HOUSE RENT APP USING MERN

# INTRODUCTION

A **House Rent Management System** built using the MERN (MongoDB, Express.js, React.js, Node.js) stack is a platform designed to simplify and manage house renting activities for landlords and tenants. It enables streamlined communication, better organization, and improved user satisfaction while ensuring compliance with security and data protection standards. Below is a description of its key features:

1. **User Registration**: Landlords and tenants can create accounts to manage their profiles, properties, and rental agreements.
2. **Property Listings and Search**: Landlords can list rental properties with details such as location, rent, description, and images. Tenants can search for properties using filters like location, budget, and property type.
3. **Rental Application and Submission**: Tenants can apply for rental properties by submitting their details, documents, and rental preferences.
4. **Tracking and Notifications**: Both landlords and tenants can track the status of rental applications and agreements. Notifications via email or SMS are sent for application updates, rent payment reminders, and agreement changes.
5. **Interaction Between Landlord and Tenant**: The system facilitates direct communication between landlords and tenants, allowing discussions related to agreements, complaints, or property maintenance.
6. **Assignment and Scheduling**: The system helps landlords assign and schedule property maintenance tasks or rental agreement meetings with tenants.
7. **Security and Confidentiality**: Ensures data security through user authentication, encrypted communication, and role-based access control, while complying with relevant data protection laws.

This platform centralizes and optimizes the rental process, enhancing the experience for both landlords and tenants. Let me know if you'd like more technical details or assistance with the development process!

# DESCRIPTION

A **House Rent App** built using the MERN (MongoDB, Express.js, React.js, Node.js) stack streamlines rental processes for landlords and tenants. It enables property listings, rental applications, and communication between users with features like tracking, notifications, and scheduling. The app ensures data security through authentication, encryption, and role-based access control. It centralizes operations to enhance efficiency and user satisfaction in the rental ecosystem.

# SCENARIO

**Scenario: David, a tenant, is looking for a rental property and wants to apply for one using the House Rent App built with the MERN stack. Below is the user journey and system functionality:**

### 1. **User Registration and Login**:

* David visits the house rent app's website and clicks on the "Sign Up" button to create a new account.
* He fills out the registration form, providing his full name, email address, and a secure password.
* After submitting the form, David receives a verification email and confirms his account.
* He then logs into the app using his email and password.

### 2. **Property Search and Application**:

* Upon logging in, David is redirected to the dashboard, where he can browse available rental properties.
* He uses search filters like location, budget, and property type to find suitable options.
* David selects a property and clicks on the "Apply Now" button, filling out the application form with details like move-in date, personal information, and supporting documents (e.g., ID proof or employment verification).
* After reviewing his application, David submits it to the landlord.

### 3. **Tracking and Notifications**:

* After applying, David receives a confirmation message that his application has been successfully submitted.
* He navigates to the "My Applications" section of the dashboard, where he can track the status of his rental application in real-time.
* Notifications via email or SMS keep David informed about updates, such as the landlord reviewing his application or scheduling a property visit.

### 4. **Interaction Between Tenant and Landlord**:

* The landlord, Sarah, reviews David's application and contacts him through the app’s built-in messaging system.
* David receives a notification about Sarah’s message and accesses the chat feature to discuss further details, such as the rental agreement, move-in date, or property conditions.
* Both parties finalize the agreement terms within the app.

### 5. **Payment and Feedback**:

* Once the agreement is finalized, David makes the initial rent payment and deposit securely through the app’s payment gateway.
* After moving in, David receives a request for feedback on his experience with the house rent app and the landlord.
* He provides positive feedback, praising the seamless process and effective communication features of the platform.

### 6. **Admin Management**:

* The system administrator oversees all transactions and interactions on the platform.
* Admin assigns customer support tickets to agents based on their expertise and monitors issues like payment discrepancies or fraudulent activities.
* They ensure compliance with platform policies, handle disputes, and maintain the app's overall functionality.

This flow ensures a streamlined process for property rental and fosters transparent communication between landlords and tenants.

4o

# TECHNICAL ARCHITECTURE



### **1. Frontend** (React.js):

* **Purpose**:
  + Provides an interactive and user-friendly interface for tenants and landlords.
  + Handles dynamic content rendering and form submissions.
* **Components**:
  + **Authentication Pages**: Login, Sign-up, Forgot Password.
  + **Dashboard**: Separate dashboards for tenants and landlords.
  + **Property Search**: Filters for location, rent range, and property type.
  + **Rental Application Form**: To submit rental applications.
  + **Messaging System**: For tenant-landlord communication.
  + **Notifications**: Real-time updates using WebSockets (via Socket.IO).
* **Libraries/Tools**:
  + React Router (for navigation)
  + Axios/Fetch API (for API requests)
  + Redux/Context API (for state management)

### **2. Backend** (Node.js + Express.js):

* **Purpose**:
  + Acts as the server-side application to handle business logic and manage API endpoints.
  + Manages authentication, authorization, and communication with the database.
* **Key Modules**:
  + **Authentication**:
    - JWT-based token authentication for secure access.
    - Role-based access control (RBAC) for tenants, landlords, and admin.
  + **Property Management**:
    - APIs for adding, editing, and deleting property listings.
    - Search functionality with filters.
  + **Application Management**:
    - APIs to handle rental applications, document uploads, and status updates.
  + **Messaging**:
    - Real-time communication between tenants and landlords using WebSocket.
  + **Notifications**:
    - Email and SMS notifications via third-party services like Twilio or SendGrid.
  + **Payment Processing**:
    - Integration with payment gateways like Stripe or PayPal for handling rent payments.

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

* **Purpose**:
  + Stores application data persistently in a structured and efficient manner.
* **Key Collections**:
  + **Users**: Stores tenant and landlord profiles (name, email, role, etc.).
  + **Properties**: Contains property details like address, rent, images, and landlord ID.
  + **Applications**: Tracks rental applications, tenant details, and application status.
  + **Messages**: Manages chat data between tenants and landlords.
  + **Payments**: Logs payment transactions (amount, date, payer, and property ID).
* **Tools**:
  + Mongoose for schema management and object modeling.

### **4. APIs** (RESTful):

* **Purpose**:
  + Facilitates communication between the frontend and backend.
* **Example Endpoints**:
  + POST /api/auth/register – User registration.
  + POST /api/auth/login – User login.
  + GET /api/properties – Fetch property listings.
  + POST /api/properties – Add a property.
  + POST /api/applications – Submit rental application.
  + GET /api/messages/:conversationId – Fetch chat history.

### **5. Real-Time Features**:

* **WebSocket (Socket.IO)**:
  + Real-time messaging between tenants and landlords.
  + Updates for notifications and application status.

### **6. Payment Gateway Integration**:

* **Purpose**:
  + Allows secure online rent payments and deposit handling.
* **Flow**:
  + Tenant initiates payment via the app.
  + Backend processes the payment using gateways like Stripe or PayPal.
  + Confirmation sent to both tenant and landlord upon successful payment.

