Full Stack Development with MERN
Project Documentation

1.INTRODUCTION:

- **Project Title:** HOUSE RENT APP USING MERN
- **Team Members:** Anandhamahalakshmi, Sasikala, Vigneshwaran V, Vignesh V

2.PROJECT OVERVIEW:

- **Purpose :** The House Rent Application is designed to simplify the process of renting homes by providing an intuitive, user-friendly platform for renters and landlords. It aims to connect renters with available properties, allowing landlords to list and manage their rental units efficiently. The application is built using the MERN stack (MongoDB, Express, React, Node.js) for a scalable and fast solution. The core objective is to streamline property searches, enhance communication, and improve the rental experience for both renters and property owners.

- **Features :**

  User Authentication: Renters and landlords can securely register, log in, and manage their profiles with JWT-based authentication.

  Property Listings: Landlords can add, update, and remove property listings with details like price, location, photos, and amenities.

  Search & Filters: Renters can search for properties based on criteria like location, price, and size, with real-time filtering options.

  Property Details: Renters can view detailed information about properties, including images and landlord contact details.

  Responsive Design: The app is built with React, ensuring a mobile-friendly and responsive user interface.

Admin Panel: Admins can manage user accounts and moderate property listings.

Real-time Communication: Renters and landlords can communicate via messaging or contact forms for inquiries and negotiations.

Property Image Uploads: Landlords can upload multiple property images, enhancing listings.

Scalable and Secure: Built with the MERN stack for high performance, security, and future scalability.

3. ARCHITECTURE :

Frontend (React):

- Component-based UI with React, using **React Router** for navigation.
- **State Management** with Context or Redux.
- **Axios** for API communication.
- UI components from libraries like

Material-UI. **Backend (Node.js & Express.js)**:

- **Node.js** as runtime, **Express.js** for API routing.
- **REST API** with routes for user auth (JWT), house listings, and user profiles.
- Middleware for **authentication** and **error handling**.

Database (MongoDB):

- **Mongoose** for schema management.
- Collections for **Users**, **Houses**, and **Rentals**.
- Operations: CRUD for houses, user registration/login, rental agreements.
- Indexing for better search performance (location, price, availability).

4. Setup Instructions:

Prerequisites

- Node.js (v14 or higher) - For running the backend.

- MongoDB - Local or MongoDB Atlas for the database.
- npm or yarn - For managing dependencies.
- Git - For version control and cloning the repository.

Installation Steps

1. **Clone the repository**:

```
git clone https://github.com/your-repo/house-rent-app.git

cd house-rent-app
```

2. **Install Backend Dependencies:**

- Navigate to the backend folder (if separated):
  `cd backend`
- Install required

  dependencies: `npm install`

3. **Install Frontend Dependencies:**

- Navigate to the frontend
  folder: `cd ../frontend`
- Install required

  dependencies: `npm install`

4. **Set up Environment Variables:**

- Create a `.env` file in the root of the **backend** directory with the following values:

```
MONGO_URI=your-mongodb-uri

JWT_SECRET=your-secret-key

PORT=5000
```

- For **frontend,** you may also need a `.env` file:

```
REACT_APP_API_URL=http://localhost:5000/api
```

5. **Run the app:**

- Start the

  Backend: cd

  backend

  npm start

- Start the

  Frontend: cd

  frontend

  npm start

6. **Test the Application:**

- Open your browser and go to http://localhost:3000 for the frontend and http://localhost:5000 for the backend API.

5. Folder Structure:

1. Client: React Frontend Structure:

- The React frontend of the house rent app is typically organized into the following folder structure:

```
/src
  /components          # Reusable UI components (e.g., Navbar, Footer, Listings)
  /pages               # Components representing different pages (e.g., HomePage, ListingDetails, LoginPage)
  /context             # Global state management using Context API or hooks (e.g., UserContext)
  /services            # API calls and interactions with the backend (e.g., api.js)
```

```
  /assets              # Static assets like images, icons,
and styles

  /styles              # Global CSS, SASS, or
styled-components (e.g., global.css)

  /utils               # Utility functions (e.g., date
formatting, validation)

  App.js               # Main app component

  index.js             # Entry point of the application
```

2. Server: Node.js Backend Structure:

```
  /server

  /controllers         # Logic for handling requests (e.g.,
userController.js, listingController.js)

  /models              # Database schema definitions (e.g.,
User.js, Listing.js, RentRequest.js)

  /routes              # Express routes for different
endpoints (e.g., userRoutes.js, listingRoutes.js)

  /middleware          # Authentication and validation
middleware (e.g., authMiddleware.js)

  /config              # Configuration files (e.g., dbConfig.js,
environment variables)

  /utils               # Utility functions (e.g.,
emailSender.js, fileUploader.js)

  server.js            # Entry point for the server and
express setup

  .env                 # Environment variables (e.g.,
database URI, JWT secret)
```

