# MERN STACK POWERED BY MONGODB

## HOUSE RENTAL APPLICATION

A PROJECT REPORT

*Submitted by*

| | |
|---|---|
| KARAMSETTY LIKHITHA | 113321104038 |
| PASUPULETI MRUDHULA | 113321104071 |
| POKALA DEEPIKA | 113321104075 |
| SHAIK RESHMA | 113321104089 |

BACHELOR OF TECHNOLOGY

*COMPUTER SCIENCE AND ENGINEERING*

VELAMMAL INSTITUTE OF TECHNOLOGY

CHENNAI 601 204

ANNA UNIVERSITY: CHENNAI 600 025

**ANNA UNIVERSITY**

**BONAFIDE CERTIFICATE**

Certified that this project report **"HOUSE RENTAL APPLICATION"** is the Bonafide work of **KARAMSETTY LIKHITHA -113321104038, PASUPULETI MRUDHULA - 113321104071, POKALA DEEPIKA -1133210475, SHAIK RESHMA – 113321104089"** who carried out the project work under my supervision.

| | |
|---|---|
| SIGNATURE | SIGNATURE |
| **Dr.V.P.Gladis Pushparathi** | **Mrs.Pratheeba R S** |
| **PROFESSOR,** | **ASSISSTANT PROFESSOR ,** |
| **HEAD OF THE DEPARTMENT,** | **NM COORDINATOR ,** |
| Computer Science and Engineering,<br>Velammal Institute of Technology,<br>Velammal Gardens, Panchetti,<br>Chennai-601 204. | Computer Science and Engineering,<br>Velammal Institute of Technology,<br>Velammal Gardens, Panchetti,<br>Chennai-601 204 |

## ACKNOWLEDGEMENT

We are personally indebted to many who had helped us during the course of this project work. Our deepest gratitude to the **God Almighty.**

We are greatly and profoundly thankful to our beloved Chairman **Thiru.M.V.Muthuramalingam** for facilitating us with this opportunity. Our sincere thanks to our respected Director **Thiru.M.V.M Sasi Kumar** for his consent to take up this project work and make it a great success.

We are also thankful to our Advisors **Shri.K.Razak**, **Shri.M.Vaasu**, our Principal **Dr.N.Balaji** and our Vice Principal **Dr.S.Soundararajan** for their never ending encouragement that drives ustowards innovation.

We are extremely thankful to our Head of the Department **Dr.V.P.Gladis Pushparathi** and Naan Mudhalvan Coordinator **Mrs.Pratheeba R S**, for their valuable teachings and suggestions.

The Acknowledgment would be incomplete if we would not mention word of thanks to our Parents.Teaching and Non-Teaching Staffs, Administrative Staffs and Friends for their motivation and support throughout the project.

Finally,we thank all those who directly or indirectly contributed to the successful completion of this project .Your contributions have been a vital part of our success.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

The **House Rental Application** is an innovative web-based platform designed to streamline the process of renting residential properties. Built using the MERN Stack, which includes MongoDB, Express.js, React, and Node.js, this project provides a dynamic, scalable solution for both tenants and landlords. The application aims to simplify property browsing, enhance user interaction, and offer a comprehensive management system for landlords, all while ensuring a smooth, responsive experience for users on both desktop and mobile devices.

At its core, the House Rental Application enables tenants to browse through various rental properties, each featuring detailed information such as location, price, amenities, and high-quality images. The system includes advanced search and filtering capabilities, allowing tenants to narrow down their choices based on specific criteria, such as price range, property type, and number of bedrooms. This feature ensures that users can easily find properties that meet their preferences and requirements. Additionally, the application supports a seamless booking system, where tenants can express interest in properties, schedule viewings, and communicate directly with landlords.

For landlords, the application offers a property management module that allows them to list, edit, and manage their rental properties. They can upload property details, update availability, and track inquiries from prospective tenants. To facilitate effective communication between tenants and landlords, the application includes a real-time messaging system, powered by Socket.io. This feature enables instant communication, helping users to stay updated on property availability, viewings, and other important matters.

With a strong emphasis on security and privacy, the House Rental Application integrates advanced measures to safeguard user data by protecting personal information. By fostering trust and reliability, the app aspires to create a safe and dependable platform for Rental System.

This project demonstrates a blend of technical proficiency and user-centered design, aiming to deliver a superior rental experience while catering to the needs of a dynamic and diverse user base. Through this House Rental Application, we aim to contribute to the growing digital Renting System, offering convenience, security, and satisfaction to all users involved.

Additionally, The **House Rental Application** is designed to offer a seamless, user-friendly experience for all parties involved in the rental process. By combining the flexibility of MongoDB with the power of the MERN stack, the application ensures a high level of performance, scalability, and ease of use. The platform not only simplifies the property rental process but also improves transparency, communication, and efficiency for both tenants and landlords. Whether you're looking for your next home or managing rental properties, this application provides a modern solution that caters to the needs of both users and property owners.

In conclusion, the House Rental Application exemplifies how the MERN stack can be leveraged to create an intuitive, feature-rich, and scalable web platform. By incorporating real-time communication, efficient property management tools, and secure authentication, this project demonstrates the power of modern web technologies in solving practical problems in the real estate sector. As the platform evolves, it will continue to provide a seamless experience for users, with the ability to handle increasing data and user traffic as the platform scales.

# CHAPTER 2

# PROJECT OVERVIEW

The House Renting App is a dynamic MERN stack application built with MongoDB, Express.js, React.js, and Node.js, designed to streamline the property rental process. It offers a user-friendly interface catering to both property owners and renters. Owners can register, log in, list properties, and manage bookings, while renters can browse available properties, check details, and make reservations. The app ensures efficient data storage and retrieval with MongoDB, while React provides a seamless and responsive user experience. With a robust backend powered by Express.js and Node.js, the application delivers secure, scalable, and real-time functionality for a hassle-free renting experience.

## PURPOSE AND GOALS

The primary goal of the House Renting App is to create a robust and user-friendly digital platform that simplifies the property rental process for both owners and renters. This app aims to bridge the gap between property owners and potential tenants by providing an intuitive interface to facilitate property listing, searching, booking, and management.

**Key goals include:**

1. **For Renters:**
    - Enable renters to browse and filter properties based on preferences such as location, price, and amenities.
    - Allow seamless property availability checks and secure booking/reservation processes.
    - Provide a personalized dashboard for managing bookings and inquiries.

2. **For Property Owners:**
   - Offer a platform to easily list and showcase rental properties with detailed information and images.
   - Enable owners to manage property availability, bookings, and tenant inquiries effectively.
   - Ensure transparency and control over rental processes through a dedicated owner portal.

**3. Technology Goals:**

- Leverage MongoDB for efficient, scalable data management of user profiles, property details, and booking records.Utilize React.js for a responsive and dynamic front-end experience.
- Build a secure and scalable backend using Express.js and Node.js to handle user authentication, property management, and data synchronization.

**4. Additional Goals:**

Ensure a seamless user experience with mobile responsiveness and real-time updates.Incorporate secure payment and notification features for enhanced functionality.By achieving these objectives, the app aspires to make property renting accessible, efficient, and hassle-free for all users.

## FEATURES

The House Renting App offers a range of features designed to meet the needs of both property owners and renters. The User Management system allows seamless registration and login, with role-based access providing personalized dashboards for owners and renters. Users can securely manage their profiles, with encrypted passwords

ensuring data security. The app facilitates Property Listing and Management, enabling owners to add, edit, and delete property details, including uploading images and updating availability status. Renters benefit from the Property Search and Discovery feature, which offers filters for location, price, and amenities, along with real-time updates on property availability.
A robust Booking System allows renters to book properties by selecting dates, while both renters and owners can manage bookings through detailed reservation histories. The app's Interactive User Experience includes responsive design for all device types, an embedded map for viewing property locations, and an image carousel for better visualization. Communication between users is streamlined with a Chat System, allowing direct interaction between renters and owners, while notifications ensure updates on bookings and inquiries.

For financial transactions, the app integrates Secure Payment Features, enabling renters to make online payments through reliable gateways, with a transaction history for reference. A Reviews and Ratings system allows renters to provide feedback on properties and owners to rate renters, fostering a trustworthy ecosystem. The app also supports an Admin Dashboard, where administrators can manage users, monitor activity, and generate reports, ensuring smooth operation and moderation.

