**Full Stack Web Development with MERN**

**Project Documentation**

**1. Introduction**

**Project Title: House Rental App Using MERN Stack**

**Team Members:**

* **S T Anantharaman – Team Leader**
* **Bharath**  J
* **Avinash R**
* **Vinoth Kumar C**

**2. Project Overview**

**Purpose:**

The **House Rent App** is designed to help users search, list, and rent houses or apartments with ease. It allows landlords to list their properties, and renters can browse listings, filter based on various criteria, and contact landlords directly. The app aims to provide an intuitive platform for both renters and landlords to manage and find properties efficiently.

**Features:**

* **User Registration & Authentication**: Users can register, login, and manage their profiles (both renters and landlords).
* **Property Listings**: Landlords can list new properties with detailed information (e.g., price, location, photos).
* **Search & Filter**: Renters can search properties based on location, price, amenities, etc., and view filtered results.
* **Property Details Page**: Detailed pages for each listing, displaying images, descriptions, and contact details.
* **Favourites**: Renters can save properties to their favourites list for easy access.
* **Contact Landlord**: Renters can send messages to landlords for inquiries or schedule viewings.
* **Admin Dashboard**: Admins can manage users, properties, and reviews.

**3. Architecture**

**Frontend:**

* **Technology:** The frontend is built with **React.js**, providing a dynamic user interface. We use **React Router** for client-side routing, enabling smooth transitions between pages without reloading.
* **State Management:** **React Context API** is used to manage global state, including user authentication status, property listings, and favourites.
* **Styling:** **Material-UI** is used for component styling and UI consistency, with **CSS** for custom styles.
* **Responsive Design:** The app is mobile-first, ensuring accessibility and responsiveness on all devices.

**Backend:**

* **Technology:** The backend is built using **Node.js** with **Express.js** for creating RESTful APIs that handle user interactions and property data.
* **Authentication:** We use **JWT (JSON Web Tokens)** for secure user authentication. Tokens are stored in local storage and sent with each request to authenticate users.
* **Middleware:** Custom middleware for validation, authentication, and error handling.
* **Routes & Controllers:** Express routes handle API requests (e.g., GET, POST, PUT, DELETE) for properties, users, and authentication.
* **Security:** We use **Helmet.js** to secure HTTP headers and **CORS** to handle cross-origin requests.

**Database:**

* **Technology:** The database is **MongoDB**, a NoSQL database, and **Mongoose** is used to model the data.
* **Schema Design:**
  + **User Schema:** Stores user details such as name, email, password, role (admin, landlord, renter).
  + **Property Schema:** Stores property details, including title, description, price, location, images, landlord information, and availability.
  + **Favourites Schema:** Tracks which properties are saved by which users.
* **Data Interactions:** The backend performs CRUD operations (Create, Read, Update, Delete) on MongoDB collection.

**4. Setup Instructions**

**Prerequisites:**

Before running the project, ensure that the following software is installed:

* **Node.js** (version 14 or above)
* **MongoDB** (local installation or MongoDB Atlas for cloud DB)
* **Git** (for cloning the repository)

**Installation:**

1. **Clone the Repository:**

bash

https://github.com/anantharaman387/Naan-Mudhalvan-Project.git

cd Naan-Mudhalvan\_Project

1. **Install Backend Dependencies:** Navigate to the server directory and install dependencies:

bash

cd server

npm install

1. **Install Frontend Dependencies:** Navigate to the client directory and install dependencies:

bash

cd client

npm install

1. **Setup Environment Variables:**
   * Create a .env file in the server directory with the following keys:

bash

MONGO\_URI=mongodb://localhost:3000/houserentdb

JWT\_SECRET=your\_jwt\_secret\_key

1. **Run MongoDB (if using local setup):**
   * If MongoDB is installed locally, run the following:

bash

mongod

**5. Folder Structure**

**Client:**

The client folder contains the React frontend.

* **public/**: Static files like index.html and images.
* **src/**: Main application code.
  + **components/**: Reusable UI components (e.g., Navbar, PropertyCard, Footer).
  + **pages/**: Contains page-level components (e.g., Home, PropertyDetails, UserProfile).
  + **context/**: Contains React context files for state management (e.g., AuthContext, PropertyContext).
  + **App.js**: The main component that handles routing and layout.

**Server:**

The server folder contains the backend logic.

* **models/**: Contains Mongoose models for user and property schemas.
* **controllers/**: Logic for handling routes (e.g., user registration, property CRUD).
* **routes/**: API route definitions (e.g., authRoutes.js, propertyRoutes.js).
* **middleware/**: Custom middleware for authentication, validation, and error handling.
* **config/**: Configuration files like database connection and JWT settings.
* **server.js**: The entry point for the backend application.

**6. Running the Application**

**Frontend:**

1. Navigate to the client directory:

bash

cd client

1. Start the React development server:

bash

npm start

The app will run at http://localhost:3000.

**Backend:**

1. Navigate to the server directory:

bash

cd server

1. Start the Node.js server:

bash

npm start

The backend API will run at http://localhost:5000.

**7. API Documentation**

**Endpoint: GET /api/properties**

* **Description:** Fetch all property listings.
* **Request:**
  + Method: GET
  + URL: /api/properties
* **Response:**

json

[

{

"id": "1",

"title": “Residential",

"price": 40000,

"location": "Chennai",

"description": "Good Looking Property",

"images": ["image1.jpg"],

"landlord": {

"name": "Ananth",

"contact": "ananth@example.com"

}

}

]

**Endpoint: POST /api/users/register**

* **Description:** Register a new user.
* **Request:**

json

{

"username": "deepak",

"email": "deepak123@example.com",

"password": "password123",

"role": "renter"

}

* **Response:**

json

{

"message": "User registered successfully"

}

**8. Authentication**

**Authentication:**

* **JWT Authentication** is used for securing API endpoints.
  + When a user logs in, a **JWT token** is generated and returned.
  + The token is stored in the client's **local Storage** and sent with each subsequent request in the **Authorization** header.

**Example:**

* **Login Endpoint:** POST /api/auth/login
* **Request:**

json

{

"email": "deepak123@example.com",

"password": "password123"

}

* **Response:**

json

{

"token": "your\_jwt\_token"

}

**9. User Interface**

**Screenshots or Demo:**

* Include screenshots or links to the demo of the application. Below are some examples:
  + **Home Page:** Screenshot of the property listings and search filters.
  + **Property Details:** Screenshot of a detailed property page with images and contact options.
  + **Login/Signup:** Screenshots of the user authentication forms.

**10. Testing**

**Testing Strategy:**

* **Unit Testing:** **Jest** is used to test individual components and logic.
* **API Testing:** **Supertest** is used to test API endpoints for correct functionality.
* **E2E Testing:** **Cypress** is used for end-to-end tests to simulate user interactions with the application.

**Example Test:**

* **Test:** Verify property listing retrieval

javascript

Copy code

test('should fetch property listings', async () => {

const response = await request(app).get('/api/properties');

expect(response.status).toBe(200);

expect(response.body).toBeInstanceOf(Array);

});