6. Running the Application:

1. Frontend (React App):

The frontend is built with React, and you'll typically run it with the following steps:

1. **Navigate to the frontend directory (client):** Open your terminal and navigate to the client directory where the React application is located:

   ```
   cd client
   ```

2. **Install dependencies (if you haven't already):** If it's your first time running the project or you've cloned it recently, you need to install the required npm packages:

   ```
   npm install
   ```

3. **Start the React development server:** Start the React app using the following command:

   ```
   npm start
   ```

This will run the React development server on http://localhost:3000 (or another port if specified). The frontend will automatically open in your default web browser.

2. Backend (Node.js Server):

The backend API is built with Node.js and typically uses Express to serve API endpoints. To start it:

1. **Navigate to the backend directory (server):** Open a new terminal window or tab and navigate to the server directory:

   ```
   cd server
   ```

2. **Install dependencies (if you haven't already):** Install the necessary npm packages for the Node.js backend:

   ```
   npm install
   ```

3. **Start the Node.js backend server:** Start the backend server with the following command:

```
npm start
```

The backend server will start running on http://localhost:5000 (or the port defined in your backend configuration). This will handle all API requests (e.g., user authentication, property listings, etc.).

7. API Documentation:

1. User Authentication

- Endpoint: /api/auth/register
- Method: POST
- Request Body:

```
{
  "username": "user123",
  "email": "user@example.com",
  "password": "securepassword"
}
```

- Response:

```
{
  "message": "Registration successful",
  "user": {
    "_id": "user-id",
    "username": "user123",
    "email": "user@example.com"
  }
}
```

- Endpoint: /api/auth/login
- Method: POST
- Request Body:

```
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

- Response:

```
{
  "token": "jwt-token-here",
  "user": {
    "_id": "user-id",
    "username": "user123",
    "email": "user@example.com"
  }
}
```

2. Property Listings:

- Endpoint: /api/properties
- Method: GET
- Query Parameters:
  - location (optional): Filter by city or area
  - minPrice (optional): Minimum price
  - maxPrice (optional): Maximum price
  - bedrooms (optional): Number of bedrooms
- **Response:**

```
[
  {
    "_id": "property-id-1",
```

```
    "title": "3-Bedroom Apartment in City Center",
    "location": "Downtown",
    "price": 1500,
    "bedrooms": 3,
    "description": "Spacious apartment with modern amenities",
    "owner": "user-id"
  },
  {
    "_id": "property-id-2",
    "title": "2-Bedroom House near Park",
    "location": "Suburb",
    "price": 1200,
    "bedrooms": 2,
    "description": "Cozy house with garden space",
    "owner": "user-id"
  }
]
```


3. Create Property Listing

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


```
{
  "title": "2-Bedroom Apartment in Greenfield",
```

```
  "location": "Greenfield",
  "price": 1000,
  "bedrooms": 2,
  "description": "Newly renovated apartment with modern kitchen",
  "owner": "user-id"
}
```


- Response:


```
    {
  "_id": "new-property-id",
  "title": "2-Bedroom Apartment in Greenfield",
  "location": "Greenfield",
  "price": 1000,
  "bedrooms": 2,
  "description": "Newly renovated apartment with modern kitchen",
  "owner": "user-id"
}
```


4. Get Single Property:

- Endpoint: /api/properties/:id
- Method: GET
- Response:


```
    {
  "_id": "property-id",
```

```
  "title": "3-Bedroom Apartment in City Center",
  "location": "Downtown",
  "price": 1500,
  "bedrooms": 3,
  "description": "Spacious apartment with modern amenities",
  "owner": "user-id"
}
```


5. Update Property Listing

- Endpoint: /api/properties/:id
- Method: PUT
- Request Body:


```
{
  "price": 1600,
  "description": "Updated description for the apartment"
}
```


- Response:


```
{
  "_id": "property-id",
  "title": "3-Bedroom Apartment in City Center",
  "location": "Downtown",
  "price": 1600,
  "bedrooms": 3,
```

```
  "description": "Updated description for the
apartment",
  "owner": "user-id"
}
```

6. Delete Property Listing

- Endpoint: /api/properties/:id
- Method: DELETE
- Response

```
{
 "message": "Property listing deleted successfully"
}
```

7. Add Property Review

- Endpoint: /api/properties/:id/reviews
- Method: POST
- Request Body