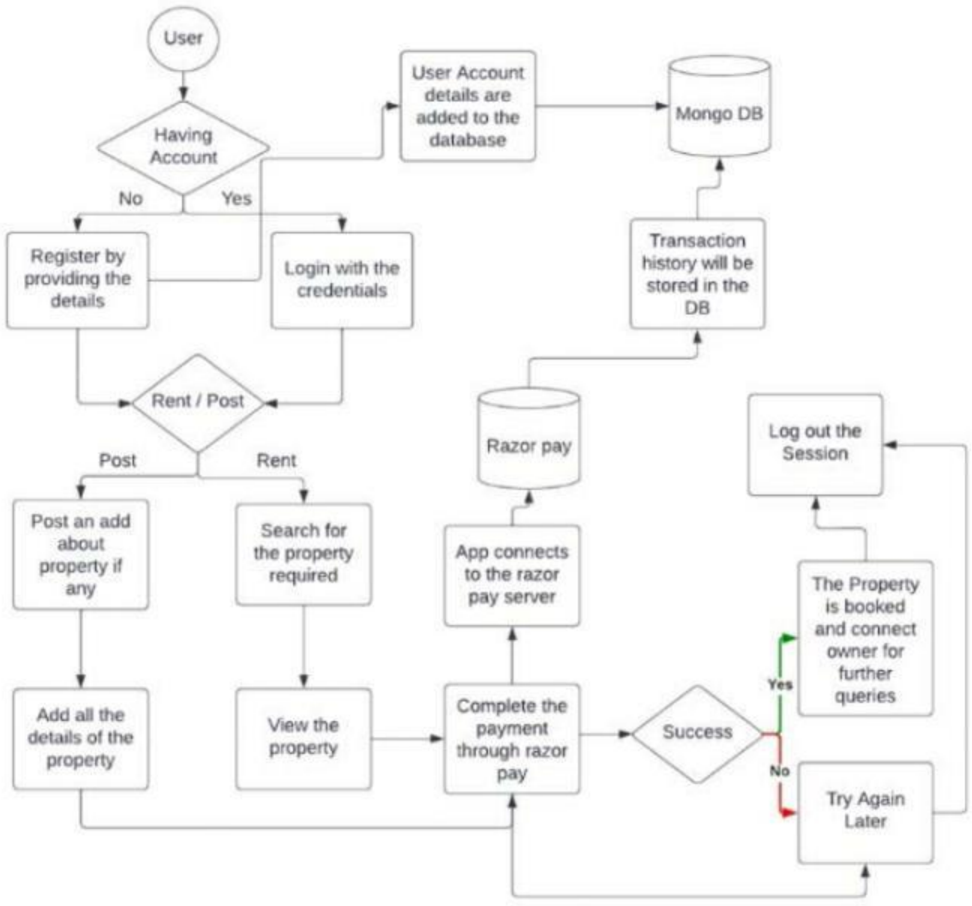
Finally, the app prioritizes Security and Data Management through encrypted data storage, regular backups, and secure session handling. These features collectively make the House Renting App a comprehensive, user-friendly, and secure platform for simplifying property rental processes.

□

# CHAPTER 3
# ARCHITECTURE

The architecture of the House Rental Application is designed to be scalable, efficient, and user-friendly. It follows a **MERN (MongoDB, Express.js, React.js, Node.js)** stack to create a full-stack application with a focus on modularity and performance. The architecture is divided into three main components: **Frontend**, **Backend**, and **Database**, with seamless communication between them.

□


User
Having Account
No
Yes
Register by providing the details
Login with the credentials
User Account details are added to the database
Mongo DB
Transaction history will be stored in the DB
Rent / Post
Post
Rent
Post an add about property if any
Search for the property required
Add all the details of the property
View the property
Razor pay
App connects to the razor pay server
Complete the payment through razor pay
Success
Yes
No
Log out the Session
The Property is booked and connect owner for further queries
Try Again Later

**1. Client-Side (Frontend):**

The Frontend is built using React.js, which interacts with the user directly and is responsible for rendering the user interface (UI). The key components of the frontend architecture include:

- React Components: Modular, reusable components (like property cards, booking forms, user profiles) manage the user interface.
- React Router: Handles navigation and routing between different pages like login, property listings, and booking details.
- State Management: The application uses React Hooks (useState, useEffect) for state management and dynamic updates to the UI. Optionally, Context API or Redux can be used for managing global application state (such as user authentication status).
- Axios: Used to make API requests to the backend for data retrieval and updates (like property data, bookings, user profiles).

**2.The Server-Side (Backend):**

The Backend is built using Node.js with Express.js, which serves as the application's server-side framework. Key components of the backend architecture include:

- Express.js: Manages API requests, routes, and handles HTTP methods (GET, POST, PUT, DELETE) for interactions between the client and server.
- JWT (JSON Web Tokens): Handles authentication and authorization. When users log in, a JWT is generated to authenticate subsequent API requests.
- Middleware: Various middlewares handle functionality like user authentication, data validation, and error handling.
- Controller Layer: Business logic is handled here for operations like property management (create, read, update, delete), user authentication, and booking management.

**3.Database Layer (MongoDB):**

The Database Layer is powered by MongoDB, a NoSQL database that stores and manages data such as user profiles, property listings, bookings, and reviews. Key components include:

- MongoDB: Stores all the application data in a flexible, scalable format. Collections like users, properties, bookings, and reviews are used to structure the data.
- Mongoose: An ODM (Object Document Mapper) library that simplifies database interactions. It defines schemas and models to structure the data in MongoDB, enabling data validation, querying, and manipulation.

**Communication Between Frontend and Backend:**

The communication between the Frontend and Backend is done via RESTful APIs. The frontend makes HTTP requests (GET, POST, PUT, DELETE) to the backend to interact with the database.

These API endpoints include:

- User Authentication API: Allows users to log in, register, and manage their session using JWTs.
- Property Management API: Handles adding, editing, listing, and deleting properties by the owner.
- Booking API: Manages booking creation, cancellation, and status updates.
- Review and Rating API: Allows renters to leave reviews and ratings for properties.

**Security Features:**

To ensure the security of sensitive data and prevent unauthorized access, the architecture includes :

- Password Hashing: Passwords are hashed using bcrypt.js before storing them in the database, ensuring that even if the database is compromised, user passwords remain secure.

- JWT-based Authentication: The backend uses JWTs for session management and authorization, ensuring only authenticated users can perform actions like making bookings or editing properties.
- HTTPS: Data transmission between the client and server is secured using HTTPS to prevent interception of sensitive data.

**Deployment and Hosting:**

The app is hosted and deployed across multiple platforms to ensure reliability and scalability:

- Frontend Deployment: The React app is deployed on Netlify or Vercel, which offer continuous deployment from GitHub and automatic scaling.
- Backend Deployment: The backend is deployed on Heroku, AWS EC2, or DigitalOcean, which provides server-side hosting for Node.js applications.
- Database Hosting: MongoDB is hosted on MongoDB Atlas for cloud-based management, scalability, and backups.

**Real-Time Communication (Optional):**

To enhance user experience, real-time features like notifications or chat could be integrated using:

- Socket.io: For real-time communication between users (renters and owners) in case of direct chat or notifications about booking status.

This architecture enables the House Renting App to be a robust, secure, and scalable solution for managing property rentals. It leverages the power of the MERN stack to handle large volumes of data, ensure real-time interactions, and provide a seamless experience for both renters and property owners.

□

## Technology Stack:

| Layer | Technology |
|---|---|
| Frontend | React.js, HTML5, CSS3 |
| Backend | Node.js, Express.js |
| Database | MongoDB, NoSQL |
| State Management | Redux or Context API |
| Authentication | JWT |
| Deployment | AWS, Netlify, MongoDB Atlas |

# CHAPTER 4

# SETUP INSTRUCTIONS

The following detailed setup instructions will guide you through the process of installing and running the House Rental Application locally on your system.

## 1. Prerequisites

Before you begin, ensure that the following software is installed on your system:

- Node.js (v14 or later): For backend development and managing dependencies.
- MongoDB: Either set up a local instance of MongoDB or use MongoDB Atlas for cloud-based database hosting.
- Git: For version control and managing project repositories.
- Code Editor: Visual Studio Code or any preferred editor.

## 2. Clone the Repository

- Clone the repository from GitHub to your local machine.
- bash
- git clone https://github.com/your-repository/house-renting-app.git  cd house-renting-app.

## 3. Install Backend Dependencies (Node.js)

Navigate to the backend folder (if separate) and install the required dependencies for the backend server.

- bash
- cd backend
- npm install

**4. Set Up MongoDB**

- Navigate to the frontend folder and install the dependencies for the React app.
- bash
- cd frontend
- npm install
- This will install packages like React.js, React Router, Axios, and Bootstrap/Tailwind CSS for building the UI and handling API requests.

**5.Install Frontend Dependencies (React.js)**

- Navigate to the frontend folder and install the dependencies for the React app.
- bash
- cd frontend
- npm install
- This will install packages like React.js, React Router, Axios, and Bootstrap/Tailwind CSS for building the UI and handling API requests.

**6. Set Up Environment Variables (Optional)**

- You may need to configure additional environment variables for both the backend and frontend (for example, for API keys, JWT secrets, or third-party services like payment gateways). Create a .env file in both the backend and frontend directories if it doesn't already exist.
  - Example for frontend .env: bash Copy code
    REACT_APP_API_URL=http://localhost:5000

**7. Start the Development Servers**

- Backend (Node.js)

□

- In the backend folder, run the following command to start the Express.js server:
- bash
- npm start
- This will run the backend server locally on port 5000 (or whatever port you set in the .env file).
- Frontend (React.js)
- In the frontend folder, run the following command to start the React development server:
- bash
- npm start
- This will launch the React app locally, usually on port 3000. The frontend will make API requests to the backend running on port 5000.