### **7. DevOps and Deployment**:

* **Hosting**:
  + Frontend: Deployed on platforms like Vercel or Netlify.
  + Backend: Hosted on cloud platforms like AWS, Azure, or Heroku.
  + Database: MongoDB Atlas for managed cloud databases.
* **CI/CD Pipeline**:
  + Automated testing, build, and deployment using tools like GitHub Actions or Jenkins.
* **Containerization**:
  + Docker to containerize the application for easy scalability and portability.

### **8. Security**:

* **Measures**:
  + Encrypted connections via HTTPS (SSL/TLS).
  + Secure password storage using bcrypt.
  + Data validation and sanitization to prevent injection attacks.
  + Role-based access control to restrict actions based on user type.

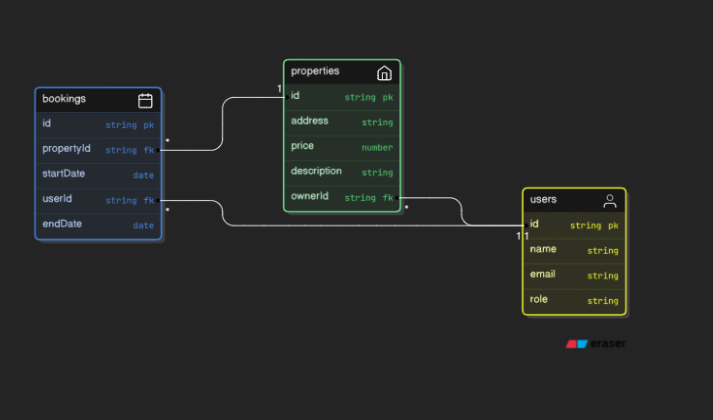
### **9. Additional Tools**:

* **Monitoring**: Tools like New Relic or Prometheus for tracking app performance and identifying bottlenecks.
* **Logging**: Winston or Morgan for logging server requests and errors.
* **Error Tracking**: Sentry for capturing and analyzing frontend and backend errors.

### **Data Flow Diagram**:

1. User interacts with the frontend (React.js).
2. Frontend sends API requests to the backend (Express.js).
3. Backend fetches/stores data in MongoDB via Mongoose.
4. Notifications/messages are sent to users via WebSocket or third-party services.
5. Payment gateway processes and validates transactions.

# ER DIAGRAM



An **ER (Entity-Relationship) Diagram** is a visual representation of the data structure for a system, showing entities, their attributes, and relationships. For a **House Rent App**, key entities include **Users** (with roles like Tenant and Landlord), **Properties** (details like address, rent, and type), **Rental Applications** (application status and tenant details), **Payments** (amount, date, and payer), and **Messages** (chat between users). Relationships connect entities, such as "Users post Properties," "Tenants submit Rental Applications," and "Payments are linked to Properties." The diagram helps in database design by defining the logical structure and ensuring data integrity. It typically uses notation like rectangles for entities, ovals for attributes, and diamonds for relationships.

# PRE-REQUISITES:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js:

### **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 to run JavaScript on the server-side.
* **Download:** <https://nodejs.org/en/download/>
* **Installation instructions:** <https://nodejs.org/en/download/package-manager/>

### **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, which handles server-side routing, middleware, and API development.
* **Installation:** Open your command prompt or terminal and run the following command:

bash

Copy code

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/>

### **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>

### **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.

* **To connect the database with Node.js, refer to:**  
  https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/

### **Front-end Framework**

Utilize **React.js** to build the user-facing part of the application, including property listings, rental applications, and admin dashboards.

* Use UI libraries like **Material-UI** or **Bootstrap** for enhanced design and responsiveness.

### **Version Control**

Use **Git** for version control, enabling collaboration and tracking changes throughout the development process. Platforms like **GitHub** or **Bitbucket** can host your repository.

* **Git download and installation:** <https://git-scm.com/downloads>

### **Development Environment**

Choose a code editor or Integrated Development Environment (IDE) that suits your preferences:

* **Visual Studio Code:** <https://code.visualstudio.com/download>

### **Steps to Run the House Rent App Locally**

#### **Clone the Repository:**

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to store the house rent app.
3. Execute the following command to clone the repository:

bash

Copy code

git clone https://github.com/awdhesh-student/house-rent-app.git

#### **Install Dependencies:**

1. Navigate into the cloned repository directory:

bash

Copy code

cd house-rent-app

1. Install the required dependencies for the frontend and backend:

bash

Copy code

cd frontend

npm install

cd ../backend

npm install

#### **Start the Development Server:**

1. To start the development server, execute the following command:

bash

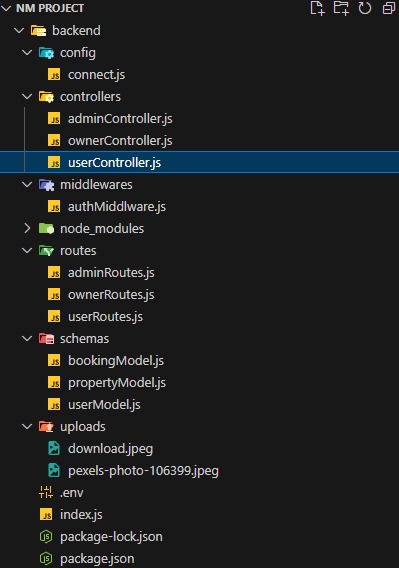
Copy code

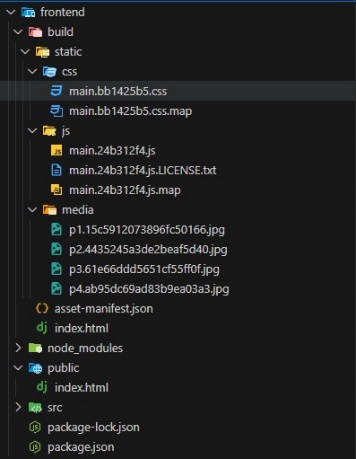
npm start

1. The **House Rent Management System** will be accessible at:  
   [**http://localhost:3000**](http://localhost:3000)

You have successfully installed and set up the **House Rent Management System** on your local machine. You can now proceed with further customization, development, and testing as needed.

# PROJECT STRUCTURE:





The first image is of frontend part which is showing all the files and folders that have been used in UI development

The second image is of Backend part which is showing all the files and folders that have been used in backend development

# APPLICATION FLOW**:**

**HOUSE RENT APP USING MERN**

### **1. Tenant/Ordinary User:**

* **Role:** Search for rental properties, submit applications, manage rental agreements, interact with landlords, and manage profile information.
* **Flow:**

1. **Registration and Login:**
   * Create an account by providing necessary information such as email and password.
   * Log in using the registered credentials.
2. **Property Search:**
   * Use search filters (e.g., location, rent, property type) to find rental properties.
   * View property details, including images, descriptions, and contact information.
3. **Rental Application Submission:**
   * Fill out the rental application form with details such as contact information, employment details, and references.
   * Submit the application for landlord review.
4. **Status Tracking:**
   * View the status of submitted applications on the dashboard or in the status section.
   * Receive real-time updates on the progress of applications.
5. **Interaction with Landlords:**
   * Connect with landlords using the built-in messaging feature.
   * Discuss property details, rental agreements, and other concerns.
6. **Profile Management:**
   * Manage personal profile information, including contact details and preferences.