```
{
  "rating": 5,
  "comment": "Great property! Very spacious and well
maintained."
}
```

- Response:

```
{
  "review": {
    "user": "user-id",
    "rating": 5,
```

```
    "comment": "Great property! Very spacious and well
  maintained.",
    "date": "2024-11-16"
  }
}
```


8. Get Property Reviews

- Endpoint: /api/properties/:id/reviews
- Method: GET
- Response:


```
[
  {
    "user": "user-id",
    "rating": 5,
    "comment": "Great property! Very spacious and well
  maintained.",
    "date": "2024-11-16"
  },
  {
    "user": "user-id-2",
    "rating": 4,
    "comment": "Good place, but the parking is
  limited.",
    "date": "2024-11-15"
  }
]
```

9. User Profile

- Endpoint: /api/users/:id
- Method: GET
- Response:

```
{
  "_id": "user-id",
  "username": "user123",
  "email": "user@example.com",
  "properties": ["property-id-1", "property-id-2"]
}
```

10. Update User Profile

- Endpoint: /api/users/:id
- Method: PUT
- Request Body:

```
{
  "username": "newuser123",
  "email": "newuser@example.com"
}
```

- Response:

```
{
  "_id": "user-id",
  "username": "newuser123",
  "email": "newuser@example.com"
}
```

8. Authentication:

1. Authentication Process

- **JWT (JSON Web Token)** is used for stateless authentication in this app.
- Upon **user registration** or **login**, the server generates a JWT token that is sent to the client.
- The token contains user-specific information (such as user ID) and an expiration time. This allows the client to authenticate subsequent requests.

2. Login Flow

- **User submits login credentials (email and password).**
- The server **validates the credentials** (checks email and hashed password in the database).
- If valid, the server generates a **JWT token** and sends it back to the client.
- The client stores this token (typically in **localStorage** or **sessionStorage**).
- The client includes the token in the Authorization header of future requests (e.g., Authorization: Bearer <token>).

3. Token Storage and Use

- **Client-Side:**
  - The token is stored in **localStorage** or **sessionStorage** on the frontend, ensuring that it persists across page reloads and sessions.
- **Server-Side:**
  - On receiving an API request with a token, the server **verifies the token** using a secret key.
  - If the token is valid, the user is authenticated, and the requested operation is allowed.
  - If the token is expired or invalid, the server responds with an **HTTP 401 Unauthorized** status.

4. Authorization