**8. Test the Application Locally**

Once both servers are running, open your browser and navigate to http://localhost:3000. You should see the House Renting App's homepage. Try logging in, browsing properties, and making bookings to test the functionality.

**9.Additional Configurations**

- SSL/HTTPS: For secure communication, consider setting up HTTPS for production using services like Let's Encrypt or an SSL certificate.
- Logging and Monitoring: Integrate tools like Winston (for logging) or Sentry (for error tracking) in production environments.

By following these steps, you can install and set up the House Renting App locally and run it for development and testing.

# CHAPTER 5

# RUNNING THE APPLICATION

**1. Start the MongoDB Service:**

○ Ensure the MongoDB service is running before starting the application.

```
mongodb
```

**2. Start the Backend Server:**

Navigate to the backend directory:

```
Cd backend
```

Start the backend server:

```
npm run start
```

○ The backend server will run at http://localhost:5173.

**3. Start the Frontend Server:**

Open a new terminal and navigate to the client directory:

```
cd frontend
```

Start the frontend development server:

```
npm run dev
```

○ The frontend will run at http://localhost:5173.

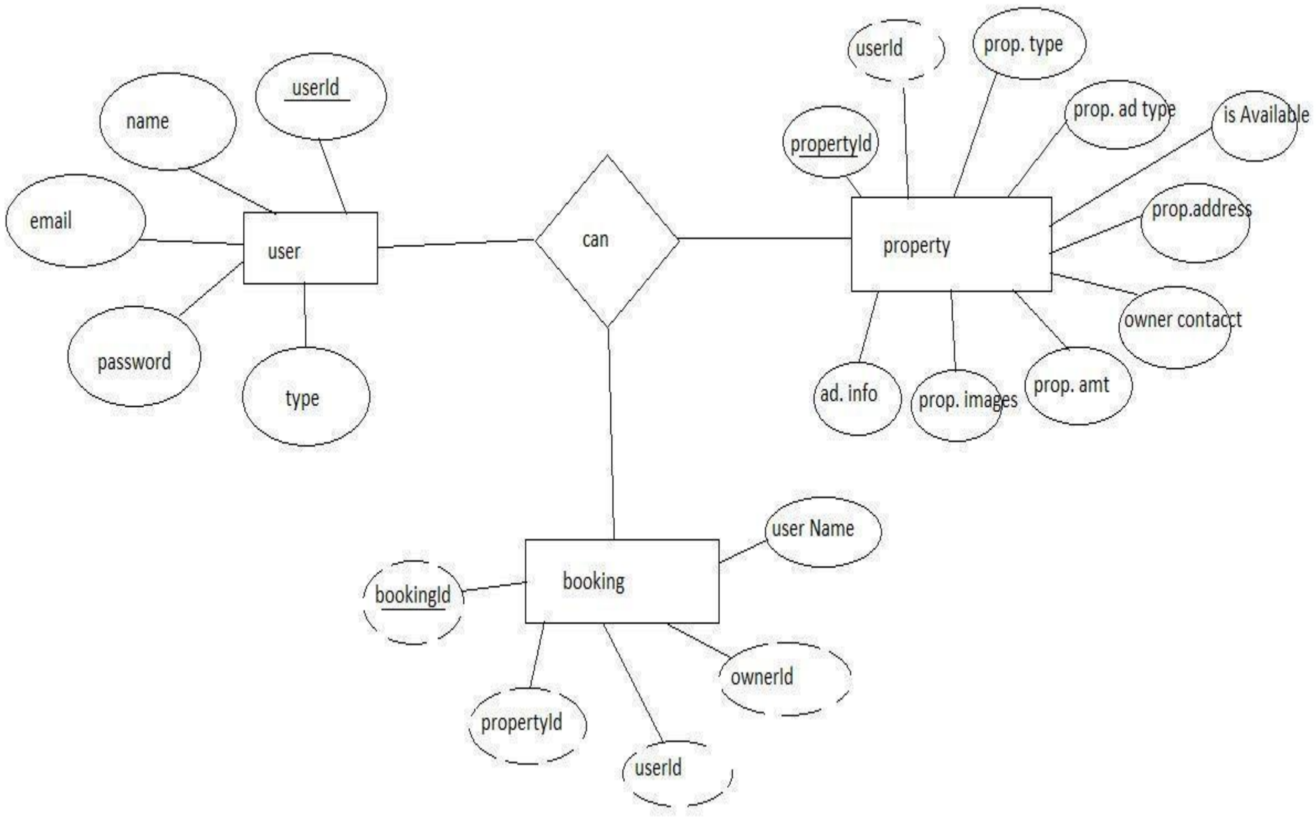
**Verify the Setup**

- Open your web browser and go to http://localhost:3000 to see the frontend of the house rental application.
- Test the functionality:

- Browse properties.
- Register/login as a tenant or landlord.
- Add new properties (if logged in as a landlord).
- Apply filters to search for properties.

Check both the backend terminal (localhost:5000) and the browser console for any errors or issues.

**ADDITIONAL NOTES :**



1. **Database Design and Schema Structure**
   Designing an efficient database schema that could handle diverse data (properties, bookings, users, reviews) was challenging. Ensuring that relationships between users, properties, and bookings were well-structured, without redundancy, required careful thought and the use of **MongoDB**'s flexible schema design.
2. **User Authentication and Authorization**
   Implementing secure user authentication, especially ensuring that **JWT (JSON Web Tokens)** were properly handled for session management, was tricky. Managing user roles (renters vs. property owners) and ensuring proper access control to different parts of the app (such as property creation or booking) added complexity.
3. **Payment Integration**
   Integrating a reliable payment gateway for bookings posed a challenge. Ensuring **secure transactions**, handling payments for deposits, and managing payment failures or cancellations required careful attention to security and API handling.
4. **Real-time Updates and Notifications**
   Providing **real-time updates** (for booking confirmations or property availability) was difficult to implement initially. The challenge was ensuring that users received timely notifications, especially in scenarios like last-minute cancellations or new bookings.
5. **Handling Large Data Sets for Property Listings**
   As the property listings grew, **performance optimization** became a key issue. Fetching large sets of property data, applying filters, and sorting them efficiently required optimizing queries and implementing pagination or lazy loading to improve speed and responsiveness.
6. **Cross-Browser Compatibility**
   Ensuring that the app worked seamlessly across various browsers (Chrome, Firefox, Safari, etc.) was a challenge. It required consistent testing and addressing compatibility

issues, particularly with styling, layout issues, and performance differences between browsers.

**7.Mobile Responsiveness and UI/UX**

Ensuring that the app was fully **responsive** and provided a smooth user experience across different screen sizes (mobile, tablet, desktop) was challenging. The interface needed to be intuitive, with clear navigation, user-friendly forms, and easy-to-read content, particularly for renters browsing properties on smaller devices.

**8. Security Vulnerabilities and Data Protection**

Securing sensitive data, such as user passwords, booking details, and payment information, was crucial. Implementing proper **encryption** and ensuring **data privacy** involved overcoming challenges related to **input validation**, preventing **SQL injection** (in case of future migrations to SQL databases), and safeguarding user data during API requests.

These challenges were significant but provided valuable learning opportunities, especially in areas like security, database management, and user experience. Overcoming them made the development of the House Renting App more robust and user-friendly.

# CHAPTER 6

# API DOCUMENTATION

This document outlines the key API endpoints for the **House Rental Application**, which covers the essential API endpoints needed for functionalities like property listings, user authentication, and managing rentals. It assumes that the backend is built using Express.js and that MongoDB is used as the database.

**Base URL**

arduino

https://api.houserentalapp.com/v1

## 1. Authentication API

These endpoints handle user login, registration, and authentication.

**POST /api/auth/register**

Description: Registers a new user (tenant or landlord).

Request Body:

```
{
  "username": "johndoe",
  "email": "johndoe@example.com",
  "password": "securepassword123",
  "role": "tenant"  // 'tenant' or 'landlord'
}
```

Response:

```
{
  "message": "User registered successfully."
}
```

**POST /api/auth/login**

Description: Logs in a user and returns a JWT token for subsequent req.

Request Body:

```
{
 "email": "johndoe@example.com",
 "password": "securepassword123"
}
 Response:
200 OK: Successful login.

{
 "token": "jwt_token_here"
}
```

**GET /api/auth/me**

Description: Get the details of the logged-in user using the provided JWT token.

Headers:

```
{
  "Authorization": "Bearer jwt_token_here"
}
```

Response:

200 OK: User details

```
{
  "username": "johndoe",
  "email": "johndoe@example.com",
  "role": "tenant"
}
```

## 2. Property Listing API

These endpoints allow landlords to create, update, delete, and view property listings.

**POST /api/properties**

Description: Create a new property listing (only accessible to landlords).