### **2. Landlord:**

* **Role:** Manage rental properties, review tenant applications, communicate with tenants, and update property statuses.
* **Flow:**

1. **Registration and Login:**
   * Create an account using email and password.
   * Log in using the registered credentials.
2. **Property Listing:**
   * Add properties to the system by providing details such as location, rent, description, and images.
   * Update or delete property listings as needed.
3. **Application Review:**
   * Access the dashboard to view tenant applications for listed properties.
   * Approve or reject applications based on eligibility and requirements.
4. **Status Update:**
   * Update the status of properties (e.g., rented, available).
   * Notify tenants about application results or agreement changes.
5. **Tenant Interaction:**
   * Respond to tenant inquiries, discuss rental terms, and address feedback or complaints.

### **3. Admin:**

* **Role:** Oversee the operation of the house rent platform, manage users, properties, and ensure compliance with platform policies.
* **Flow:**

1. **Management and Monitoring:**
   * Monitor property listings, applications, and user interactions.
   * Enforce platform policies, terms of service, and privacy regulations.
2. **User Management:**
   * Manage landlord and tenant accounts, including registration, login, and profile information.
   * Address user concerns and ensure fair usage of the platform.
3. **Complaint Resolution:**
   * Address issues raised by tenants or landlords regarding platform misuse or disputes.
4. **Continuous Improvement:**
   * Implement new features and improve platform functionality and user experience.

Monitor platform security and address vulnerabilities promptly.

### **Project Flow:**

#### **Before Starting the Project:**

**Demo:** A project demo is available for reference:  
[Project Demo](https://drive.google.com/file/d/1YwXaHRBZJL_V7dcEK8SOmtPWZasAxccm/view?usp=drive_link)

Use the code in:  
[GitHub Repository](https://github.com/awdhesh-student/house-rent-management.git)

Or follow relevant videos/tutorials for better understanding.

### **Milestone 1: Project Setup and Configuration**

#### **1. Create Project Folders and Files:**

* Create separate folders for the **frontend** and **backend** to organize code.
  + **Client Folder** (for React.js).
  + **Server Folder** (for Node.js and Express.js).

#### **2. Install Required Tools and Software:**

**Backend Libraries:**  
Install the following tools and libraries for backend development:

* **Node.js**
* **MongoDB**
* **Bcrypt** (for password hashing).
* **Body-parser** (for parsing request bodies).

Run the following commands:

bash

Copy code

npm install express body-parser bcrypt mongoose cors

**Frontend Libraries:**  
Install the following tools and libraries for frontend development:

* **React.js**
* **Material-UI** (for design components).
* **Bootstrap** (for responsive design).
* **Axios** (for making API calls).

Run the following commands:

bash

Copy code

npm install react-router-dom @mui/material bootstrap axios

After the installation, the package.json files for the **frontend** will contain dependencies similar to this:

json

Copy code

{

"dependencies": {

"axios": "^1.4.0",

"bootstrap": "^5.2.0",

"react": "^18.2.0",

"react-dom": "^18.2.0",

"react-router-dom": "^6.12.0",

"@mui/material": "^5.12.0"

}

}

For **backend**, the package.json will contain:

json

Copy code

{

"dependencies": {

"express": "^4.18.0",

"body-parser": "^1.20.2",

"bcrypt": "^5.1.0",

"mongoose": "^7.0.0",

"cors": "^2.8.5"

}

}

You are now set to proceed with further customization, feature development, and testing of the **House Rent Management System**!

2/2

4o

Milestone 1**:**

**Project Setup and Configuration:**

To set up the project structure for a **House Rent App** using the **MERN stack (MongoDB, Express.js, React.js, Node.js)**, follow these steps for both the **frontend** and **backend** setup. Below are detailed instructions on folder structure and installation of required libraries.

### 1. Project Folder Structure:

Create a project directory and separate the frontend and backend into their respective folders.

bash

Copy code

house-rent-app/

│

├── client/ # Frontend React app

│ ├── public/

│ ├── src/

│ └── package.json

│

└── server/ # Backend Node.js app

├── config/

├── controllers/

├── models/

├── routes/

└── server.js

### 2. Install Required Tools and Libraries:

#### A. Backend Setup (Node.js + Express.js):

1. **Navigate to the server directory** and initialize a new Node.js project.

bash

Copy code

cd house-rent-app/server

npm init -y

1. **Install the required libraries for the backend**:

bash

Copy code

npm install express mongoose bcryptjs body-parser cors dotenv

* express: Web framework for Node.js.
* mongoose: MongoDB ODM to interact with MongoDB.
* bcryptjs: For hashing passwords.
* body-parser: Middleware for parsing incoming request bodies.
* cors: For enabling Cross-Origin Request Sharing (CORS).
* dotenv: For managing environment variables.

1. Create server.js to set up your Node.js server:

javascript

Copy code

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const bodyParser = require('body-parser');

require('dotenv').config();

const app = express();

// Middleware

app.use(cors());

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Connected to MongoDB'))

.catch(err => console.log(err));

// Routes

// Define your routes here

app.listen(5000, () => {

console.log('Server running on port 5000');

});

#### B. Frontend Setup (React.js):

1. **Navigate to the client directory** and initialize a new React app.

bash

Copy code

cd house-rent-app/client

npx create-react-app .

1. **Install the required libraries for the frontend**:

bash

Copy code

npm install react-router-dom axios @mui/material @emotion/react @emotion/styled bootstrap

* react-router-dom: For routing in React.
* axios: For making HTTP requests.
* @mui/material: Material UI components.
* @emotion/react and @emotion/styled: Styling for Material UI components.
* bootstrap: CSS framework for responsive design.

1. Create a simple App.js to set up routing and basic UI:

javascript

Copy code

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import { Container } from '@mui/material';

import 'bootstrap/dist/css/bootstrap.min.css';

function App() {

return (

<Router>

<Container>

<h1>House Rent App</h1>

<Routes>

<Route path="/" element={<h2>Welcome to House Rent App</h2>} />

{/\* Add other routes here \*/}

</Routes>

</Container>

</Router>

);

}

export default App;

### 3. package.json for Frontend:

Once the frontend dependencies are installed, your package.json should look similar to this:

json

Copy code

{

"name": "house-rent-app",

"version": "0.1.0",

"private": true,

"dependencies": {

"@emotion/react": "^11.10.5",

"@emotion/styled": "^11.10.5",

"@mui/material": "^5.10.9",

"axios": "^1.3.4",

"bootstrap": "^5.2.3",

"react": "^18.2.0",

"react-dom": "^18.2.0",

"react-router-dom": "^6.4.3",

"react-scripts": "^5.0.1"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

### 4. Environment Variables:

* In the server folder, create a .env file to store sensitive data like MongoDB URI and other configurations.

