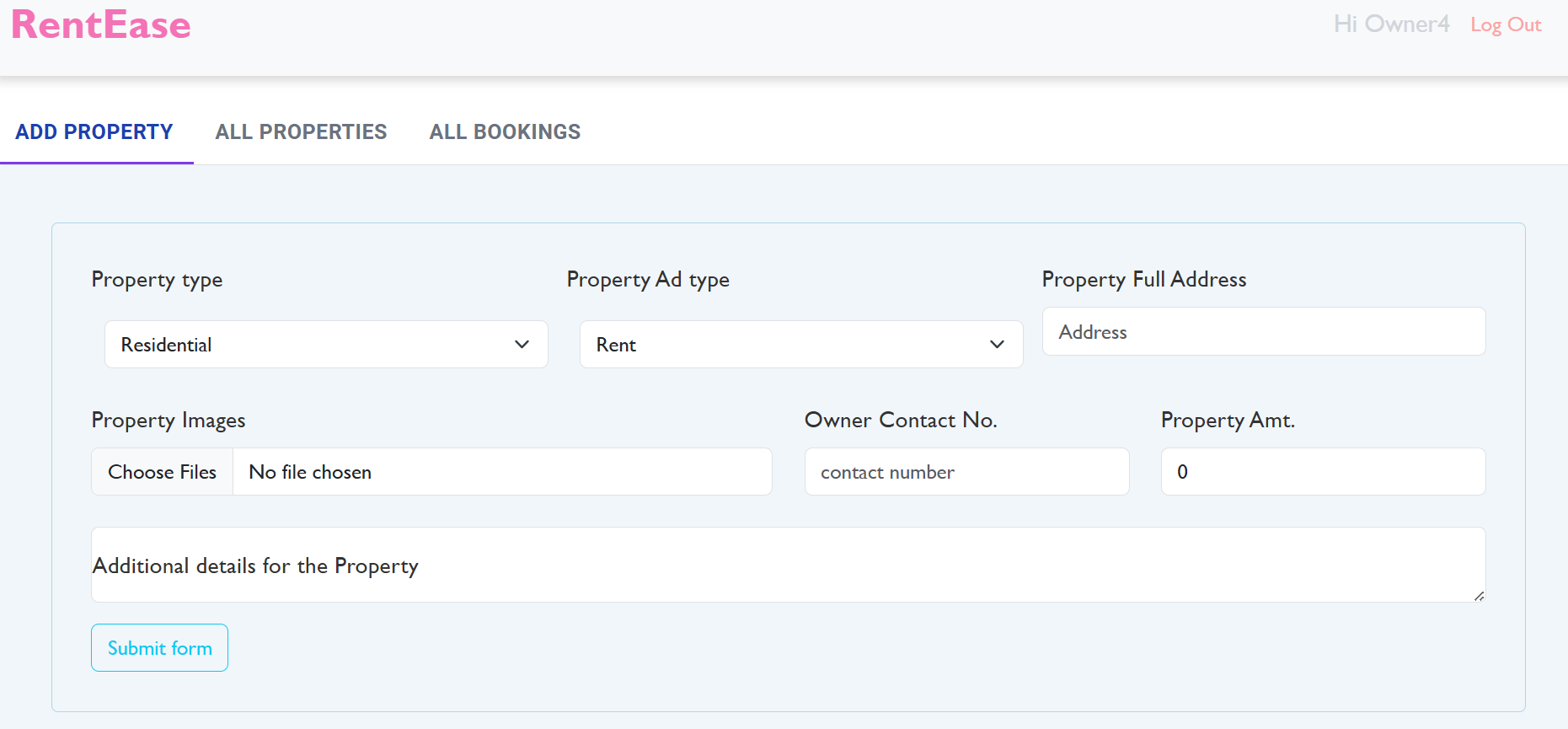
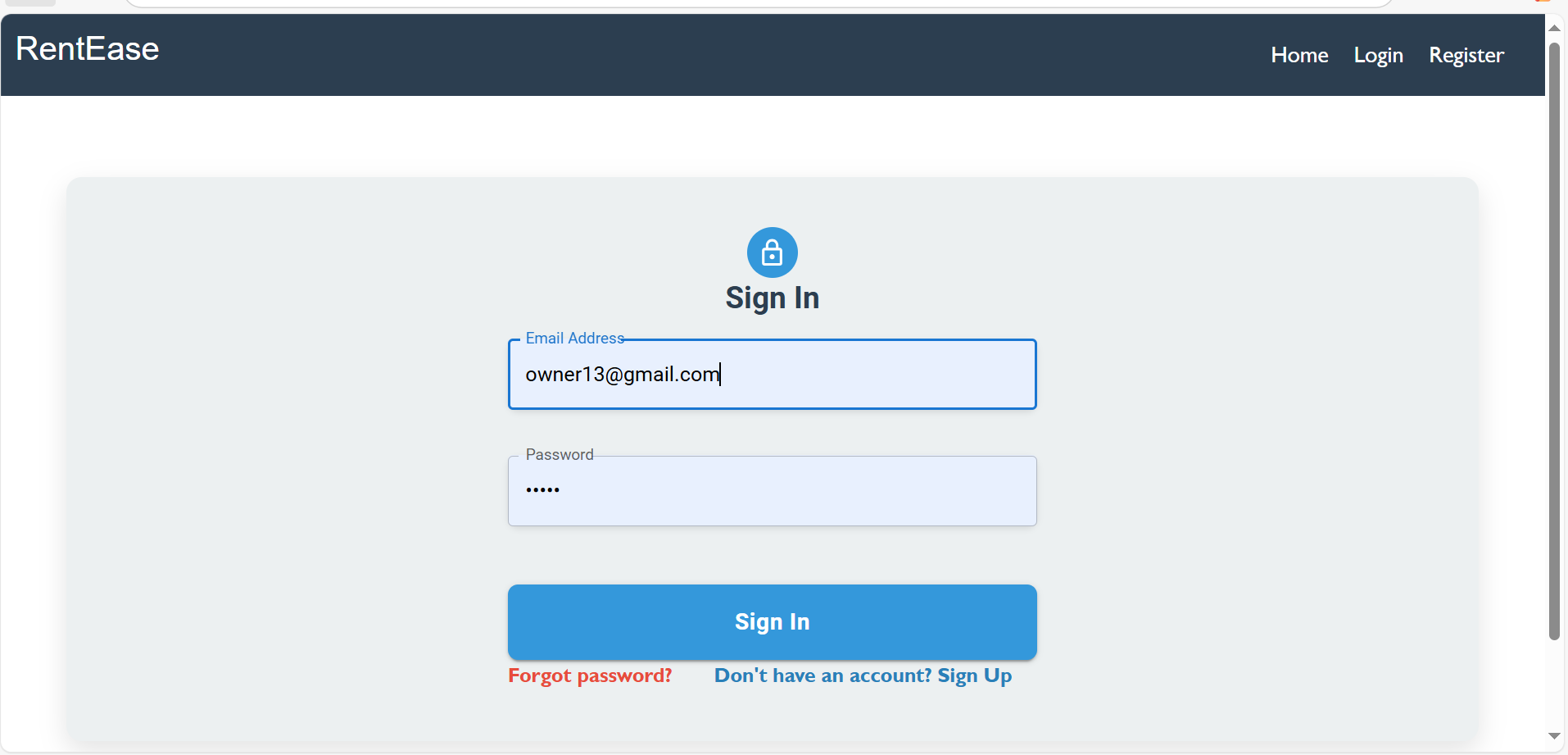
* **Objective**: Ensure data integrity and validate that database operations are consistent and reliable.
* **Focus Areas**:
  + Testing data schemas, constraints, and CRUD operations.
  + Ensuring that queries return expected results, even under load.
* **Tools**:
  + **MongoDB Atlas**: Useful for monitoring and managing databases.
  + **Mockgoose**: Allows mocking MongoDB interactions in unit tests without affecting the production database.