- **Role-based Access Control (RBAC)** can be implemented to restrict access to certain routes based on the user's role (e.g., regular user, admin, property owner).
- For example, only **authenticated users** can create or update property listings, while **admins** can manage users and all properties.
- Authorization middleware checks the role of the user associated with the provided token before accessing certain routes.

5. **Token Expiry and Refresh**

- JWT tokens have a **short expiry time** (e.g., 1 hour). When the token expires, the user needs to log in again.
- A **refresh token** can be used to obtain a new JWT token without requiring the user to re-enter credentials. This refresh token is typically stored securely (e.g., in an HTTP-only cookie).

6. **Logout**

- When the user logs out, the frontend clears the **JWT token** from storage.
- The server doesn't maintain session state, so logging out is handled entirely client-side.

9. User Interface:

Overview of Key UI Features

1. **Navigation Bar**

The navigation bar provides quick access to the main features of the app. It includes:

- **Logo or App Name: Positioned on the left for branding.**
- **Navigation Links: Links to pages like Home, Browse Rentals, My Bookings, Contact Us, and Login/Sign Up.**
- **Search Bar: Allows users to search for properties by keywords.**
- **Mobile-Friendly Design: Includes a collapsible menu for smaller screens.**

2. **Home Page**

The landing page welcomes users with a visually appealing

layout:

- **Hero Section: A banner with a tagline and a central search bar for exploring properties.**
- **Property Categories: Cards or tiles showcasing categories like apartments, villas, or shared accommodations.**
- **Featured Listings: Highlighted properties displayed with thumbnails, pricing, and a "View Details" button.**

3. **Property Listings Page**

A dedicated page for browsing rental properties with advanced filtering options:

- **Filters: Sidebar or dropdown filters for refining results by location, price range, property type, and other criteria.**
- **Grid View: Displays properties in a uniform, responsive grid with images, titles, descriptions, and pricing.**
- **Pagination: Enables smooth navigation through multiple property pages.**

4. **Property Details Page**

This page provides comprehensive details about a selected property:

- **Image Carousel: Allows users to view multiple images of the property.**
- **Details Section: Displays the property's description, amenities, location, and pricing.**
- **Action Buttons: Includes options like "Contact Landlord" or "Book Now."**

