HOUSE RENT APP USING MERN

1. Introduction

Project Title: HOUSE RENT APP USING MERN

TEAM MEMBERS:

S.NO	NAME	ROLE	RESPONSIBILITIES
1.	**Gowri Rajan.M**	Full-Stack Developer	Responsible for overall development, Including front-end, back-end, server-side logic, and database design
2.	**Govind.R.M**	Frontend Developer	Responsible for UI/UX design using React, Material UI, and Bootstrap.
3.	**Gokulalakshmi.S**	Database Administrator	Responsible for MongoDB setup and ensuring data integrity.
4.	**Surya Durga Rani.G**	Backend Developer	Responsible for Express.js setup and API development.

2. Project Overview

Purpose: The goal of a “House Rent App” using the MERN stack (MongoDB, Express.js, React, and Node.js) is to streamline the rental process, providing a unified, efficient platform for landlords and renters. It enables landlords to list properties, manage inquiries, and handle rent payments, while renters can browse, book, and communicate directly within the app. Secure payment processing, user authentication, and property management tools create a seamless, trustworthy rental experience. Built with scalability in mind, the app leverages MERN to ensure responsiveness and reliability, making renting more accessible, transparent, and manageable for all parties involved.

Features:

1. **User Authentication and Profiles**
    - Secure login and registration for landlords, property managers, and renters with profile management.
2. **Search and Filter Options**
    - Renters can search for properties using filters like location, price range, property type, number of bedrooms, and amenities.
3. **Detailed Property Pages**
    - Each property has a dedicated page with images, detailed descriptions, and information on rental terms.
4. **Booking and Scheduling**
    - Renters can book or schedule viewings of properties, with calendar integration to manage availability.
5. **Secure Payment Processing**
    - Integration of payment gateways to allow rent payments or deposits directly through the app.
6. **Dashboard and Analytics**
    - Landlords have access to a dashboard with analytics on listing views, inquiries, and rental trends.

3. Architecture

Frontend:

- The frontend is built with React.js, providing a component-based architecture that allows for reusable UI elements.
- The React Router handles navigation between pages such as the login page, dashboard, and Booking and Scheduling.
- Material UI and Bootstrap are used for styling, ensuring the app has a modern, responsive Material UI provides ready-to-use components for consistent UI elements like buttons, text fields, and dialogs.
- Axios is used for making HTTP requests from the frontend to the backend, enabling communication with RESTful APIs.

Backend:

- The backend is powered by Node.js and Express.js. Express.js simplifies routing and middleware management.
- The MongoDB database is used to store user and property managers , as well as renters with profile management.
- Mongoose ODM is used to define models and interact with MongoDB, allowing for easy schema-based data validation and manipulation.
- The backend is also responsible for handling authentication via JWT (JSON Web Token) and managing user sessions.

Database:

- The application uses MongoDB, a NoSQL database, to store and manage the data. MongoDB is chosen for its flexibility and scalability.
- Mongoose is used to define the application's schemas and models. The key models are:
- User Model: Stores renters and property managers, including email, hashed password, and role.
- Login Model: Stores secure login and registration for landlords, property managers, and renters with profile management.

4. Setup Instructions

Prerequisites:

- **Node.js:** Make sure Node.js is installed. This is needed to run the application locally and install necessary dependencies.
- **MongoDB:** Ensure that you have a running instance of MongoDB. You can use either a local MongoDB setup or a cloud-based solution like MongoDB Atlas.
- **npm:** The Node Package Manager (npm) is required to install dependencies.

Installation:

1. Clone the Repository:
   https://github.com/gowrirajanm/House-Rent-App--Using-MERN
2. Install Frontend Dependencies: Navigate to the client folder and run:
   cd client
   npm install
3. Install Backend Dependencies: Navigate to the server folder and run:
   cd server
   npm install
4. Set up Environment Variables: In the server folder, create a .env file to store sensitive information such as:
   MONGODB_URI: The connection string for your MongoDB database.
   JWT_SECRET: A secret key used for signing JWT tokens.
   PORT: Port on which the backend server will run (default is 3000).
5. Start the Backend Server: Run the following command in the server directory:
   npm start
6. Start the Frontend Server: Run the following command in the client directory:
   npm start

The application will be available at http://localhost:3000 (frontend) and http://localhost:5000 (backend).