**Screenshots&Demo:**

* **landing page:** This is the landing page.
* **Register Page:** This is the register page.

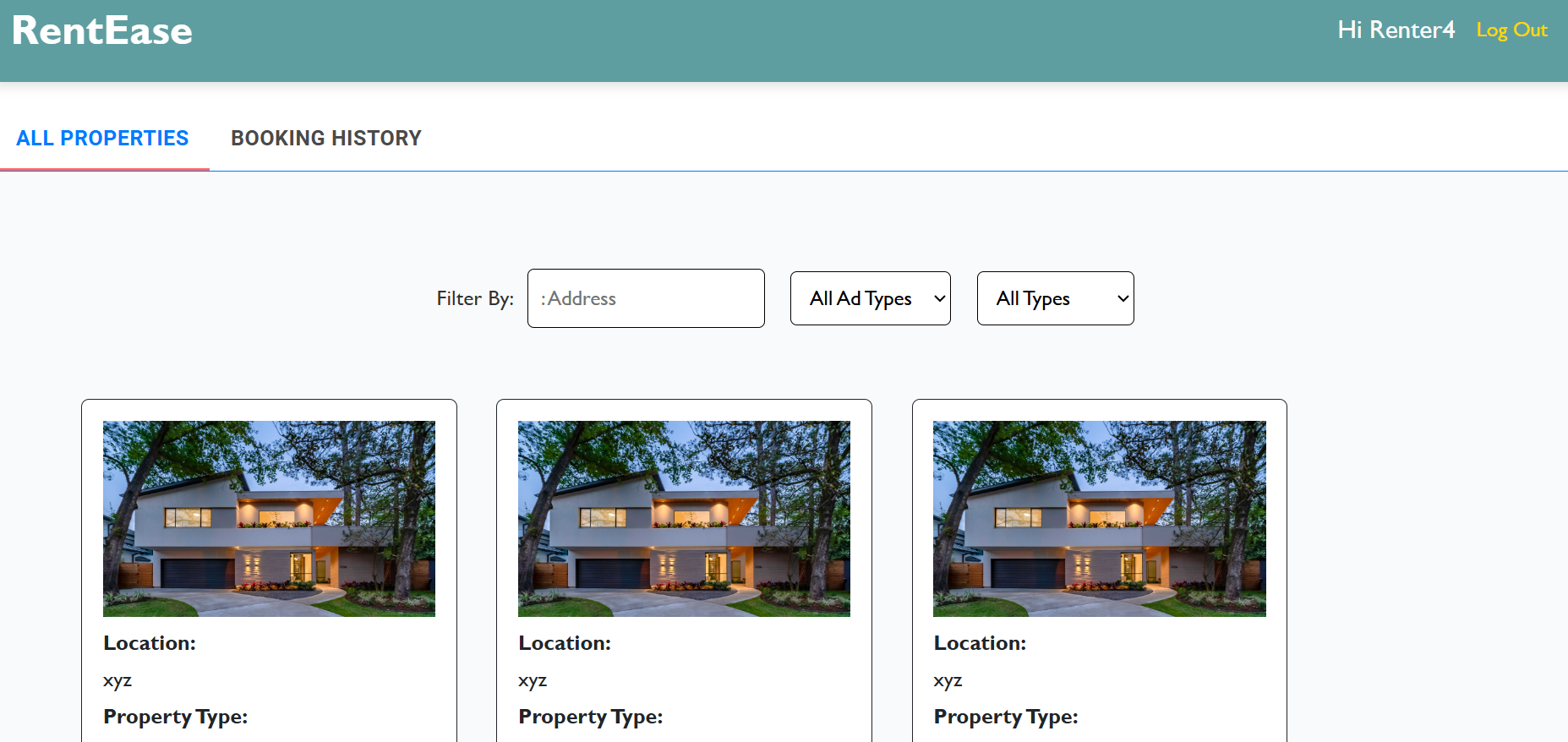


* **Login Page:** This is the login page.