Request Body:

```
{
  "title": "Beautiful 2 Bedroom Apartment",
  "description": "A cozy 2-bedroom apartment in the heart of the city.",
  "price": 1200,
  "location": "123 Main St, City, Country",
  "imageUrls": ["url1", "url2"],
  "amenities": ["Wi-Fi", "Parking", "Air Conditioning"],
  "availableFrom": "2024-12-01"
}
```

Response:
201 Created: If property is added successfully.

```
{
"message": "Property created successfully."
}
```

**GET /api/properties**

Description: Get a list of all available properties.
Query Parameters (optional):

- priceMin: Minimum price for filtering.
- priceMax: Maximum price for filtering.
- location: Search by location.
- amenities: Search by amenities (comma-separated).
- Response:
- 200 OK: List of properties.

```
[
  {
   "id": "123",
   "title": "Beautiful 2 Bedroom Apartment",
   "price": 1200,
   "location": "123 Main St, City, Country",
   "availableFrom": "2024-12-01",
   "amenities": ["Wi-Fi", "Parking", "Air Conditioning"]
  },
  {
   "id": "124",
   "title": "Spacious Studio",
   "price": 800,
   "location": "456 Oak St, City, Country",
   "availableFrom": "2024-12-15",
   "amenities": ["Wi-Fi", "Elevator"]
  }
]
```

**GET /api/properties/:id**

Description: Get details of a single property by ID.
Response:
200 OK: Property details.

```
{
 "id": "123",
  "title": "Beautiful 2 Bedroom Apartment",
  "description": "A cozy 2-bedroom apartment in the heart of the city.",
  "price": 1200,
  "location": "123 Main St, City, Country",
  "imageUrls": ["url1", "url2"],
  "amenities": ["Wi-Fi", "Parking", "Air Conditioning"],
  "availableFrom": "2024-12-01"
}
```

## 3. Admin API (Optional)

These endpoints are for administrative functions like managing users and properties.

**GET /api/admin/users**

Description: Get a list of all users (admin functionality).

Response:

200 OK: List of users.

```
[
  {
    "id": "1",
    "username": "johnsmith",
    "email": "johnsmith@example.com",
    "role": "tenant"
  },
  {
    "id": "2",
    "username": "janedoe",
    "email": "janedoe@example.com",
    "role": "landlord"
  }
]
```

CHAPTER 7

# AUTHENTICATION & AUTHORIZATION

Authentication is a critical feature in the House Rental Application as it ensures secure and personalized access to the platform. Here's a detailed breakdown of the authentication process:

**1. User Registration and Login**

- **Renters** and **Property Owners** can sign up using their email addresses and create unique accounts.
- On the registration page, users must provide basic details like name, email, password, and role (either owner or renter).
- After registering, users can log in using their credentials. The authentication is handled through **JWT (JSON Web Token)** for secure access and session management.
- Upon successful login, users are redirected to their respective dashboards based on their role (owner or renter).

**2. Property Owners' Workflow**

**Add a Property**

- Owners can list properties they want to rent by clicking on the **“Add Property”** button from their dashboard.
- A form will appear where they can enter property details such as:
- Property title, description, and price
- Address, city, and zip code
- Available dates and amenities (e.g., Wi-Fi, parking, pool)  Upload images of the property.

- Once submitted, the property is added to the system and listed on the property search page for renters to view.

### Manage Property Listings

Property owners can view all their listed properties under the **“My Properties”** section of their dashboard.

- They can **edit** or **delete** properties as needed to ensure up-to-date information and availability.
- Owners can also mark properties as **available** or **booked** depending on the current status.
- **View Bookings**
- Owners can view **bookings** made by renters for their properties, including booking dates and renter details.
- They can approve or reject bookings depending on availability and other factors.

## 3. Renters' Workflow

### Search and Discover Properties

- Renters can search for properties by using various filters:
- **Location**: Search properties by city or neighborhood.
- **Price**: Filter by price range to find options within budget.
- **Property Type**: Search by property type (e.g., apartment, house, and studio).
- **Availability**: Filter properties that are available for booking on selected dates.
- The results are displayed dynamically with relevant property details and images.

### View Property Details

- Renters can view more information about a property, including:
- Detailed description of the property and amenities.
- Location on a map to understand the proximity to key areas.
- User reviews and ratings for feedback on the property.

### Book a Property

- Once a renter selects a property, they can proceed to book it by:

- Selecting check-in and check-out dates.
- Providing contact and payment details.
- After booking, renters receive a **booking confirmation** and can track their reservations in their dashboard.
- Renters can also view and **modify or cancel** their bookings, depending on the property owner.

**4. Communication Between Renters and Owners**

**Chat System**:

- Renters and owners can communicate directly through an integrated **chat system**.
- This allows renters to ask questions about the property, clarify booking details, or negotiate terms.
- **Notifications**:
- Renters and owners receive notifications for booking confirmations, cancellations, or changes to bookings.
- Email and/or in-app push notifications keep both parties updated on any critical information.

**5. Reviews and Ratings**

- After a booking ends, **renters can leave reviews** and **rate properties** based on their experience. This helps future renters in making informed decisions.
- **Owners** can also leave reviews for renters, rating them based on communication, punctuality, and adherence to the rental terms.

**6. Admin Workflow (Optional)**

- An **Admin Dashboard** is available to manage the entire platform, including:
- **User Management**: Admins can deactivate or block users who violate terms of service.

- **Property Management**: Admins can moderate and approve or remove listings that don't meet platform guidelines.
- **Analytics**: Admins can view usage statistics like the number of active users, number of bookings, and popular locations.
- **Content Moderation**: Admins can monitor and flag inappropriate reviews or content.

**7. Payment System (If Integrated)**

- Renters can **pay online** using integrated payment gateways for booking deposits or full payments.
- The **transaction history** is available to renters and owners for transparency.
- Owners can set **deposit amounts** and specific payment terms for their properties.

**8. Security and Privacy**

All sensitive data, such as passwords, are **hashed and encrypted** to ensure privacy.

- **JWT Authentication** ensures secure and authorized access to user accounts and data.
- All transactions are conducted via **HTTPS**, encrypting the data during transmission.

# CHAPTER 8

# TESTING

Testing a **House Rental Application** built using the **MERN Stack** (MongoDB, Express.js, React, Node.js) involves multiple aspects, including testing both the **backend** (API) and the **frontend** (React components). Below, I'll cover the key steps and best practices for testing various parts of your application.

## 1. Testing the Backend (Node.js + Express API)

For backend testing, you can use **Jest**, **Mocha**, or **Supertest** to test your API endpoints, business logic, and database integration.

**Key Backend Tests:**

- **Unit Tests**: Test individual components or functions (e.g., authentication functions).
- **Integration Tests**: Test API endpoints, database connections, and how they interact together.
- **E2E Tests**: End-to-end tests simulate a real user flow

Here's how you would write **unit tests** for the registration endpoint using **Jest** and **Supertest**:

```
// auth.test.js

const request = require('supertest');

const mongoose = require('mongoose');

const app = require('../app');  // Assuming you have an
Express app

const User = require('../models/User');

const bcrypt = require('bcryptjs');
```

```
// Mock the MongoDB connection for testing
jest.mock('mongoose', () => ({
  ...jest.requireActual('mongoose'),
  model: () => ({
    findOne: jest.fn(),
    save: jest.fn(),
  }),
}));

describe('User Registration', () => {
  beforeAll(() => {
    // Setup MongoDB connection here, if needed for the
tests
    mongoose.connect('mongodb://localhost:27017/test', {
useNewUrlParser: true, useUnifiedTopology: true });
  });

  afterAll(() => {
    // Cleanup
    mongoose.connection.close();
  });
  it('should register a user successfully with valid data',
async () => {
    // Mock the database interaction
    User.findOne.mockResolvedValue(null); // Simulate no
user found in DB
    User.prototype.save.mockResolvedValue(true);

    const res = await request(app)
      .post('/api/auth/register')
      .send({
        username: 'johndoe',
        email: 'johndoe@example.com',
        password: 'securepassword123',
        role: 'tenant'
      });

    expect(res.status).toBe(201);
    expect(res.body.message).toBe('User registered
successfully');
    expect(res.body.token).toBeDefined();
  });

  it('should return 400 if the user already exists', async ()
=> {
    // Mock the database interaction
    User.findOne.mockResolvedValue({}); // Simulate user
already exists

    const res = await request(app)
      .post('/api/auth/register')
      .send({
        username: 'johndoe',
        email: 'johndoe@example.com',
        password: 'securepassword123',
        role: 'tenant'
      });

    expect(res.status).toBe(400);
    expect(res.body.message).toBe('User already exists');
  });

  it('should return 500 if there is a server error', async ()
=> {
    // Mock a server error during saving user
    User.prototype.save.mockRejectedValue(new
Error('Database error'));

    const res = await request(app)
```

```
        .post('/api/auth/register')
        .send({
          username: 'johndoe',
          email: 'johndoe@example.com',
          password: 'securepassword123',
          role: 'tenant'
        });

      expect(res.status).toBe(500);
      expect(res.body.message).toBe('Server error');
    });
  });
```