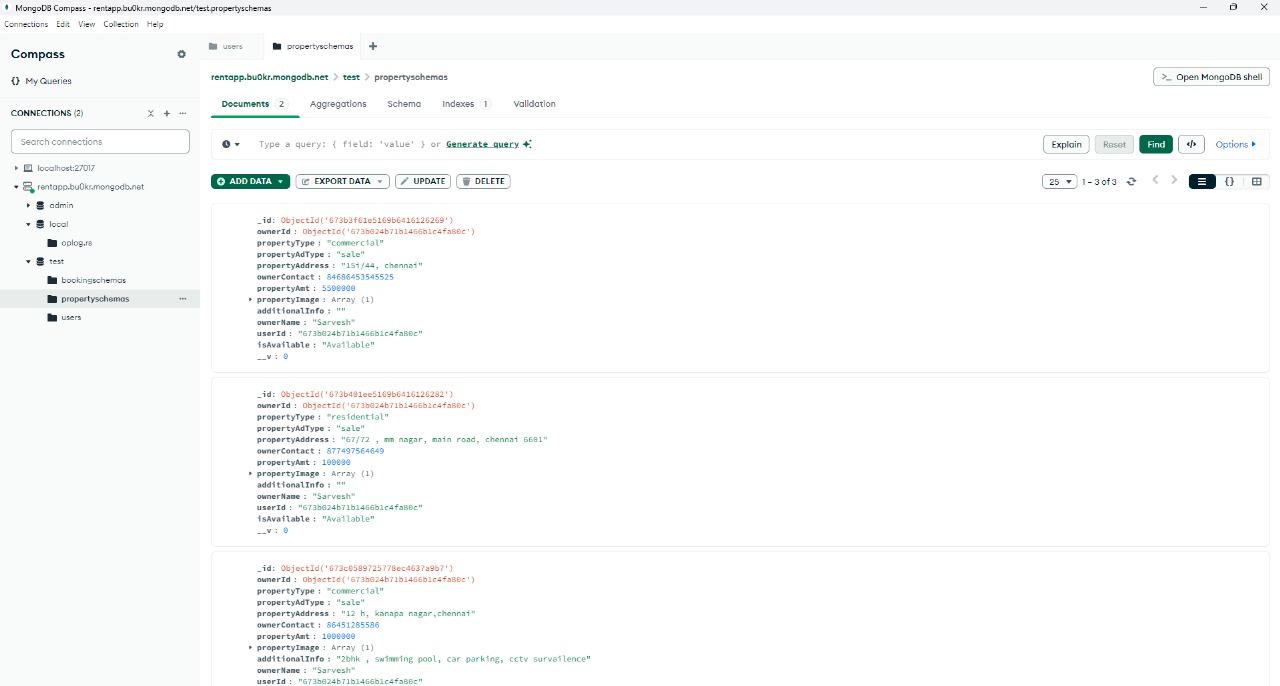
env

Copy code

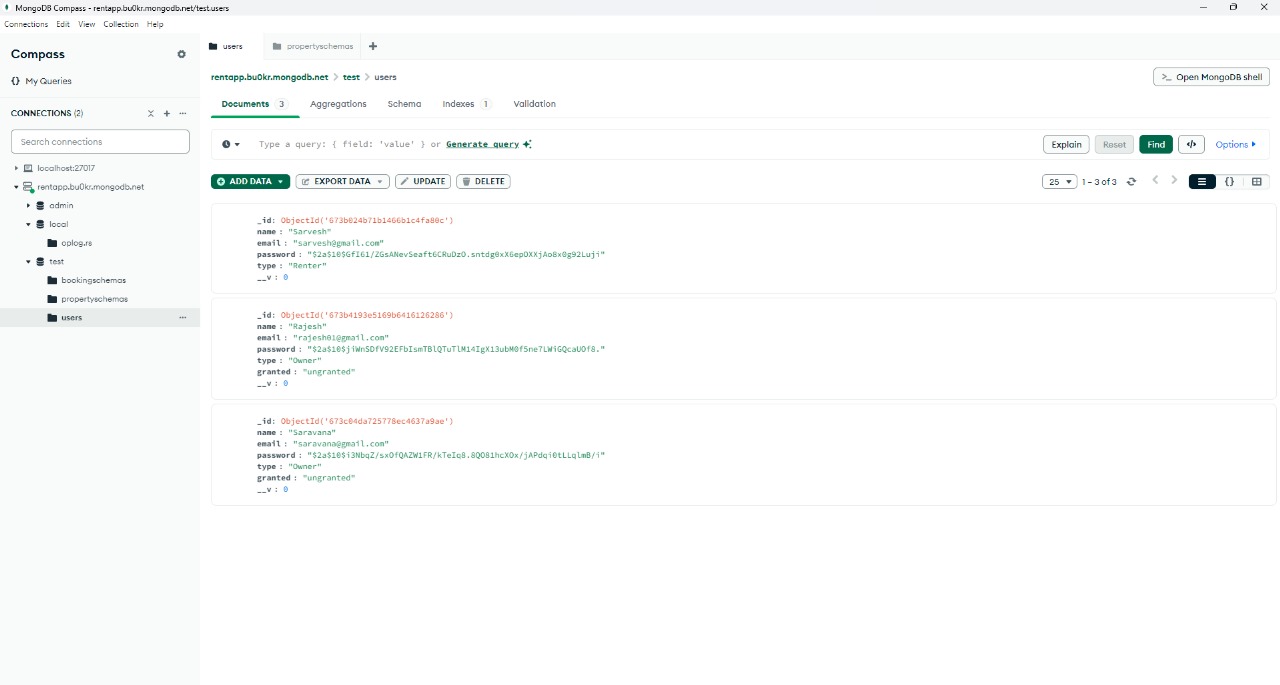
MONGO\_URI=mongodb://localhost:27017/house-rent-app

### 5. Final Notes:

* Ensure MongoDB is installed and running locally or use a cloud database service like MongoDB Atlas.
* Set up API routes in the server/routes folder, and create models for data storage in the server/models folder.
* Implement authentication (e.g., JWT) and business logic (e.g., house listing, user registration) as per the app’s requirements.



After the installation of all the libraries, the package.json files for the backend looks like the one mentioned below.