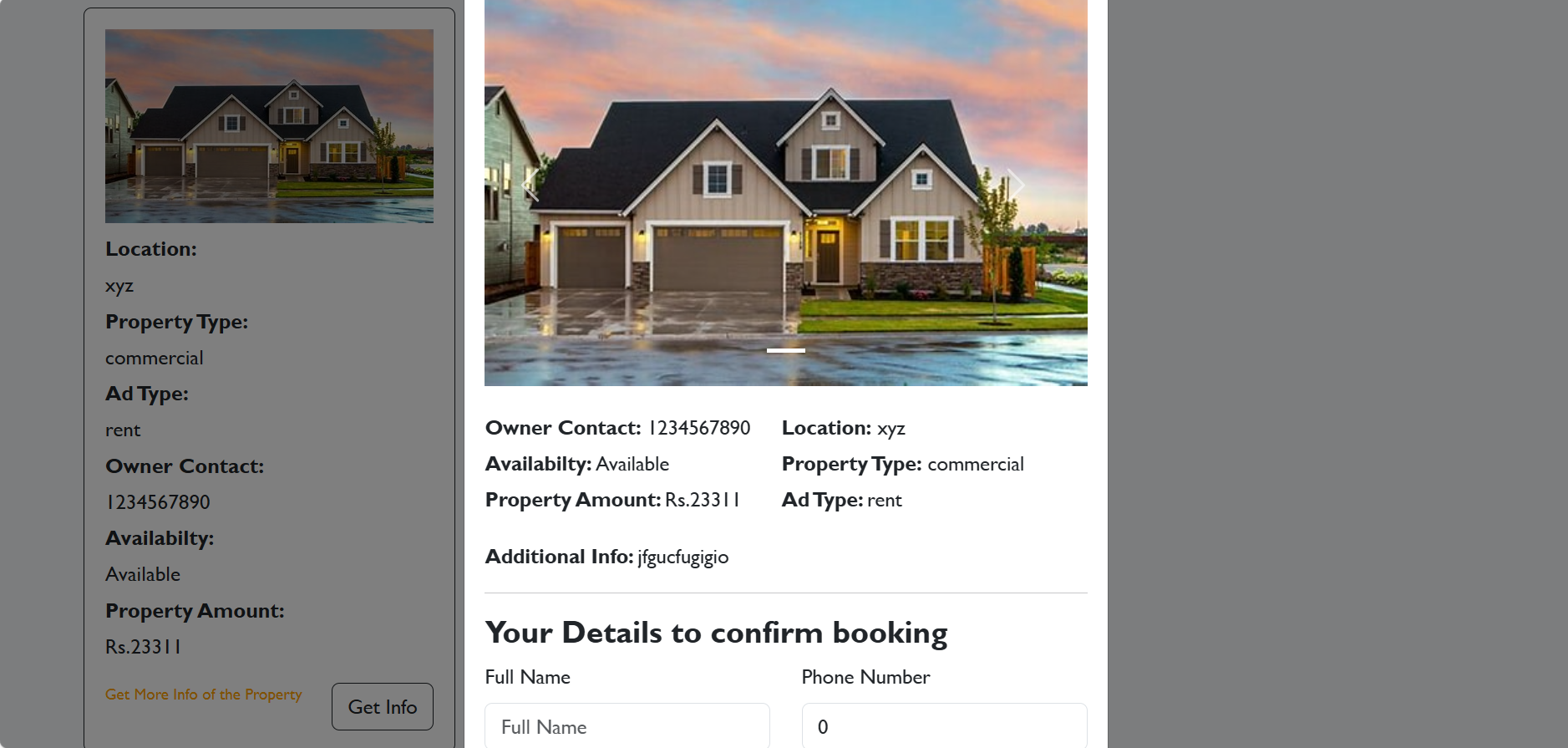


* **Owner Dashboard Page:** This is the owner dashboard page where we can add new properties.