Below is an example of writing **integration tests** for your **auth endpoints** (e.g., **registration** and **login**) using **Jest** and **Supertest**.

```
const request = require('supertest');
const app = require('../app');  // Express app
const mongoose = require('mongoose');

describe('Auth API', () => {
let server;

 beforeAll(async () => {
```

```
// Setup before all tests run

server = app.listen(5000);

await      mongoose.connect('mongodb://localhost:27017/houseRentalAppTest',      {
    useNewUrlParser: true, useUnifiedTopology: true });

});


afterAll(async () => {

// Cleanup after all tests run

await mongoose.connection.close();

server.close();

});


  it('should register a new user successfully', async () => {

  const res = await request(server).post('/api/auth/register').send({

  username: 'testUser',

  email: 'testuser@example.com',

  password: 'securePassword123',

  role: 'tenant',

 });


expect(res.status).toBe(201);

expect(res.body.message).toBe('User registered successfully.');

});
```

```
  it('should login successfully with correct credentials', async () => {
    const res = await request(server).post('/api/auth/login').send({
      email: 'testuser@example.com',
      password: 'securePassword123',
    });

    expect(res.status).toBe(200);
    expect(res.body.token).toBeDefined();   // Token should be returned on successful
        login
  });

  it('should return 400 for invalid login credentials', async () => {
    const res = await request(server).post('/api/auth/login').send({
      email: 'invalid@example.com',
      password: 'wrongPassword',
    });

    expect(res.status).toBe(401);  // Unauthorized status
    expect(res.body.message).toBe('Invalid credentials.');
  });
});
```

**End-to-End Testing (Optional)**

For **End-to-End (E2E)** testing, you can use tools like **Cypress** or **Playwright** to test entire user flows (from logging in to browsing properties). These tools interact with the app in a real browser and simulate actual user actions

Let's write an E2E test for the **user registration** and **login** flow.

1. **Create a New Test File**:

Create a new file inside the cypress/integration directory, e.g., auth_flow_spec.js.

2. **Test User Registration**:

Here's how you can write a simple test that simulates the user registration process:

```
// cypress/integration/auth_flow_spec.js


  describe('User Authentication Flow', () => {


  it('should register a user successfully', () => {
    cy.visit('http://localhost:3000/register'); // Navigate to the register page


   cy.get('input[name="username"]').type('testUser');  // Type in username
   cy.get('input[name="email"]').type('testuser@example.com');  // Type in email
  cy.get('input[name="password"]').type('Password123');  // Type in password
  cy.get('button[type="submit"]').click();  // Submit the form


  // Assert that the user is redirected to the home page or login page
  cy.url().should('include', '/login'); // Check that we are on the login page
```

```
      // Assert that the success message is shown
      cy.contains('User registered successfully').should('be.visible');
    });

  it('should log in a registered user successfully', () => {
    cy.visit('http://localhost:3000/login'); // Navigate to the login page

    cy.get('input[name="email"]').type('testuser@example.com');  // Type in email
   cy.get('input[name="password"]').type('Password123');  // Type in password
   cy.get('button[type="submit"]').click();  // Submit the login form

    // Assert that the user is redirected to the dashboard or home page
    cy.url().should('include', '/home'); // Assuming successful login redirects to the
  home page

 // Assert that the user is logged in (you could check if a logout button is visible)
 cy.contains('Welcome').should('be.visible');
});

it('should display an error message for invalid login', () => {
   cy.visit('http://localhost:3000/login'); // Navigate to the login page

 cy.get('input[name="email"]').type('invaliduser@example.com');  // Invalid email
 cy.get('input[name="password"]').type('wrongpassword');  // Invalid password
 cy.get('button[type="submit"]').click();  // Submit the login form


// Assert that the login error message is displayed
    cy.contains('Invalid credentials').should('be.visible');
 });


});
```

**Explanation:**

1. **User Registration**:
   - The test starts by visiting the /register page.
   - It fills in the **username**, **email**, and **password** input fields.
   - After submitting the form, it checks if the user is redirected to the /login page and if the success message is displayed.
2. **User Login**:
   - The test simulates logging in with the previously registered user's credentials.
   - It navigates to the /login page, fills in the login credentials, and checks if the user is redirected to the /home page (or a dashboard) after a successful login.
   - It also checks that a welcome message is displayed.
3. **Invalid Login**:
   - The test simulates an invalid login scenario with incorrect credentials.
   - It asserts that the error message "Invalid credentials" is displayed when the login fails.

**Step 4: Test Property Creation and Viewing Listings**

Let's add another test to simulate the creation of a property listing and viewing listings on homepage.

javascript

Copy code

```
describe('Property Listings Flow', () => {


  it('should create a new property listing', () => {
    // Login first (use cy.login() if you have a custom login command)
    cy.login('testuser@example.com', 'Password123');


    cy.visit('http://localhost:3000/new-property'); // Navigate to the property creation
      page


    cy.get('input[name="title"]').type('Beautiful 3-Bedroom House');
    cy.get('textarea[name="description"]').type('A lovely 3-bedroom house in the city
      center.');
    cy.get('input[name="price"]').type('1500');
    cy.get('input[name="location"]').type('City Center');
    cy.get('button[type="submit"]').click(); // Submit the property form


    // Assert that the property was created and the user is redirected to the listings page
    cy.url().should('include', '/properties');
    cy.contains('Beautiful 3-Bedroom House').should('be.visible');
  });
  it('should view a list of properties', () => {
    cy.visit('http://localhost:3000/properties'); // Visit the properties listing page


    // Check if the property is listed
    cy.contains('Beautiful 3-Bedroom House').should('be.visible');
    cy.contains('City Center').should('be.visible');
    cy.contains('$1500').should('be.visible');
  });


});
```

**Explanation:**

1. **Property Creation**:
    - The test assumes that the user is already logged in (you can use cy.login() if you have a custom Cypress command to handle login).
    - It navigates to the **new property** page (/new-property), fills in the property details (title, description, price, location), and submits the form.
    - After submitting, the test checks if the user is redirected to the properties listing page (/properties) and if the new property is listed.
2. **Viewing Properties**:
    - This test checks if the list of properties is being displayed correctly by visiting the properties page (/properties).
    - It verifies that the newly created property is visible on the page.

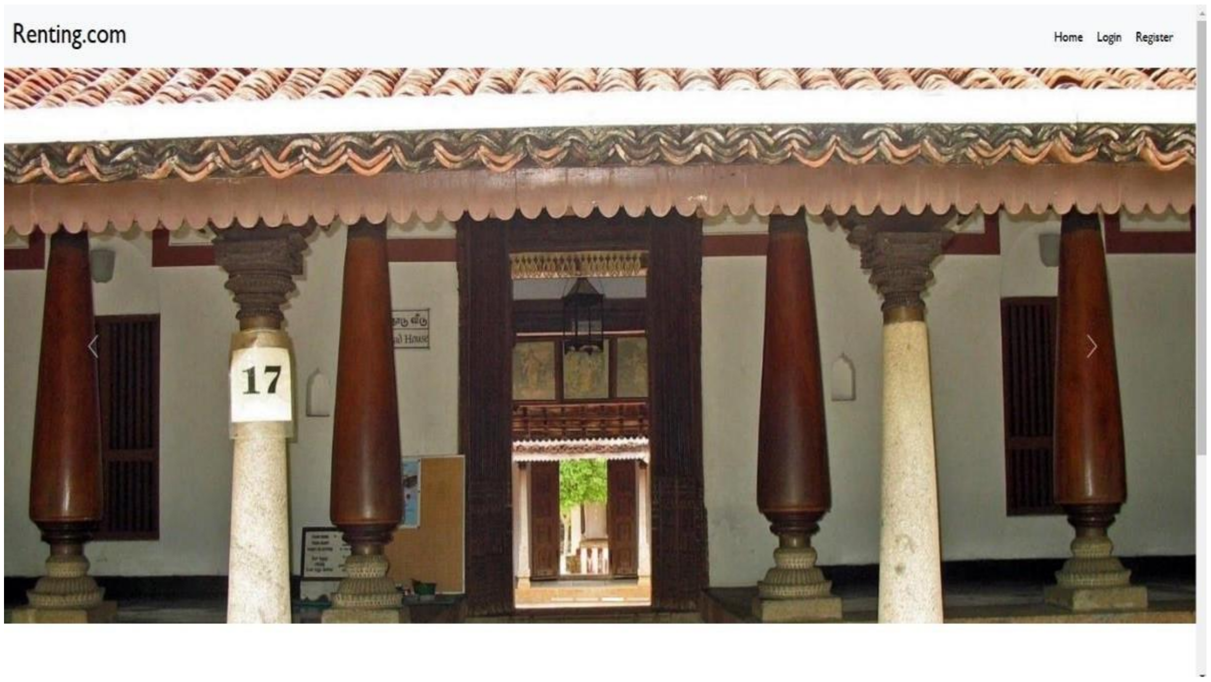
In this guide, we covered how to perform **End-to-End (E2E) testing** for a **House Rental Application** built with the **MERN stack** using **Cypress**. The tests simulate user interactions such as **registration**, **login**, **property creation**, and **viewing listings**. These tests ensure that the application functions as expected from the user's perspective and that all components (frontend and backend) work together seamlessly.