5. Folder Structure

Client:

- **src/:** Contains all React components, hooks, and utilities.
- **components/:** Reusable React components such as Header, LoginForm ,registeration page,etc.
- **App.js:** Main entry point for the React application, managing routing and rendering components.
- **services/:** Contains files for Axios API calls to interact with the backend.

Server:

- **controllers/:** Contains the logic for handling requests, such as registering users, profile of renters, etc.
- **models/:** Defines Mongoose models for the MongoDB database .
- **routes/:** Defines the API endpoints.
- **middleware/:** Contains middleware for handling authentication, token verification, and error handling.
- **config/:** Stores configuration files like the database connection and environment variables.

6. Running the Application

Frontend: To start the frontend server, run the following command in the client folder:

```
npm start
```

Backend: To start the backend server, run the following command in the server folder:

```
npm start
```

7. API Documentation

1. User Authentication and Profile:

Register User:

Endpoint: POST /api/auth/register

- **Request Body:**

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "securepassword",
  "role": "renter"
}
```

- **Response:**

```
{
 "success": true,
 "data": {
  "id": "userId",
  "name": "John Doe",
  "email": "john@example.com",
  "role": "renter"
 }
}
```

2. Property Listings:

Add New Property:

- **Endpoint: POST /api/properties**
- **Request Body:**