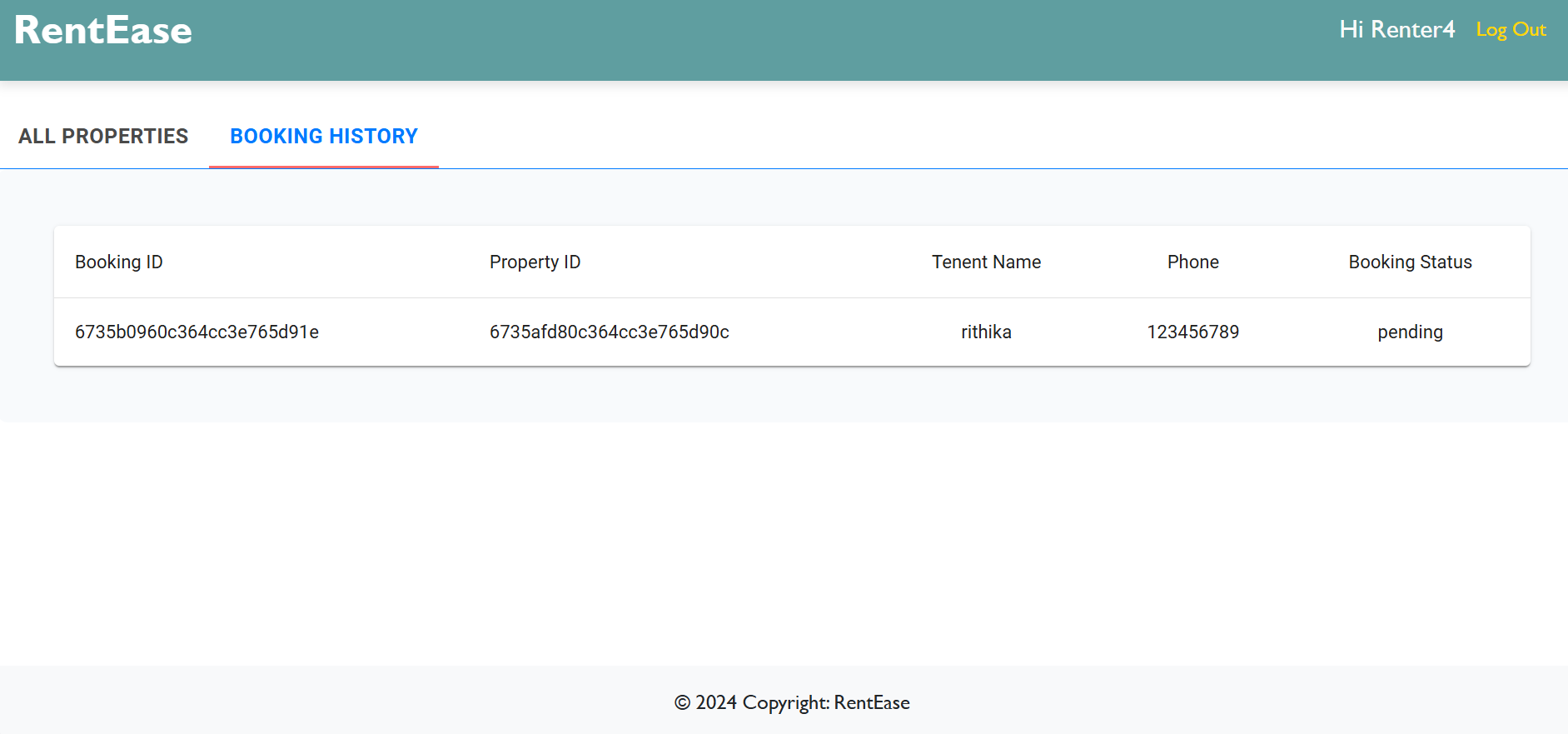
**All Properties :** This is the all properties page where we can view all properties posted by owner.