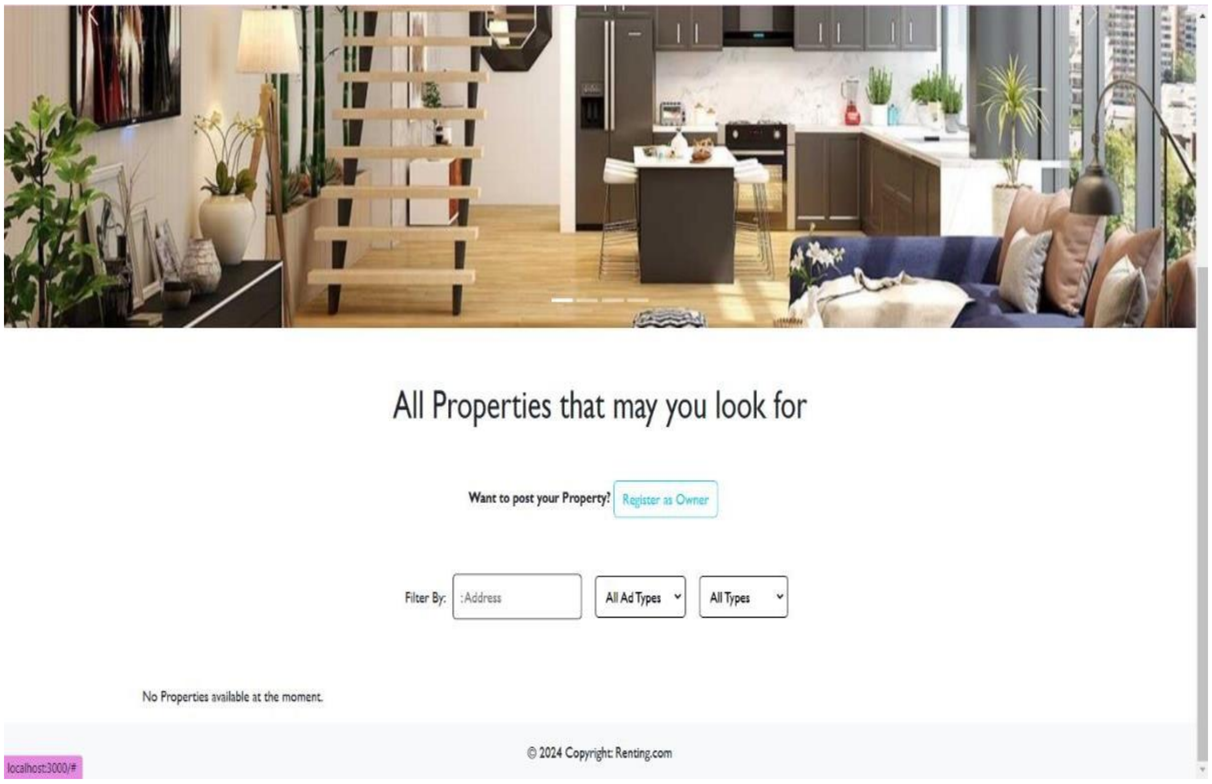
E2E testing helps to verify the entire flow of the application, providing confidence that your application is ready for production.

# CHAPTER 9

# SCREENSHOTS

-  **Landing page:**




All Properties that may you look for

Want to post your Property? Register as Owner

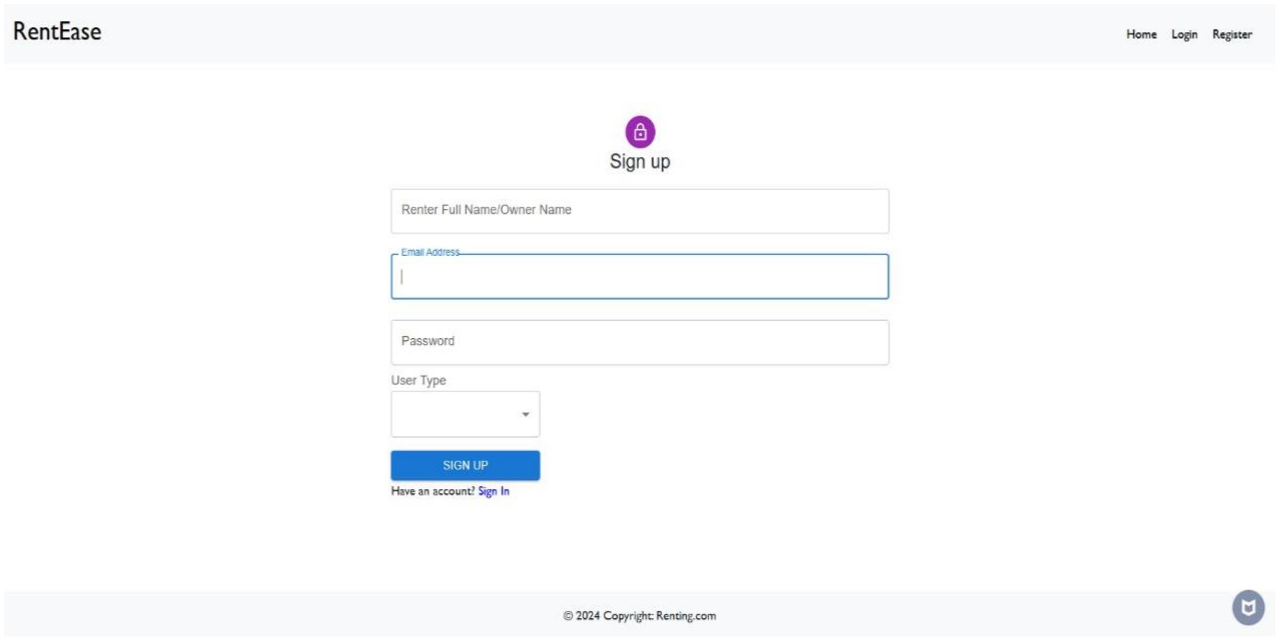
Filter By: :Address | All Ad Types | All Types

No Properties available at the moment.