```
{
  "title": "2 Bedroom Apartment",
  "description": "Spacious apartment near downtown",
  "price": 1500,
  "location": "City Center",
  "amenities": ["Wi-Fi", "Gym"],
  "images": ["img1.jpg", "img2.jpg"]
}
```

- **Response:**

{
"success": true,
"data": { "propertyId": "123", "title": "2 Bedroom Apartment" }
}

3. Booking and Scheduling:

Book a Property:

- **Endpoint: POST /api/bookings**
- **Request Body:**
  {
  "propertyId": "123",
  "startDate": "2023-06-01",
  "endDate": "2023-06-10"
  }
- **Response:**
  {
  "success": true,
  "data": { "bookingId": "789", "propertyId": "123", "status": "confirmed" }
  }

4. Payment Processing:

Initiate Payment:

- **Endpoint: POST /api/payments/initiate**
- **Request Body:**
  {
  "bookingId": "789",
  "amount": 1500,
  "paymentMethod": "credit card"
  }
- **Response Body:**
  {
  "success": true,
  "data": { "paymentId": "payment123", "status": "initiated" }
  }

8. Authentication

JWT Authentication: In this system, users authenticate by logging in and receiving a **JWT (JSON Web Token).** This token must be included in the authorization header of all future requests to protected routes. The token serves as proof that the user is authenticated and ensures that only authorized users can access specific resources, like booking property or accessing the dashboard.

Authorization Middleware:

- **Authentication Middleware:** This middleware verifies whether a user is logged in by checking the validity of the JWT token. If the token is valid, the user is allowed to proceed; otherwise, access is denied.

- **Authorization Middleware:** Once the user's identity is verified through authentication, this middleware checks if the user has the appropriate role to access a certain route. For example, only admin should be able to access admin-specific routes (e.g., managing their schedule), while renters should be restricted to booking lands and viewing their own data.

9. User Interface

The User Interface (UI) of the House Rent App with MERN application is designed to provide a smooth and intuitive experience for the users. Below are the main UI components:

Login Page:

- Users can log in using their email and password.
- Features a clean design with clear input fields and a login button.
- Responsive design for mobile and desktop screens.

Dashboard (Customer):

- Displays a list of available property with basic information like locality, specialty, and availability.
- Option to book an house by selecting a price, date, and uploading necessary documents.

Booking Form:

- After selecting a house, the user is directed to a form to book the appointment.
- The form includes fields for booking date and document upload.

Property Owner Dashboard:

- admin can view their upcoming bookings.
- Option to confirm or cancel bookings based on user actions.

Admin Dashboard:

- Admins can manage owner applications, monitor all user bookings, and enforce platform policies.

GIFs showing the user flow and interactions (e.g., login, booking process, dashboard navigation) can also be provided.

10. Testing

Testing Strategy:

Unit Testing:

- Used for testing individual components and backend functions.
- Jest and Mocha were used for unit testing backend routes, validation, and middleware.

Integration Testing:

- Focused on ensuring that the communication between the frontend, backend, and database worked as expected.
- Tools like Postman were used to test API endpoints.

End-to-End Testing:

- Simulated the user journey from login to booking an property and managing it.
- Cypress was used for front-end testing, ensuring that all user interactions are properly handled.

Testing Tools:

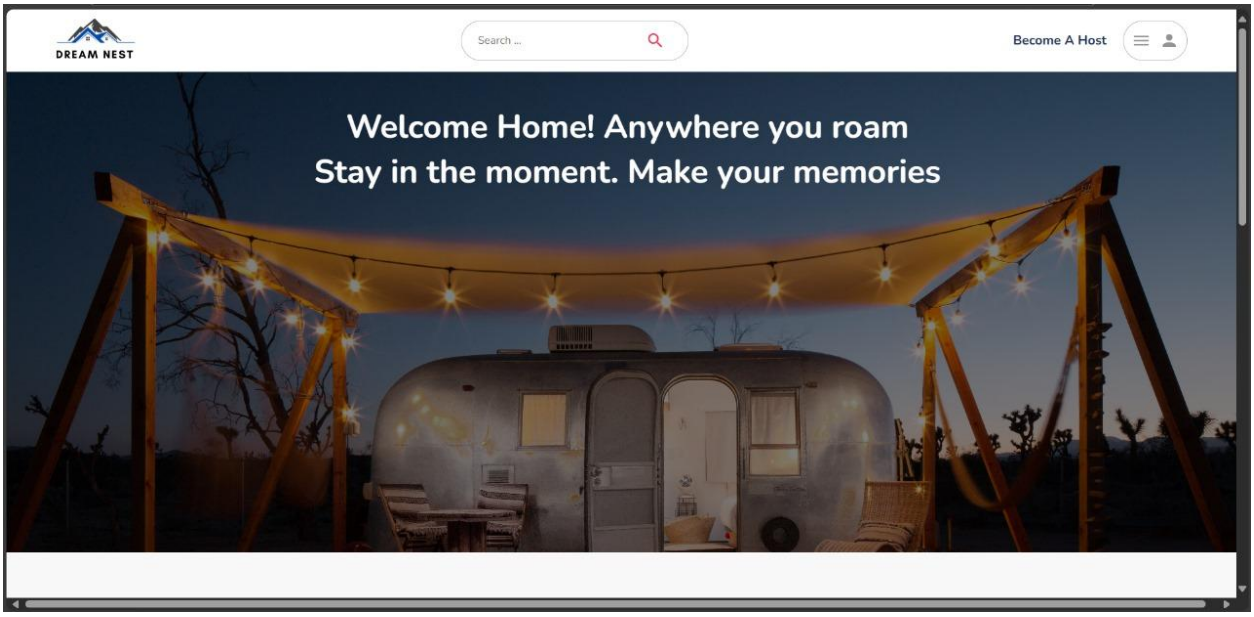
Frontend: Jest, Enzyme, React Testing Library for unit and integration testing of React components.

Backend: Mocha, Chai, Supertest for testing Express.js API routes.

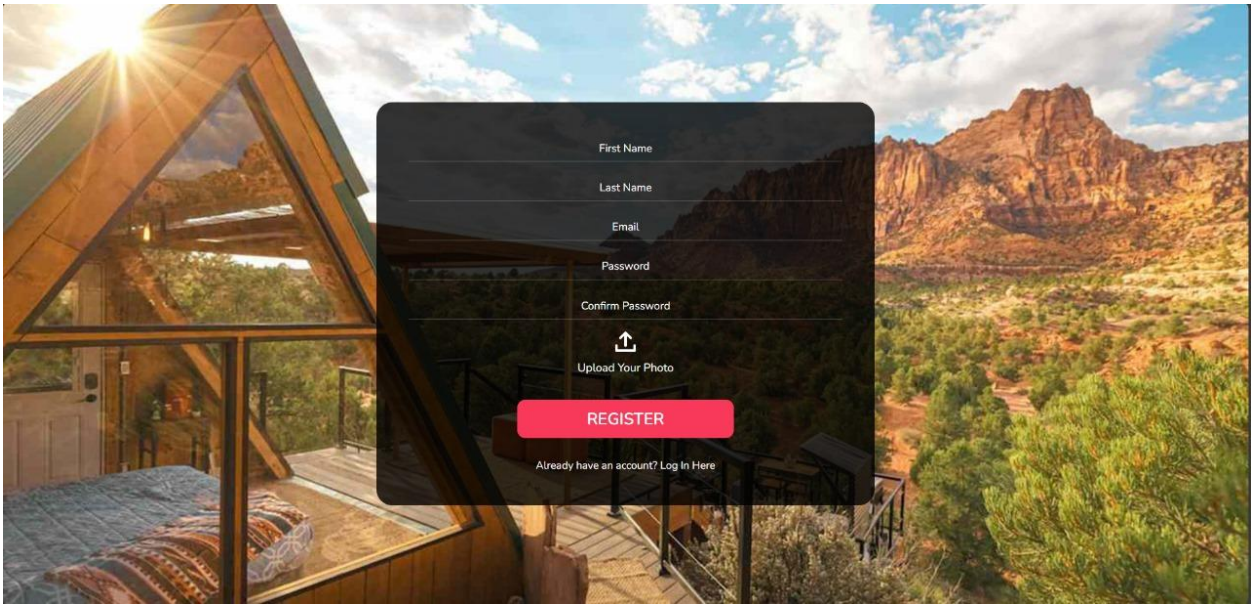
API Testing: Postman for manually testing backend endpoints.

11. Screenshots

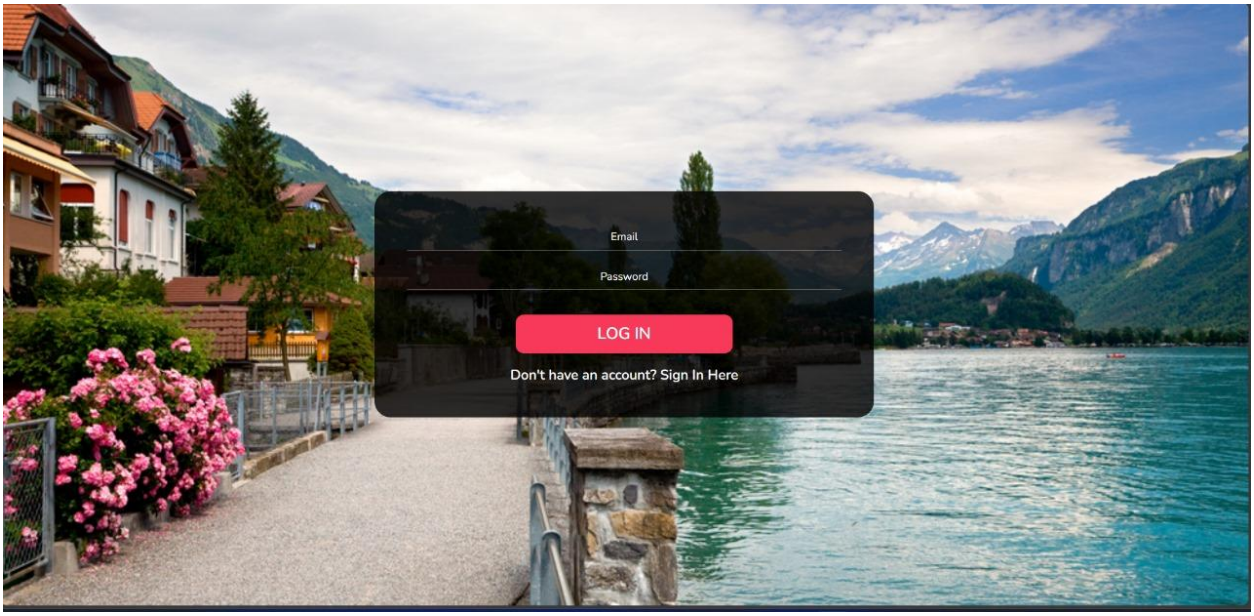
Landing page:



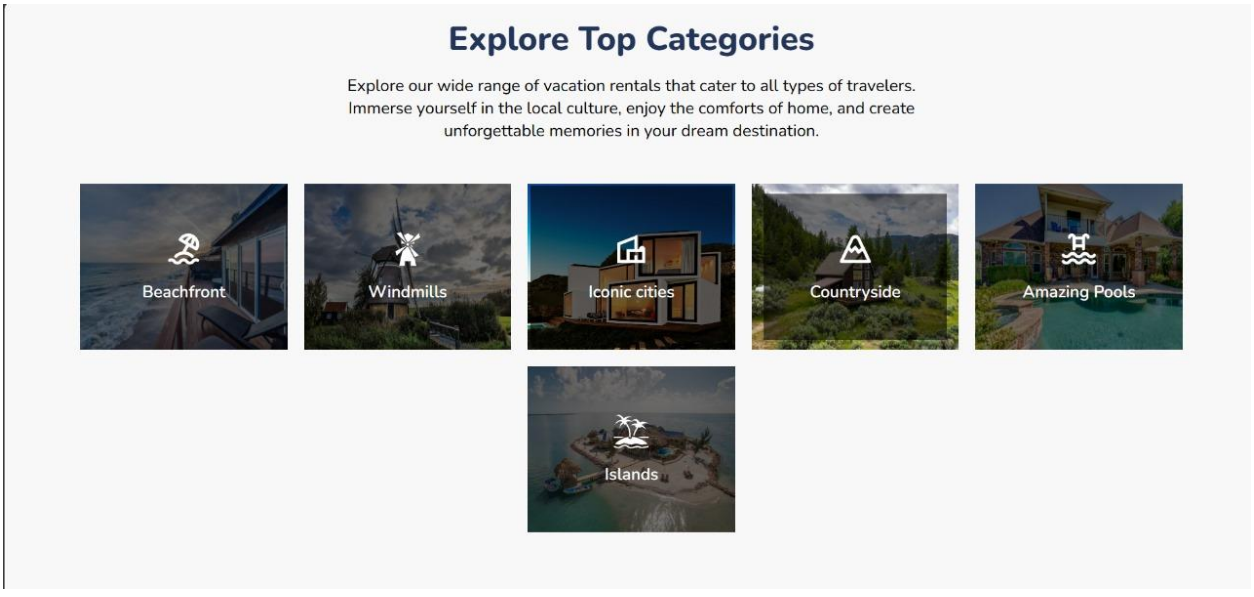
Register page:



Login Page:



Property CatLog's:



12. Known Issues

While most of the application is fully functional, there are a few issues that developers or users should be aware of:

Issue with Booking Confirmation:

- Occasionally, the booking confirmation status may take a few moments to update due to delays in backend processing.

Responsive Layout on Mobile:

- Some UI components may not fully adjust for extremely small screen sizes (e.g., older smartphones).

File Upload Issue:

- The file upload feature for appointment documents may fail intermittently with larger files (greater than 10MB).

Workarounds:

- For booking confirmation delays, refreshing the page can help load the updated status.
- Mobile users can zoom out to view the full content if it's cut off on smaller screens.
- For file uploads, users are encouraged to compress files before uploading.

13. Future Enhancements

There are several potential improvements and features that could be added to the House Rent App with MERN application:

1. Advanced Search & Filters

- Custom Filters: Enable filters for price range, property type, furnished/unfurnished, amenities, pet-friendly options, etc.
- Location-based Search: Use Google Maps or Mapbox to show nearby listings.

2. Real-time Notifications

- Push Notifications: Notify users of new listings, price drops, or status updates on their applications.
- Email and SMS Notifications: Alerts for new messages, booking confirmations, and listing expirations.

3. Booking & Scheduling

- In-app Scheduling: Allow users to book viewing appointments with property owners/agents.
- Automated Reminders: Send reminders for scheduled viewings and upcoming lease renewal dates.

4. Secure Payment Integration

- Rent Payment: Integrate payment gateways like Stripe or PayPal for secure rent payments.
- Deposit Management: Securely handle deposits and refunds through the app.

5. Tenant and Landlord Profiles

- Detailed Profiles: Allow tenants and landlords to create detailed profiles with verification (e.g., ID, background checks).
- Reviews and Ratings: Enable tenants and landlords to review each other, building trust within the platform.

17. AR/VR Integration for Property Viewing

- Augmented Reality: Use AR to let users “walk through” properties virtually.
- VR Tour Integration: Provide a more immersive experience through VR for users