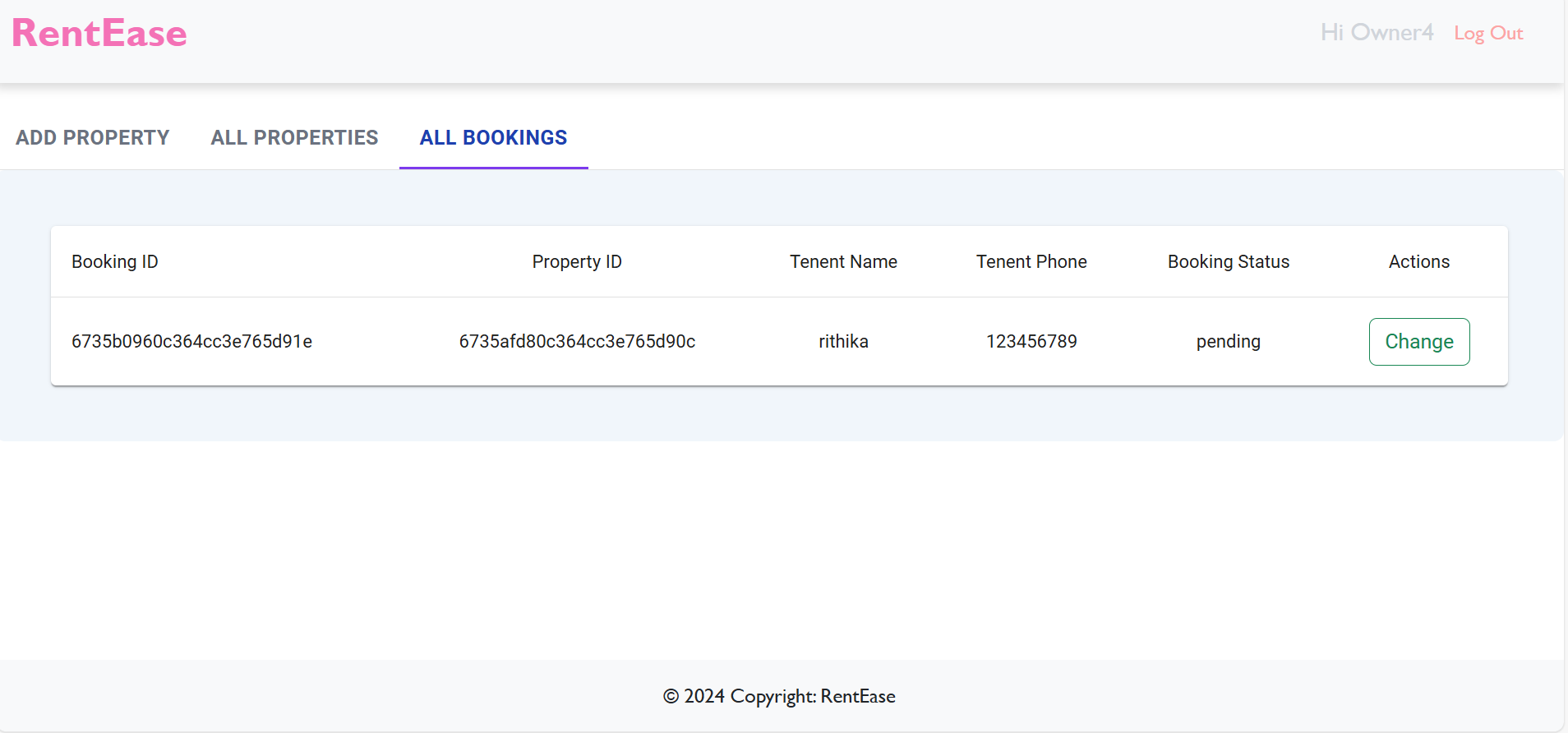
**House booking :** This is the house booking page .



**Booking History:** This is the booking history page where we can view the history of the properties booked by the renters and they can also view the booking status.



**Status Updating Page :** In this page the owner can able to change the status of the property booked by the renter.



**Updating Page : In this page the owner can able to update the property detail**