Milestone 2:

### 1. **Set Up Project Structure**:

* **Initialize Project**:
  + Create the main project folder and set up the backend (server) folder.
  + Initialize the Node.js project with npm init to generate package.json.
* **Install Necessary Dependencies**:
  + **Backend libraries**: Install essential libraries for backend development.
    - express: To create the server and handle HTTP requests.
    - mongoose: To interact with MongoDB and define data models.
    - bcryptjs: To hash passwords for user authentication.
    - jsonwebtoken (JWT): To generate tokens for user authentication.
    - body-parser: To parse incoming request bodies (for API calls).
    - cors: To handle Cross-Origin Resource Sharing and enable communication between frontend and backend.
    - dotenv: To manage environment variables like MongoDB URI and JWT secret.

### 2. **Create Express.js Server**:

* Set up an **Express.js server** to handle incoming HTTP requests.
* **Connect to MongoDB**: Use mongoose.connect() to link the server with the MongoDB database.
* Set up necessary **middleware**:
  + **Body-parser**: For parsing incoming JSON requests.
  + **CORS**: To ensure frontend and backend can communicate seamlessly, especially if they are hosted on different domains.

### 3. **Define API Routes**:

* **Authentication Routes**:
  + Implement routes for user registration, login, and token generation using **JWT**.
  + Create route handlers for user registration (with email and password) and login (validating email and password).
* **User Routes**:
  + Create API routes to manage user-related tasks like viewing user profiles, updating user details, etc.
  + Add **authentication middleware** to protect routes that require the user to be logged in.
* **Transaction Routes**:
  + Allow users to initiate transactions, transfer funds between accounts, and manage their rent payments.
  + Implement logic for updating user account balances and storing transaction details.
* **Admin Routes**:
  + Implement routes that only admin users can access, such as fetching all users, transactions, and houses listed for rent.

### 4. **Implement Data Models**:

* **User Model**:
  + Define a schema with fields like name, email, password, and isAdmin (to distinguish admin users).
  + Use **Mongoose** to define the schema and interact with MongoDB.
* **Transaction Model**:
  + Define a schema for transactions, including fields like fromUser, toUser, amount, and date.
  + This will allow tracking rent payments, transfers, and other financial transactions within the app.
* **House Model**:
  + Define a schema to represent house listings with fields like owner, location, price, and description.
  + This model will store details about houses available for rent and allow users to view listings.

### 5. **User Authentication**:

* Implement **JWT authentication**:
  + On registration, hash the user's password using **bcryptjs** and store the hashed password.
  + On login, validate the provided password against the stored hash.
  + On successful login, generate a **JWT token** containing the user's ID and return it to the frontend.
  + Protect certain routes (e.g., user profile) by validating the JWT token with middleware.
* Implement **authentication middleware** to secure routes that require the user to be logged in.

### 6. **Handle New Transactions**:

* Allow users to **make rent payments** or other financial transactions to other users.
* When a transaction is initiated, update the **transaction model** and the **user model** (adjust account balances).
* Handle edge cases such as insufficient funds and invalid user accounts.

### 7. **Admin Functionality**:

* Admin users should be able to access routes that provide data on:
  + **All users**: List of all users registered in the system.
  + **All transactions**: History of all transactions made on the platform.
  + **All house listings**: View and manage all houses available for rent.
* Implement appropriate **controllers and routes** for these admin-specific functionalities.

### 8. **Error Handling**:

* Implement a global **error handling middleware** to catch errors throughout the app.
  + Return appropriate error messages and **HTTP status codes** for different types of errors (e.g., validation errors, server errors).
  + Ensure that sensitive information is not exposed in error responses.
* Handle common errors like:
  + User already exists (for registration).
  + Invalid credentials (for login).
  + Insufficient funds (for transactions).
  + Invalid route access (unauthorized or forbidden access).

### 9. **Additional Considerations**:

* **Validation**: Ensure that input data is properly validated before interacting with the database. This can be done using **Joi** or **express-validator**.
* **Real-Time Updates**: If you need real-time features (e.g., displaying updated balance after a transaction), you can use **Socket.io** for real-time communication between the frontend and backend.
* **Security**:
  + Use **HTTPS** for secure communication.
  + Protect sensitive data like user passwords and JWT tokens.
  + Regularly update dependencies to avoid security vulnerabilities.

By following this structure, you'll have a solid backend architecture for your **House Rent App** using the **MERN** stack. This will handle everything from user registration and authentication to managing transactions and admin functionalities.

Milestone 3:

**Database Development**

In a **House Rent App**, the backend database is crucial for storing and managing various types of data. The **MongoDB** database, accessed via **Mongoose**, is used to model and manage this data. Here’s an outline of how to structure the database and define the models for different functionalities in the app.

### 1. **Key Entities for the Database:**

The primary entities for a House Rent App will be:

* **Users**: Store information about tenants, landlords, and admin users.
* **Houses**: Store information about properties available for rent.
* **Transactions**: Track financial transactions between tenants and landlords.
* **Admins**: Manage and control overall app data such as users, houses, and transactions.

### 2. **MongoDB Setup:**

Before developing the database structure:

* Set up a **MongoDB database** either locally or by using a cloud-based service like **MongoDB Atlas**.
* Store sensitive credentials, like the MongoDB URI and JWT secret, in environment variables (.env file).

### 3. **Database Models and Structure:**

Each entity will have a **Mongoose schema** to define the structure of data and manage interactions with the MongoDB database. Below is the explanation of each model and its fields:

#### **User Model:**

* **Purpose**: To manage tenant, landlord, and admin user details.
* **Key Fields**:
  + **Personal Information**: Name, Email, Password (hashed).
  + **Role**: Determines whether the user is a tenant, landlord, or admin.
  + **Additional Info**: Address, phone number, etc.
  + **Date Joined**: Timestamp when the user created their account.

#### **House Model:**

* **Purpose**: To store details about houses listed for rent.
* **Key Fields**:
  + **Owner**: Reference to the user (landlord) who owns the property.
  + **Property Details**: Address, rent price, number of bedrooms, description, etc.
  + **Availability**: Whether the house is available for rent.
  + **Date Listed**: When the house was listed.

#### **Transaction Model:**

* **Purpose**: To record transactions (rent payments, security deposits, etc.) between tenants and landlords.
* **Key Fields**:
  + **From User**: The tenant making the payment.
  + **To User**: The landlord receiving the payment.
  + **Transaction Type**: Rent payment, security deposit, etc.
  + **Amount**: The monetary value of the transaction.
  + **Date**: The date of the transaction.

#### **Admin Model:**

* **Purpose**: Admins manage the system and can access data for oversight.
* **Key Fields**:
  + **Admin Info**: Basic info about the admin (name, email, etc.).
  + **Permissions**: Admins have full access to all users, houses, and transactions.