© 2024 Copyright: Renting.com

localhost:3000/#

## Login and register page:


RentEase

Home Login Register

Sign up

Renter Full Name/Owner Name

Email Address

Password

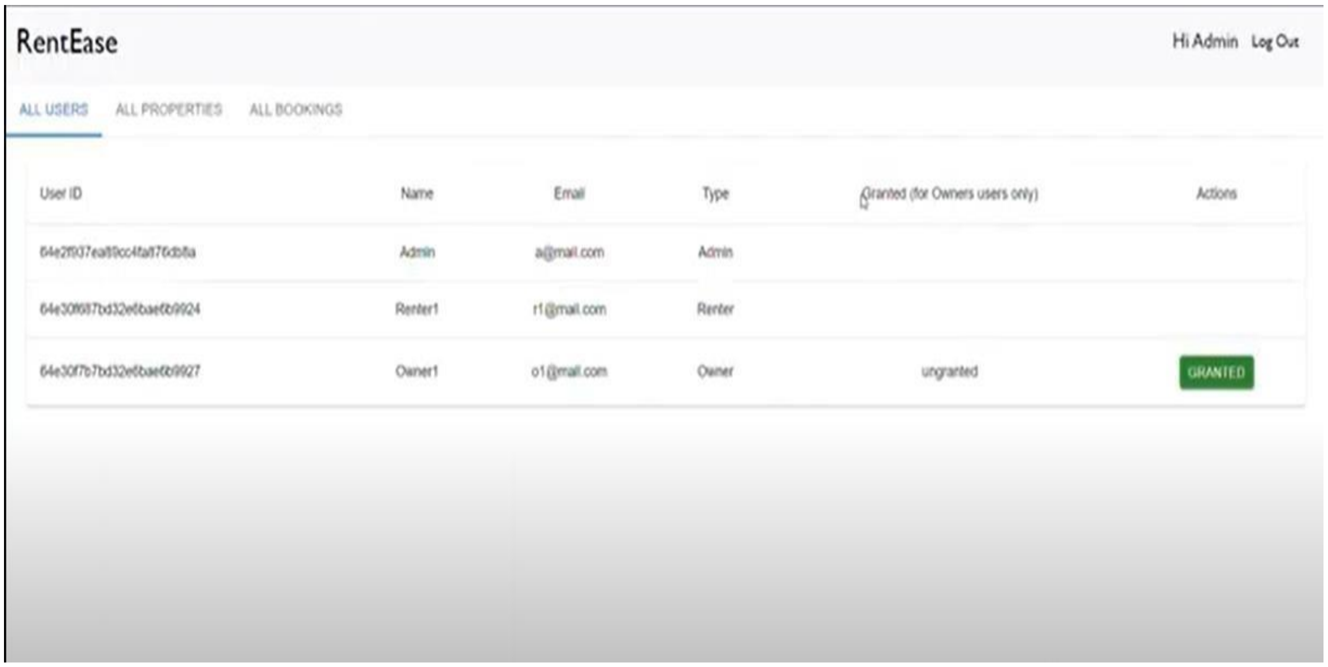
User Type

SIGN UP

Have an account? Sign In

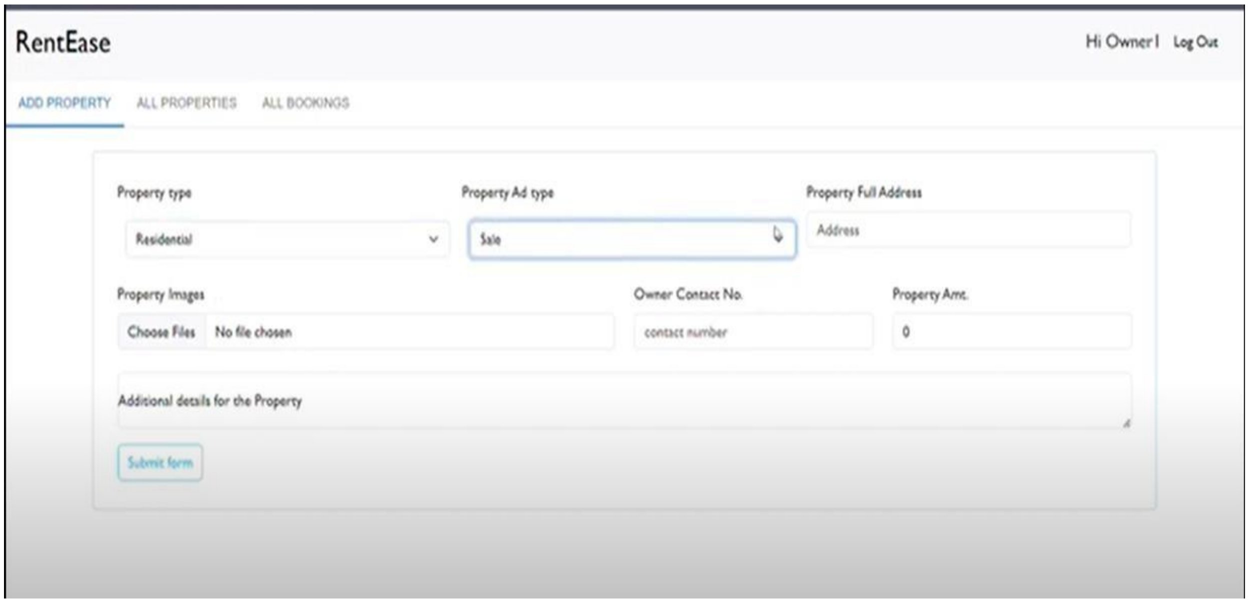
© 2024 Copyright: Renting.com

## Admin Panel:



| User ID | Name | Email | Type | Granted (for Owners users only) | Actions |
|---|---|---|---|---|---|
| 64e2f937ea89cc4fa876db8a | Admin | a@mail.com | Admin | | |
| 64e30f687bd32e6bae6b9924 | Renter1 | r1@mail.com | Renter | | |
| 64e30f7b7bd32e6bae6b9927 | Owner1 | o1@mail.com | Owner | ungranted | GRANTED |

## Owner Panel:



RentEase

Hi Owner! Log Out

ADD PROPERTY ALL PROPERTIES ALL BOOKINGS

Property type: Residential

Property Ad type: Sale

Property Full Address: Address

Property Images: Choose Files No file chosen

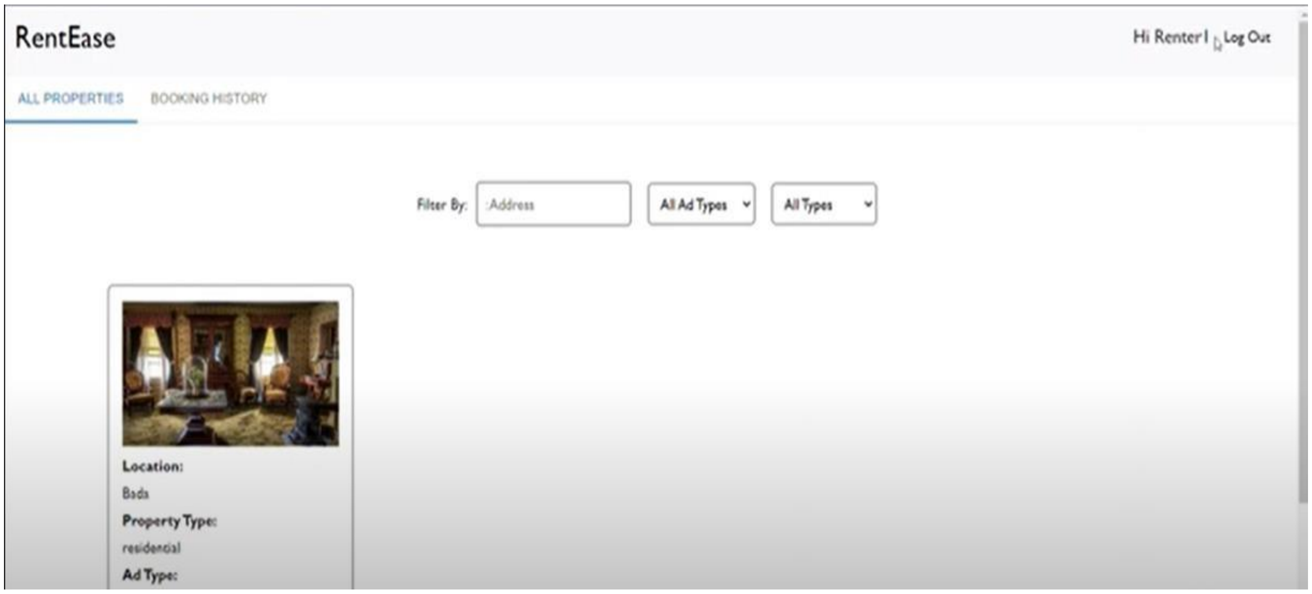
Owner Contact No.: contact number

Property Amt.: 0

Additional details for the Property

Submit form

## Tenant panel:



RentEase

Hi Renter! Log Out

ALL PROPERTIES BOOKING HISTORY

Filter By: Address All Ad Types All Types

Location:
Bada
Property Type:
residential
Ad Type:

## DEMO:

[https://drive.google.com/file/d/13ew88KNb81e5P0o6Z6gt7lyjVfVRgga-/view?usp=drive_link](https://drive.google.com/file/d/13ew88KNb81e5P0o6Z6gt7lyjVfVRgga-/view?usp=drive_link)

CHAPTER 10

# KNOWN ISSUES

While building and testing a **House Rental Application** using the **MERN stack** (MongoDB, Express.js, React, Node.js), several common issues can arise during development, deployment, and testing. Below are some **known issues** and challenges that might occur in such an application, along with potential causes and solutions.

**Issue 1: User Login Not Working After Registration**

- **Cause**: The login issue could arise due to improper handling of JWT token creation or session management.
- **Solution**: Ensure that JWT tokens are being generated and properly stored (either in localStorage or cookies). Double-check that the backend correctly validates the token and maintains the user session.

**Issue 2: Users Cannot Access Restricted Routes**

- **Cause**: Incorrect route protection could prevent users from accessing certain pages after login.
- **Solution**: Implement middleware that checks for authentication (e.g., verifying JWT tokens) on all protected routes to ensure proper access control.

**Issue 3: Registration Email Already Exists**

- **Cause**: During registration, users may encounter an error if their email is already registered in the database.
- **Solution**: Ensure that proper validation is in place to check for existing emails and provide a clear error message to the user (e.g., "Email already in use").

---

## 2. API & Backend Issues

**Issue 4: Property Listing Not Being Saved to the Database**

- **Cause**: The property may not be saving due to validation issues, incorrect data formats, or issues with MongoDB connection.
- **Solution**: Ensure that the API correctly handles validation for required fields, and check the database connection. Verify that the data being sent matches the expected format.

**Issue 5: Slow API Response**

- **Cause**: Performance issues can arise if the API queries the database inefficiently or retrieves too much data at once.
- **Solution**: Optimize API endpoints to handle smaller datasets (using pagination or filtering), and use **indexes** in MongoDB to speed up frequent queries.

**Issue 6: CORS (Cross-Origin Resource Sharing) Errors**

- **Cause**: CORS errors may occur when the frontend and backend are hosted on different domains or ports.
- **Solution**: Implement proper CORS configuration in the backend to allow specific origins to access your API. This can be done by setting up CORS middleware on the Express server.

---

**3. Frontend (React) Issues**

**Issue 7: Property Images Not Displaying**

- **Cause**: Images might not display if the file path is incorrect, or if the backend is not serving static assets properly.
- **Solution**: Ensure that the backend is set up to serve static files (e.g., images) and that the frontend URLs for images are correctly referencing the server's static file path.

**Issue 8: Forms Not Submitting Properly**

- **Cause**: This can happen due to issues in form handling, such as incorrect state management or missing input values.
- **Solution**: Check that the form fields are controlled by React state and that each form field correctly passes the data when submitted. Also, verify that the correct API endpoint is being called.

**Issue 9: UI Layout Breaks on Different Screen Sizes**

- **Cause**: The layout may not be responsive, causing UI elements to misalign on smaller screen sizes.
- **Solution**: Implement a responsive design using **CSS frameworks** (e.g., Bootstrap or TailwindCSS) and ensure that the application layout adapts to various screen sizes by using **media queries**.