5. **User Profile Page**

A personalized page where users can manage their information and bookings:

- **Profile Section: Displays user details like name, email, and profile picture.**
- **Booking History: A list of properties the user has booked, along with the option to cancel bookings.**
- **Edit Profile: A button or form for updating personal information.**

6. **Contact Us Page**

A simple page allowing users to reach out for support or inquiries:

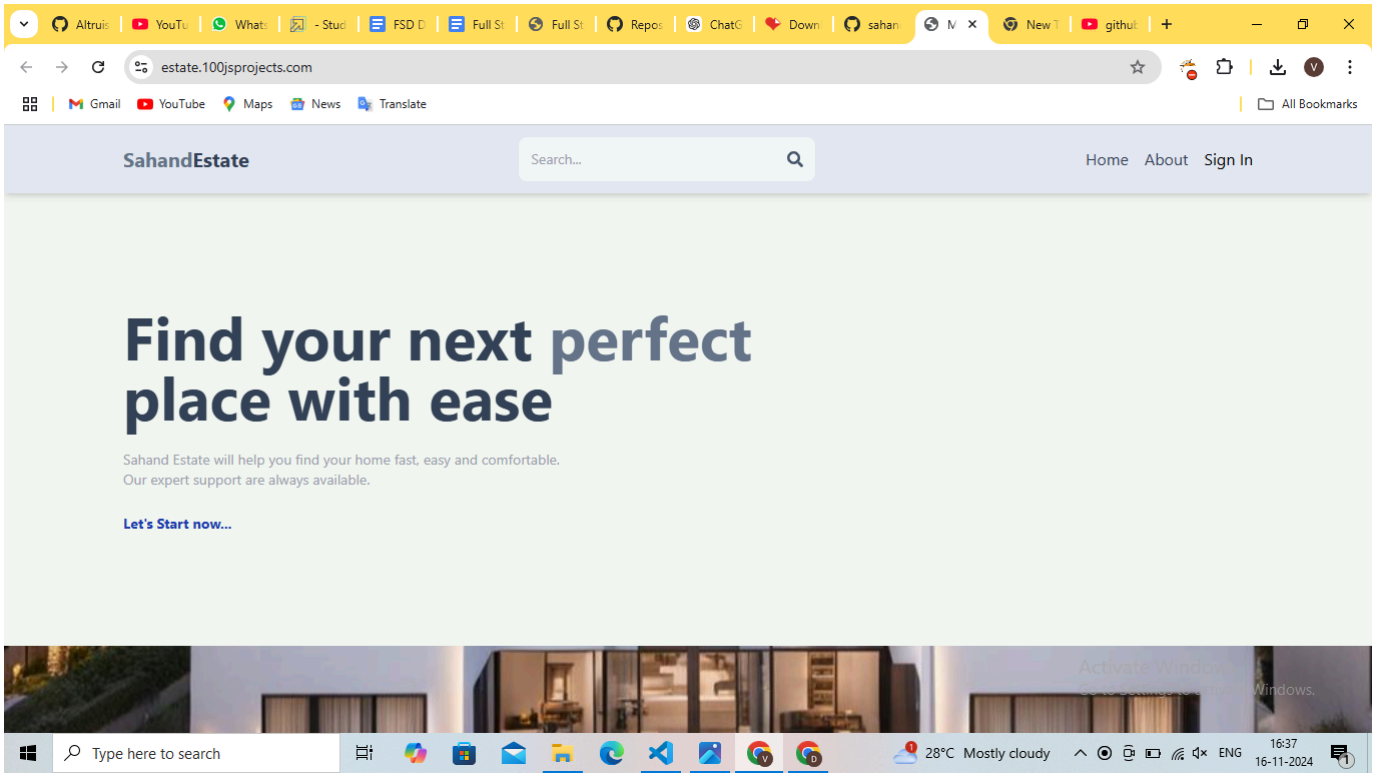
- **Contact Form: Includes fields for name, email, subject, and message.**
- **Additional Information: Provides phone numbers, email addresses, or physical office locations.**

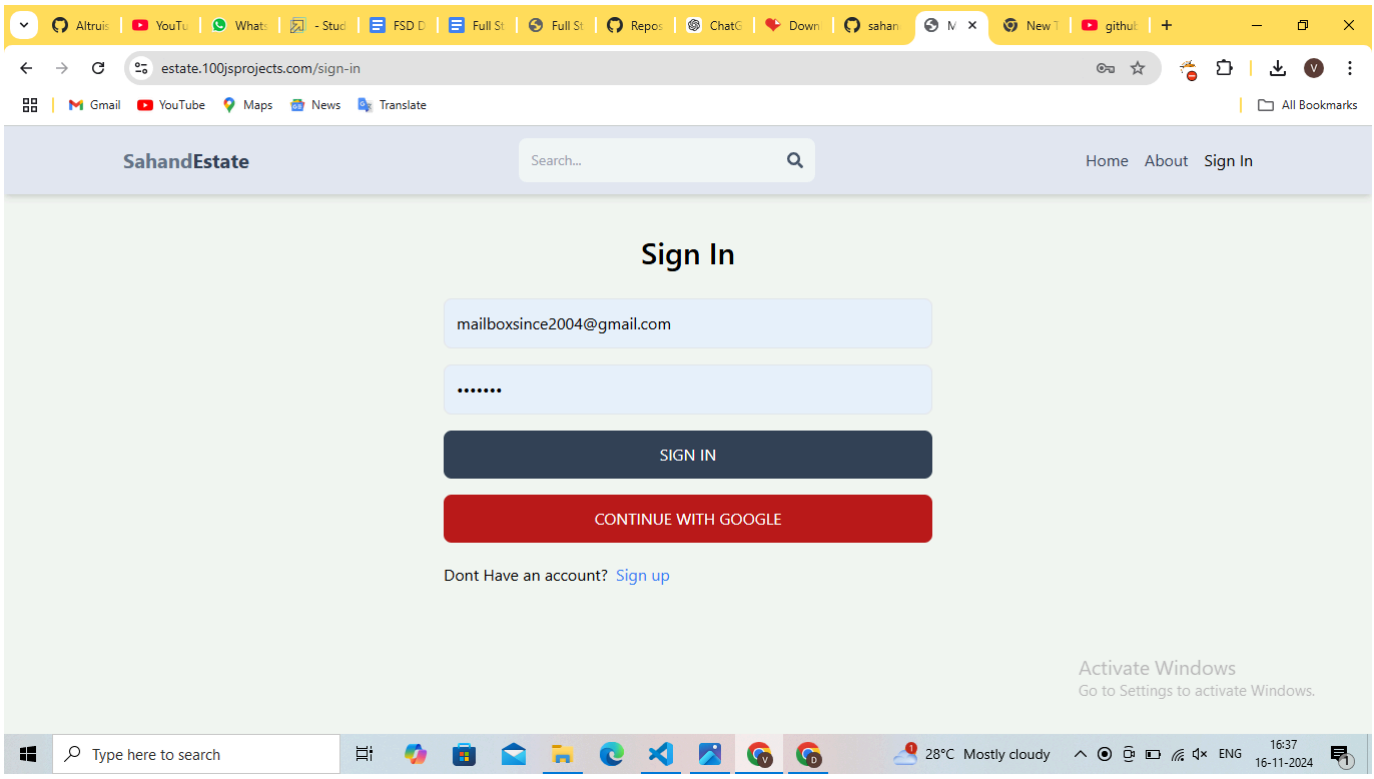
Design Principles

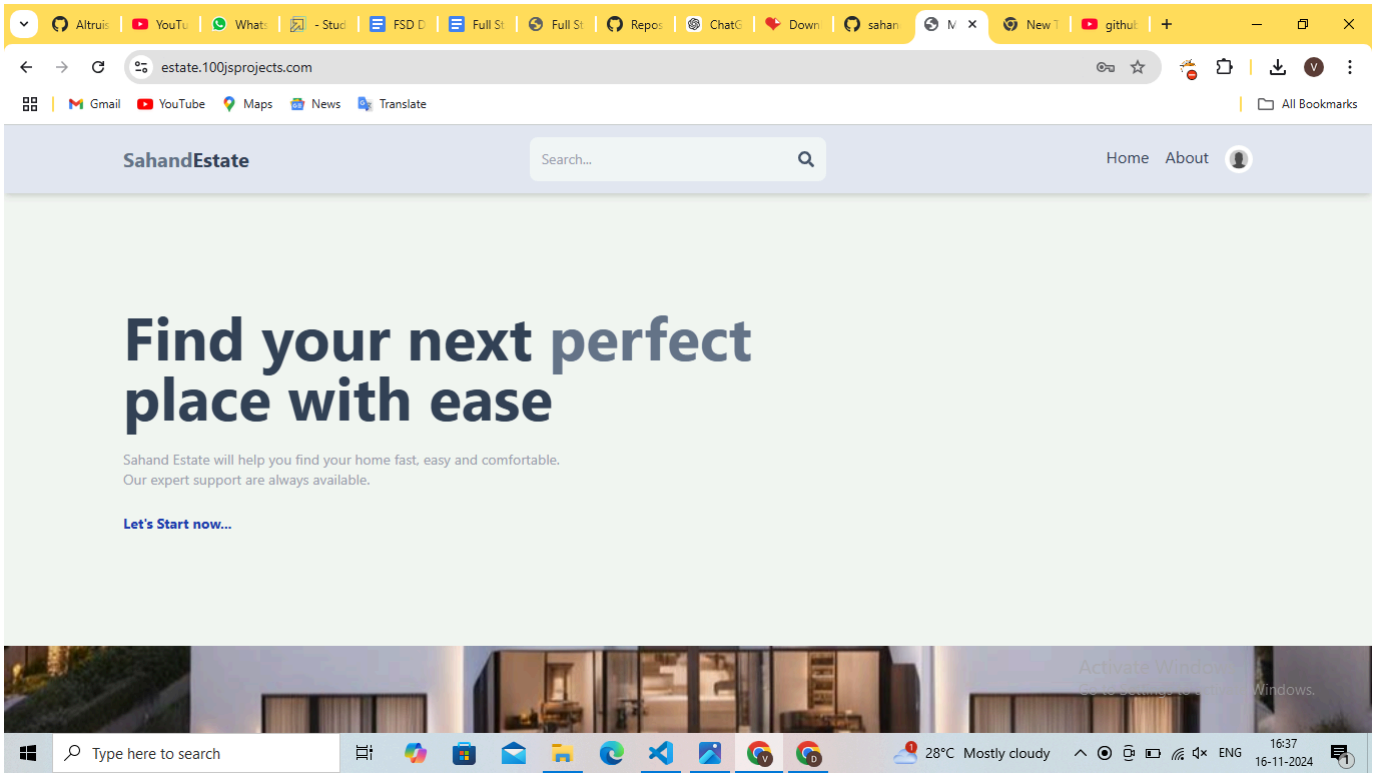
1. **Responsive Design: Ensures optimal display and usability across all screen sizes.**
2. **Consistency: Maintains a cohesive look and feel with reusable components like buttons and cards.**
3. **Ease of Use: Focuses on intuitive navigation and clear call-to-action buttons.**
4. **Aesthetic Appeal: Uses Bootstrap's predefined styles and customization for a modern, professional appearance.**

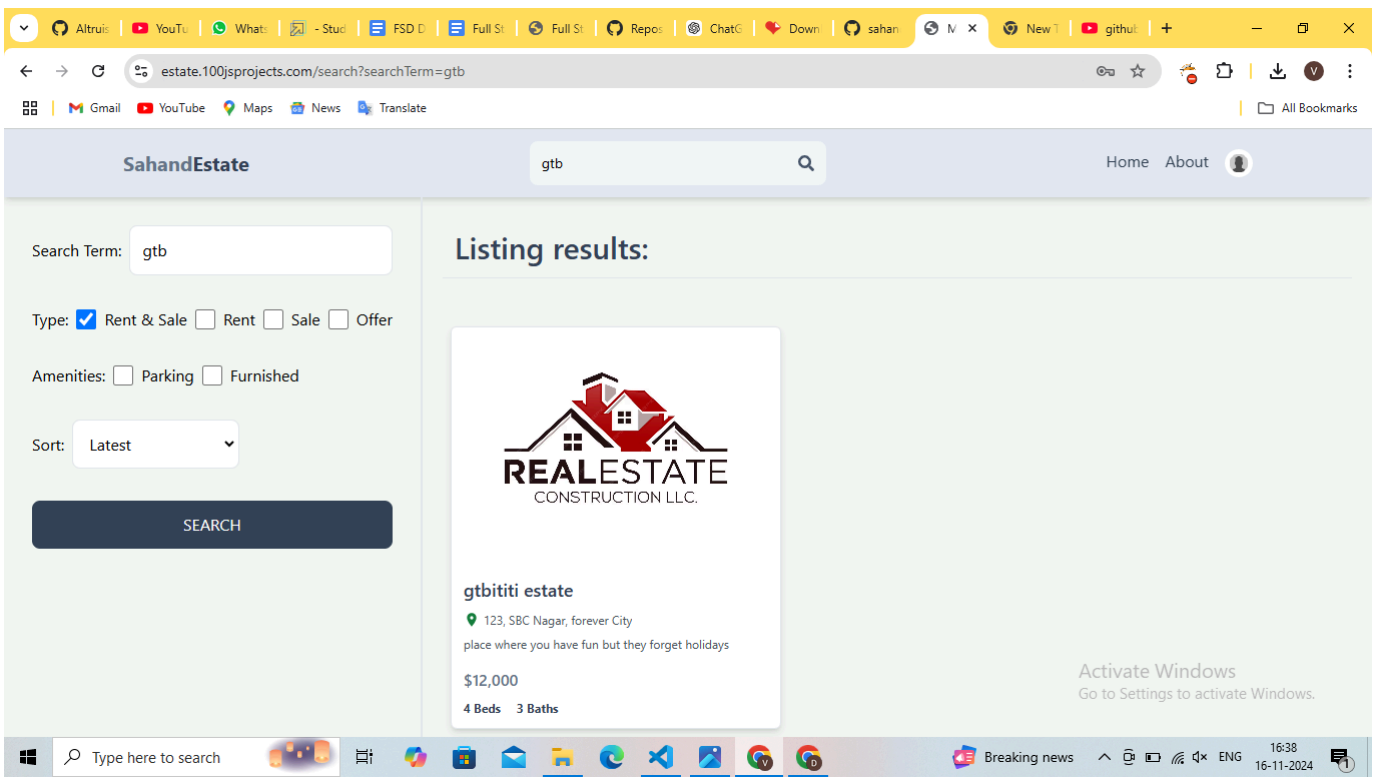
Conclusion

The UI for the house rent app combines aesthetics and functionality to provide a seamless experience for users. It is designed with Bootstrap to ensure responsiveness and scalability, making it ideal for users searching for rental properties efficiently.


estate.100jsprojects.com
SahandEstate
Search...
Home
About
Sign In
Find your next perfect place with ease
Sahand Estate will help you find your home fast, easy and comfortable.
Our expert support are always available.
Let's Start now...
Type here to search
28°C Mostly cloudy
ENG
16:37
16-11-2024


estate.100jsprojects.com/sign-in
SahandEstate
Search...
Home
About
Sign In
Sign In
mailboxsince2004@gmail.com
SIGN IN
CONTINUE WITH GOOGLE
Dont Have an account? Sign up
Activate Windows
Go to Settings to activate Windows.
Type here to search
28°C Mostly cloudy
ENG
16:37
16-11-2024


estate.100jsprojects.com
SahandEstate
Search...
Home
About
Find your next perfect place with ease
Sahand Estate will help you find your home fast, easy and comfortable.
Our expert support are always available.
Let's Start now...
Type here to search
28°C Mostly cloudy
ENG
16:37
16-11-2024


estate.100jsprojects.com/search?searchTerm=gtb
SahandEstate
gtb
Home
About
Search Term: gtb
Type: Rent & Sale Rent Sale Offer
Amenities: Parking Furnished
Sort: Latest
SEARCH
Listing results:
REALESTATE
CONSTRUCTION LLC.
gtbititi estate
123, SBC Nagar, forever City
place where you have fun but they forget holidays
$12,000
4 Beds 3 Baths
Activate Windows
Go to Settings to activate Windows.
Type here to search
Breaking news
ENG
16:38
16-11-2024

10.Testing:

Testing Overview

The testing process involves:

1. Unit Testing: Verifying individual components or functions of the application.
2. Integration Testing: Ensuring that components interact correctly.
3. End-to-End (E2E) Testing: Testing the app's workflow from start to finish.
4. Usability Testing: Evaluating the user experience.
5. Performance Testing: Checking app responsiveness and scalability under various conditions.

Testing Scenarios

1. Navigation Bar

- Objective: Verify the functionality of the navigation bar across all pages.
- Test Cases:
  - Ensure all navigation links direct to the correct pages.
  - Verify the search bar functionality returns relevant results.
  - Test the mobile-responsive toggle on different screen sizes.

2. Home Page

- Objective: Ensure the home page displays correctly and is functional.
- Test Cases:
  - Confirm the hero section loads with the tagline and search bar.
  - Check that property categories display relevant images and labels.
  - Test featured property listings for accuracy and correct linking to details pages.

3. Property Listings Page

- Objective: Validate the property browsing and filtering features.

- **Test Cases:**
  - **Ensure the filter options apply correctly and update the listings dynamically.**
  - **Verify that property cards display accurate information (e.g., title, price, and description).**
  - **Test pagination for smooth navigation across multiple pages of results.**

4. Property Details Page

- **Objective: Test the display of detailed property information.**
- **Test Cases:**
  - **Confirm the image carousel works and displays all property images.**
  - **Verify the property description, pricing, and amenities are accurate.**
  - **Check the functionality of action buttons (e.g., "Contact Landlord" or "Book Now").**

5. User Profile Page

- **Objective: Ensure users can view and manage their profiles and bookings.**
- **Test Cases:**
  - **Verify the display of user details and profile picture.**
  - **Check that the booking history lists correct property details.**
  - **Test the functionality of the "Edit Profile" button and cancellation of bookings.**

6. Contact Us Page

- **Objective: Validate the communication process for user inquiries.**
- **Test Cases:**