### 4. **Relationships between Models:**

* **User and House**: A one-to-many relationship. A **landlord** (user) can have multiple **houses** listed for rent, while each **house** is associated with one **landlord**.
* **User and Transaction**: A many-to-many relationship. A **tenant** (user) can initiate multiple **transactions**, and a **landlord** (user) can receive multiple **transactions**.
* **Admin and User/House/Transaction**: An **admin** has access to all **users**, **houses**, and **transactions**, but their role doesn’t require linking to individual records. Instead, the admin's access controls the overall functionality.

### 5. **CRUD Operations for Each Model:**

* **User Model**: Perform **Create, Read, Update, Delete** operations for user management. This includes user registration, updating profiles, and authentication for login.
* **House Model**: Handle the **listing, updating, and deleting** of house details. Admins and landlords have control over house data.
* **Transaction Model**: Handle the **creation** of transactions (payments) and update the status of payment for rent, deposits, etc.
* **Admin Model**: Admins can access and manage all users, transactions, and house listings. CRUD operations are available to them but typically used to monitor and control data rather than interact with individual records as users do.

### 6. **Validation and Security Considerations:**

* **Password Encryption**: Use a hashing algorithm (e.g., bcrypt) to securely store user passwords in the database.
* **JWT Authentication**: Implement JSON Web Token (JWT) for user authentication, ensuring secure communication between the frontend and backend.
* **Data Validation**: Ensure all input fields are validated before being stored in the database. Use libraries like Joi or express-validator to perform server-side validation.

### 7. **Performance and Scalability Considerations:**

* **Indexes**: Add indexes to frequently queried fields like email, role, and house location to optimize search performance.
* **Database Sharding**: As the app grows, consider using database sharding or replica sets in MongoDB for better scalability and availability.

Milestone 4**:**

**Frontend Development: Setup React Application:**

1. **Setup React Application**: Install React.js, React Router, Axios, Material UI, and Bootstrap to build the app's structure and functionality.
2. **UI Design**: Create reusable components like buttons, cards, forms, and navigation bars for consistency and a clean layout.
3. **Responsive Design**: Use Material UI and Bootstrap for mobile-friendly and adaptive layouts.
4. **State Management**: Use React's useState for simple state management and Redux for more complex state handling.
5. **API Integration**: Use Axios to fetch data from the backend, such as house listings and user details.
6. **Authentication**: Implement JWT authentication for user login and protect routes to ensure secure access.
7. **Routing**: Use React Router for navigation between pages like Home, Profile, and House Details.
8. **Form Handling**: Manage user registration, login, and house rental forms with validation and error handling.
9. **Error Handling**: Provide clear feedback on errors and success messages during user actions.
10. **Security**: Ensure secure data transmission with HTTPS and handle authentication tokens to protect sensitive routes.