**Issue 10: React Component Rendering Delays**

- **Cause**: Slow rendering can occur if unnecessary re-renders happen, especially with large data sets or complex components.
- **Solution**: Use **React.memo** to optimize rendering, **lazy loading** for large components, and **pagination** or **infinite scroll** to avoid rendering all items at once.

---

**4. Database (MongoDB) Issues**

**Issue 11: MongoDB Connection Issues**

- **Cause**: The app may fail to connect to MongoDB if the connection string is incorrect or MongoDB is down.
- **Solution**: Verify that MongoDB is running and check the connection string (for local or cloud databases). If using **MongoDB Atlas**, ensure that the IP whitelist and credentials are correctly configured.

**Issue 12: Data Duplication**

- **Cause**: Duplicate entries may occur due to a lack of unique constraints or checks before saving data.
- **Solution**: Implement proper validation in the backend to check for duplicate data before inserting it into the database. Use **unique indexes** in MongoDB for fields like email and property title.

**Issue 13: Database Schema Migrations**

- **Cause**: Changing the database schema (e.g., adding or removing fields) can break existing data or application functionality.
- **Solution**: Handle schema changes using migration tools or by carefully updating the schema and applying data transformations to existing records. Always back up the database before making changes.

---

**5. Deployment & Hosting Issues**

**Issue 14: Application Not Deploying Properly**

- **Cause**: Deployment issues can arise if the environment configuration is incorrect or the build process is not properly set up.
- **Solution**: Verify that all environment variables (e.g., MongoDB URI, JWT secret) are correctly set in the hosting platform. Ensure the build scripts are correctly configured for both frontend and backend.

**Issue 15: Backend API Not Reaching the Frontend After Deployment**

- **Cause**: The frontend may not be able to reach the backend API due to **CORS issues**, **incorrect endpoint URLs**, or **network configuration**.
- **Solution**: Double-check the API URL in the frontend to ensure it points to the correct deployed endpoint. Make sure the backend allows cross-origin requests from the frontend domain.

---

## 6. Testing Issues

### Issue 16: Unit Tests Failing Due to Mocking Issues

- **Cause**: Tests may fail if the backend or external services are not properly mocked during testing.
- **Solution**: Use **mocking libraries** like **Jest** or **Sinon** to mock external services and database queries in unit tests. For integration tests, use a mock or in-memory database for testing purposes.

### Issue 17: Flaky End-to-End (E2E) Tests

- **Cause**: Flaky tests might occur due to timing issues, where tests run before certain UI elements or API responses are fully loaded.
- **Solution**: Add appropriate **waits** or **delays** in E2E tests, use **Cypress commands** like cy.wait() or cy.intercept() to wait for API responses, and ensure tests interact with the page only after elements are visible.

# CHAPTER 11

# FUTURE ENHANCEMENT

**1. Advanced Search and Filtering Capabilities**
**Enhancement:**

- **Improve Property Search Filters**: Enhance the search functionality by adding more advanced filtering options, such as:
    - Price range sliders
    - Number of bedrooms, bathrooms
    - Amenities (e.g., parking, pet-friendly, furnished)
    - Location-based search with a map view (integrating with Google Maps or Mapbox)
    - Sorting options (e.g., by price, rating, proximity)

**Benefit:**

- Users will be able to more precisely find properties that meet their specific needs, increasing user engagement and satisfaction.

---

**2. AI-Powered Recommendations**
**Enhancement:**

- **Personalized Property Suggestions**: Use machine learning to provide personalized property recommendations based on user behavior and preferences (e.g., search history, location, and budget).
- **Recommendation System**: Integrate collaborative filtering or content-based algorithms to suggest properties that similar users have viewed or saved.

**Benefit:**

- A personalized experience will improve user retention and engagement by presenting them with relevant properties without manual search.

---

**3. Virtual Tours and 3D Floor Plans**
**Enhancement:**

- **Virtual Reality (VR) or 3D Property Tours**: Allow users to view properties through 360-degree virtual tours or interactive 3D floor plans.
- **Augmented Reality (AR)**: Enable potential renters to visualize the space with furniture or home decor through AR integration.

**Benefit:**

- Users can explore properties remotely, enhancing the user experience and reducing the need for in-person viewings, especially beneficial for long-distance renters.

---

**4. Mobile App Development**

**Enhancement:**

- **Develop a Mobile Application**: Build native mobile applications for iOS and Android using **React Native** or **Flutter** for better performance and access to device features (e.g., camera, GPS, notifications).

**Benefit:**

- A mobile app offers a more seamless and optimized experience, with users being able to interact with the platform on the go, increasing user engagement.

---

**5. Tenant and Landlord Communication Platform**

**Enhancement:**

- **Integrated Messaging System**: Allow tenants and landlords to communicate directly within the app. This could include:
    - Text chat functionality
    - Email notifications for new messages
    - Automated response templates for frequently asked questions (FAQs)

**Benefit:**

- Enhancing communication makes it easier for users to inquire about properties, schedule viewings, and resolve issues, creating a smoother transaction process.

---

**6. In-App Payments and Rent Management**

**Enhancement:**

- **Payment Gateway Integration**: Implement integrated payment systems such as **Stripe** or **PayPal** to allow tenants to pay rent directly through the app.
- **Rent Payment Tracking**: Include features for tenants to track their rent payments, due dates, and payment history. Landlords can also manage payments and generate receipts.

**Benefit:**

- Simplifies the payment process, reducing the chances of late payments and providing transparency for both tenants and landlords.

---

**7. Security and Data Privacy Enhancements**

**Enhancement:**

- **Two-Factor Authentication (2FA)**: Add an extra layer of security for users logging into their accounts by implementing two-factor authentication (2FA).
- **End-to-End Encryption**: Use end-to-end encryption for sensitive data, particularly when handling personal details, payment information, or chat messages between tenants and landlords.

**Benefit:**

- Strengthens user trust by ensuring their data is secure and protected from unauthorized access.

---

**8. Ratings, Reviews, and Reputation System**

**Enhancement:**

- **Review System for Landlords and Tenants**: Implement a ratings and review system where tenants can rate landlords and vice versa after each transaction.
- **Reputation Score**: Create a reputation scoring system based on user ratings that helps users make informed decisions about whom they are renting from or to.

**Benefit:**

- Builds trust within the community and helps ensure that both landlords and tenants have a positive experience.

---

**9. Automated Lease Agreement Generation**

**Enhancement:**

- **Digital Lease Agreement**: Integrate a system where landlords and tenants can generate, sign, and store digital lease agreements within the app.
- **E-Signature Integration**: Implement e-signature services such as **DocuSign** or **HelloSign** to allow parties to sign documents electronically.

**Benefit:**

- Streamlines the leasing process and eliminates paperwork, making it easier for both parties to formalize the rental agreement quickly.

---

**10. Property Maintenance Request System**

**Enhancement:**

- **Maintenance Request Portal**: Allow tenants to submit maintenance requests directly through the app. Include functionality for:
  - Uploading photos/videos of issues
  - Assigning maintenance tasks to specific landlords or property managers

    - Track the status of requests (open, in-progress, completed)

**Benefit:**

- Provides tenants with a direct way to report issues, improving tenant satisfaction. Landlords can also manage requests efficiently, improving their responsiveness.

---

**11. Advanced Analytics and Reporting**
**Enhancement:**

- **Property Performance Analytics**: Implement dashboards for landlords that provide insights into property performance, such as rental income, vacancy rates, tenant satisfaction, and maintenance costs.
- **Tenant Insights**: Allow tenants to track their payment history, expenses, and savings with respect to their rent.

**Benefit:**

- Landlords can make data-driven decisions regarding pricing and property management. Tenants benefit from better budgeting and financial tracking.

---

**12. Localization and Multi-Language Support**
**Enhancement:**

- **Multi-Language and Currency Support**: Implement a feature that allows the app to support multiple languages and currencies, especially if expanding to global markets.
- **Localization**: Adapt the app to support local units of measurement, rental laws, and tax policies for different regions or countries.

**Benefit:**

- This makes the app more accessible to a global audience, improving its usability and scalability across different regions.

---

**13. Social Media Integration**
**Enhancement:**

- **Social Media Logins**: Allow users to log in using their social media accounts (e.g., Facebook, Google) to make the registration and login process quicker and more convenient.
- **Social Sharing**: Enable tenants and landlords to share property listings on their social media profiles to reach a wider audience.

**Benefit:**

- Reduces friction during account registration and enhances marketing efforts through social sharing.

---

**14. Automated AI Chatbot for Support**

**Enhancement:**

- **AI-Powered Chatbot**: Implement a chatbot that can assist users with common queries (e.g., how to list a property, view payment history, etc.) and guide them through the application process.

**Benefit:**

- Provides 24/7 customer support and helps guide users through common tasks, improving overall user experience and reducing the workload on customer support teams.

---

## Conclusion

By integrating these **future enhancements**, the **House Rental Application** can stay competitive and offer improved user experiences. These features will not only improve functionality but also increase user engagement, trust, and retention, while making the platform scalable for future growth. Additionally, incorporating modern technologies like **AI**, **AR**, and **blockchain** can further elevate the platform's capabilities and align it with industry trends.