  - **Ensure the contact form accepts and submits valid input.**
  - **Test the error handling for incomplete or invalid form submissions.**
  - **Confirm that additional contact information (email, phone) is displayed correctly.**

Testing Types

1. **Unit Testing:**
    - **Verify React components render correctly and perform as expected.**
    - **Test API endpoints using tools like Postman to ensure proper data handling.**
2. **Integration Testing:**
    - **Validate interactions between the frontend (React) and backend (Node.js and Express).**
    - **Test database interactions with MongoDB, ensuring data is correctly retrieved and stored.**
3. **End-to-End (E2E) Testing:**
    - **Simulate user actions from logging in to booking a property using tools like Cypress or Selenium.**
4. **Usability Testing:**
    - **Conduct tests with a small group of users to gather feedback on the interface and ease of navigation.**
5. **Performance Testing:**
    - **Use tools like JMeter or Lighthouse to evaluate the app's loading times, responsiveness, and scalability under varying loads.**

Test Environment Setup

1. **Frontend: Run the React app in a browser with multiple resolutions for testing responsiveness.**
2. **Backend: Set up Node.js server and MongoDB for API and database testing.**
3. **Tools: Use tools like Jest, Mocha, or Cypress for automated testing.**
4. **Test Data: Populate the database with sample property and user data for realistic testing scenarios.**

Conclusion

The testing plan ensures that the house rent app functions reliably, provides an excellent user experience, and performs well under different conditions. A combination of automated and manual testing techniques helps identify and address issues efficiently.

11. Screenshots and demo:

demo link : Click here

demo video : click here

12. Known Issues

This section lists any current bugs, limitations, or challenges in the house rent application to provide transparency and guide further improvements.

List of Known Issues:

1. Search Functionality Delay:

- The search results sometimes take longer to display due to server response time.

- Impact: Affects user experience, especially for users with slower internet connections.

2. Filter Accuracy:

- Some filters (e.g., price range) may return irrelevant results in specific edge cases.

- Impact: May lead to user frustration when browsing properties.

3. Responsive Design:

- Minor layout issues on older browsers or very small screen resolutions.

- Impact: Causes UI components to overlap or break in certain scenarios.