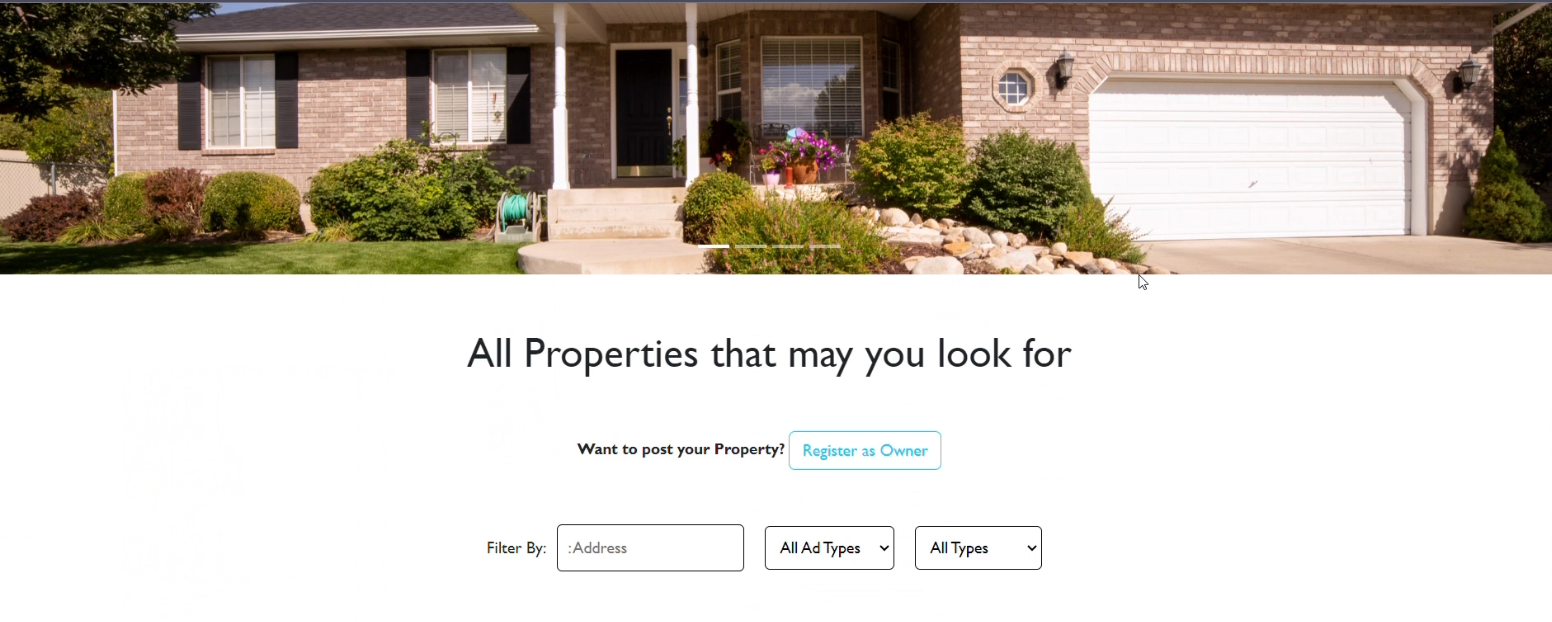
4o mini

Milestone 5**:**

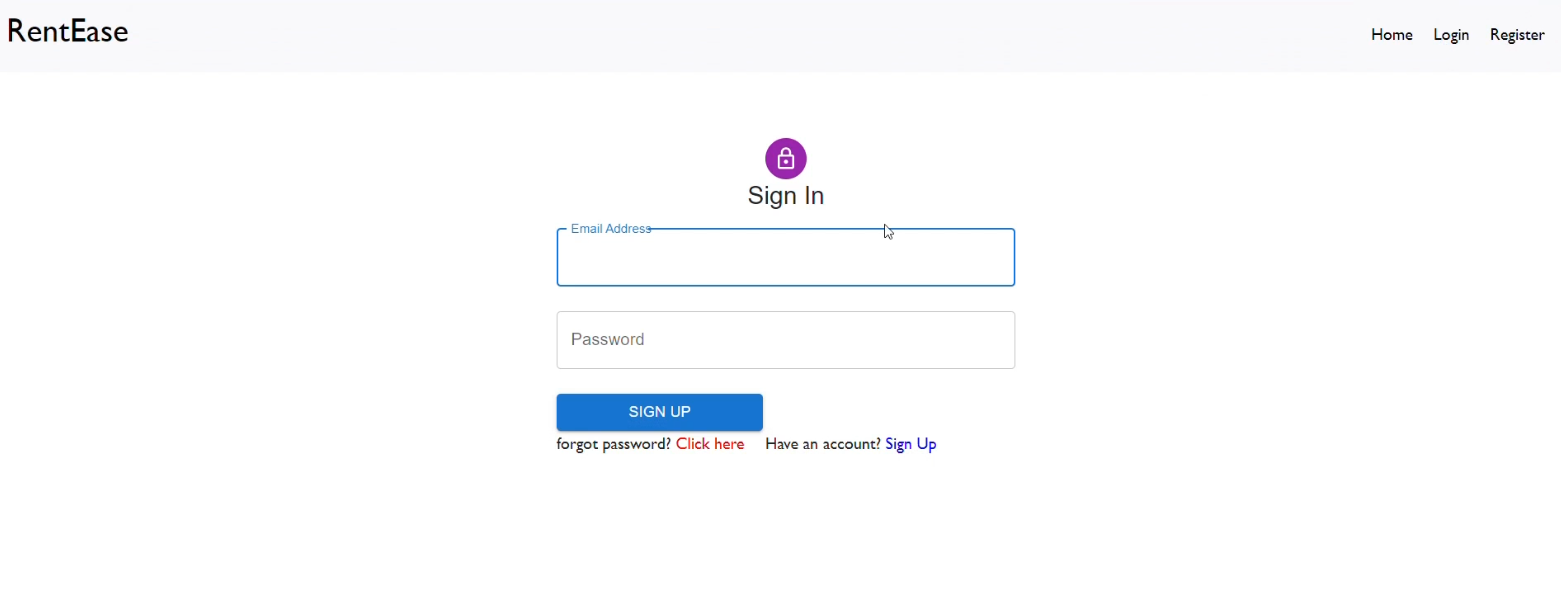
**Project Implementation:**

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one’s provided below.