4. Booking Confirmation Email:

- Occasionally, the confirmation email fails to send due to SMTP configuration issues.

- Impact: Users may not receive booking confirmation in their inbox.

5. Profile Picture Upload:

- Upload functionality is inconsistent, especially for large image files.

- Impact: Users may face difficulties updating their profile pictures.

---

13. Future Enhancements

This section outlines potential features and improvements to enhance the app's functionality and user experience in upcoming iterations.

Planned Enhancements:

1. Advanced Search and Filters:

- Add additional filters like property amenities (e.g., swimming pool, parking) and proximity to public transport.

- Expected Benefit: Improves search precision and user satisfaction.

2. Improved User Dashboard:

- Include analytics on user activity, such as recently viewed properties and recommendations.

- Expected Benefit: Enhances user engagement and personalization.

3. Multilingual Support:

- Provide language options to cater to a diverse user base.

- Expected Benefit: Expands the app's reach to international markets.

4. Enhanced Notifications:

- Integrate SMS and push notifications for booking updates and property recommendations.

- Expected Benefit: Keeps users informed in real time.

5. Payment Gateway Integration:

- Enable users to make rental payments directly through the app.

- Expected Benefit: Simplifies transactions and adds convenience for users.

6. Admin Panel for Property Management:

- Develop an admin dashboard for landlords to manage their properties, view bookings, and communicate with tenants.

- Expected Benefit: Streamlines property management and reduces developer intervention.

7. AI-Powered Recommendations:

- Use machine learning to suggest properties based on user behavior and preferences.

- Expected Benefit: Improves user retention and satisfaction.

8. Offline Mode:

- Allow users to browse previously viewed properties without an active internet connection.

- Expected Benefit: Enhances accessibility in areas with limited connectivity.

9. Bug Fixes and Optimizations:

- Address all known issues, optimize performance, and ensure smooth user interactions.

- Expected Benefit: Builds trust and reliability among users.

Conclusion

Documenting known issues and potential future enhancements ensures transparency and sets clear goals for ongoing development. Resolving current challenges while planning new features helps maintain a high-quality application that evolves with user needs.