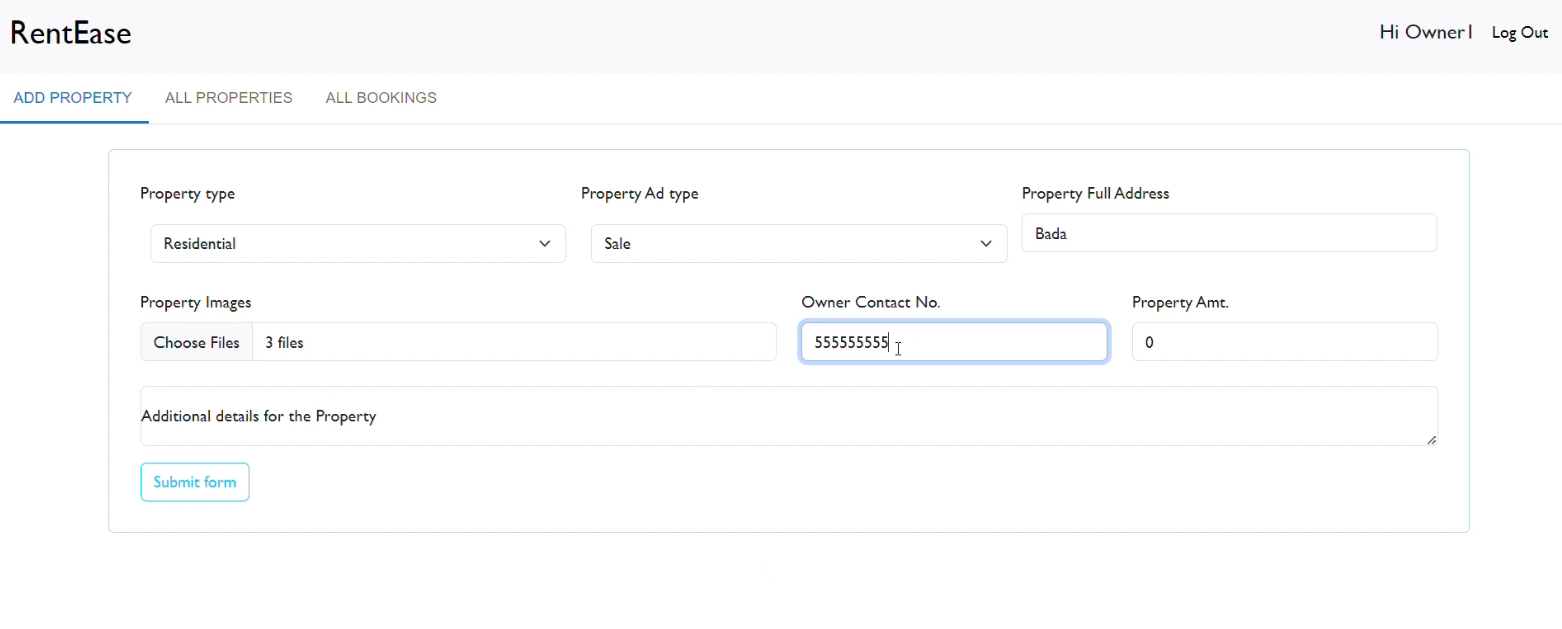
* Landing Page



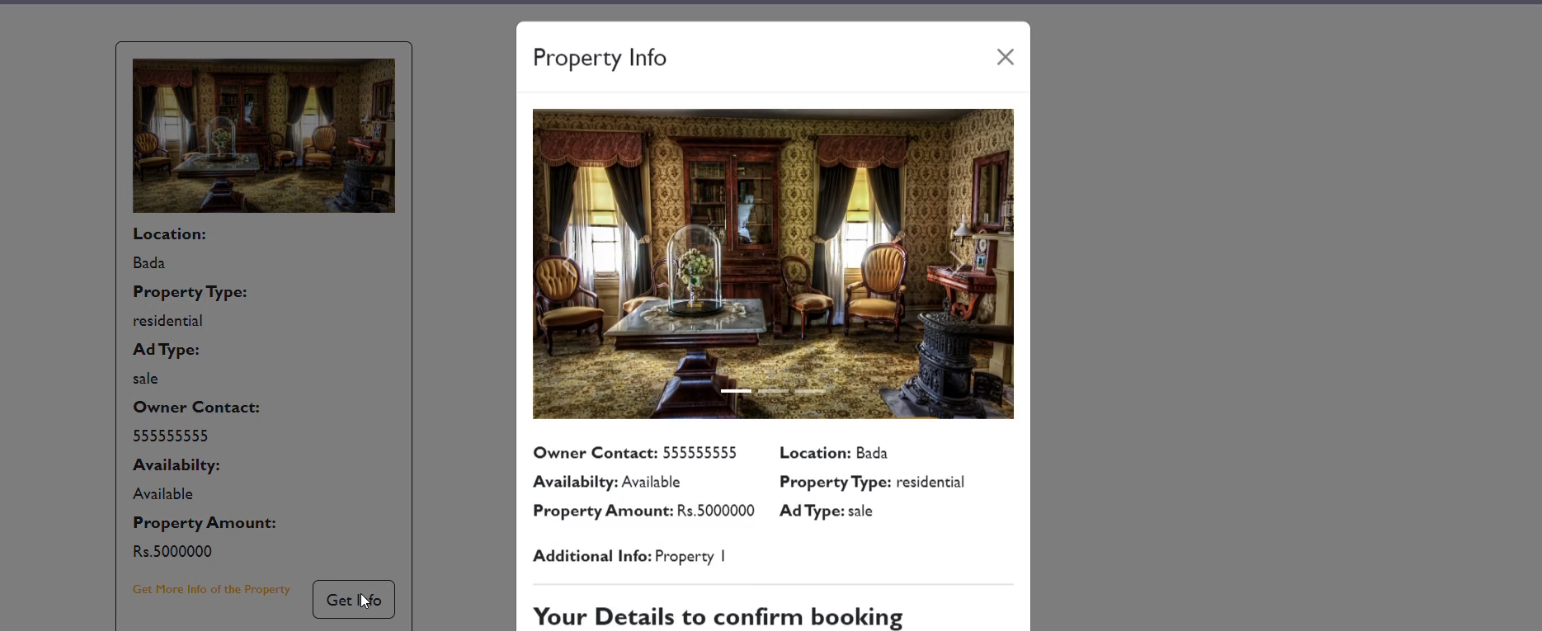
* Login Page



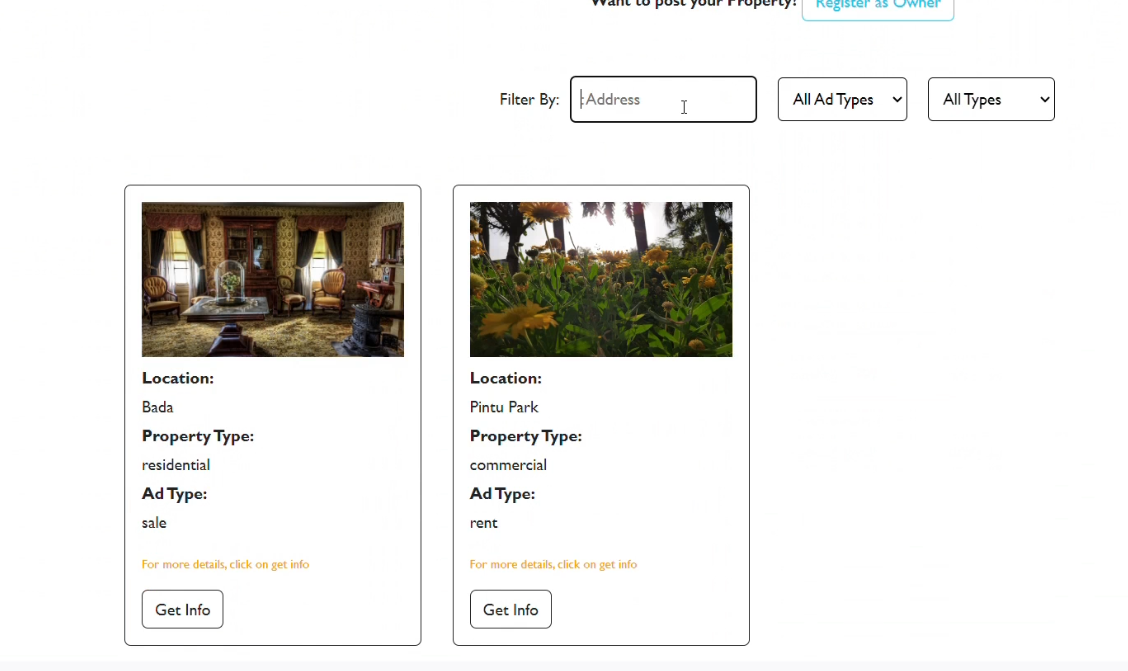
* Registration Page



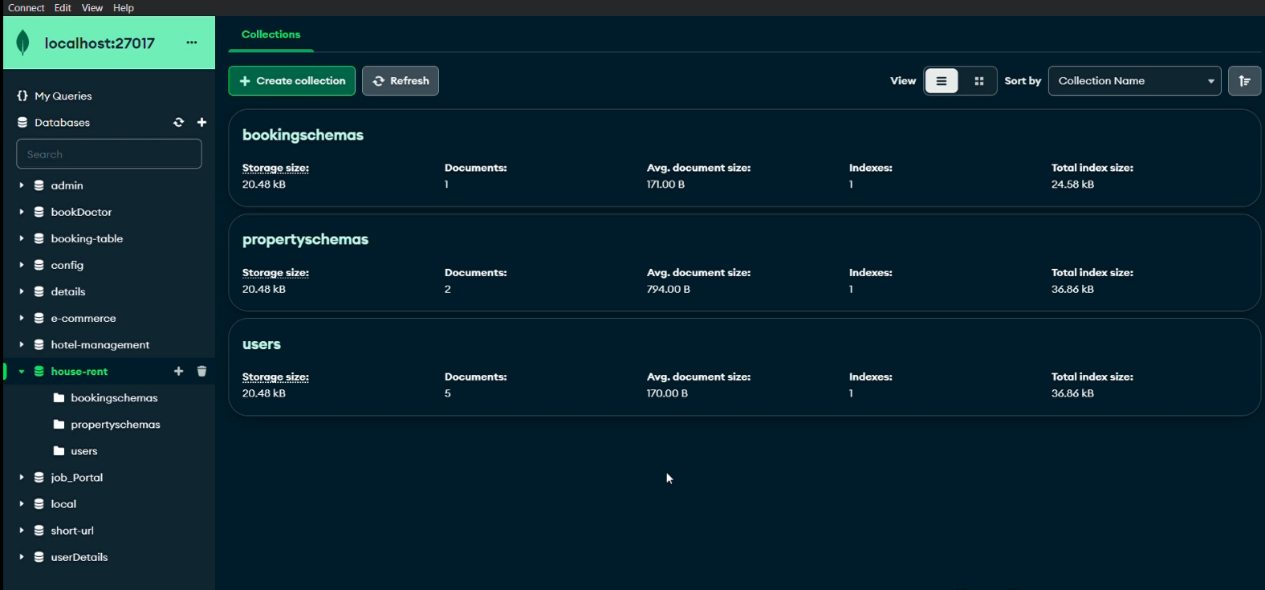
* Common Dashboard For Complaint



* Admin Dashboard



* Agent Dashboard



For any further doubts or help, please consider the code from the google drive,

<https://drive.google.com/drive/folders/118rI9Vp8jkx-xoICSzbVULIQAE9LUzXe>