**1.Introduction:**

**Project title: House Rent App**

Team member:

1.Ezhilarasi.P

2.Pooja.PN

3.Priya.M

4.Ramya.S

**2.Abstract:**

This Web Application helps user to register individual home or apartment toassist you in finding the perfect rental home or property. Also we can find yournext rental from search view in your targeted area. Online Rental System is aapplication for simple reservation services, rental service or any other service.Rental systems are easily integrated into your own and other affiliate websitesso any updates you make are automatically reflected across all sites.This is an internet portal dedicated to meet every aspect of the consumers’needsin the RealEstate industry. It is aforum where Property Owners,Brokers, Tenants and Investors can exchange information, quickly, effectivelyand inexpensively. It features commercial and residential properties for sale rentand lease the fore properties, rates, locations, property news

**3.Project overview:**

Purpose:house rental app helps renters find properties and landlords manage listings. Renters can browse properties, view photos, contact landlords, and apply online. Landlords can list properties, respond to inquiries, screen tenants, and collect rent. It streamlines the rental process, making it quicker, more convenient, and secure for both parties.

**Simplify the Rental Process:**Make it easy for renters to find homes and landlords to manage properties.

**Enhance Accessibility:**Provide a user-friendly platform for quick access to listings.

**Increase Efficiency:**Reduce the time and effort involved in searching, listing, and managing rentals.

**Facilitate Secure Transactions:**Ensure safe communication, applications, and payments.

**Improve Tenant-Landlord Communication:**Enable direct contact to streamline scheduling and negotiations.

Feature:

**Search and Filters:**Search by location, budget, property type, and amenities.

**Detailed Listings:**Property descriptions with photos, videos, and virtual

**Online Applications:**Digital application and tenant screening tools.Rent Payments: Secure, in-app rent collection.

**Notifications:** Alerts for new listings, messages, and payment reminders.

**Direct calling:**calling to their owner.

**4.Architecture:**

Frontend:

**1.Component-Based Structure:** Break down the UI into reusable components, organized in a tree hierarchy. Each component manages its own state and logic, ensuring modularity.

**2.State Management:**React State for local component state.Context API for shared state across components.External Libraries like Redux for complex global state, with middleware for async actions.

**3.Routing:** Use React Router to manage navigation, enabling single-page app behavior with URL-based routing.

**4.Styling:** Options include CSS Modules, Styled Components, or frameworks like Tailwind CSS for scoped, maintainable styles.

**5.API Integration:** Use libraries like Axios or Fetch API, typically managed by hooks (e.g., useEffect) or data-fetching libraries like React Query for efficient API handling.

**6.Build and Tooling**: Use tools like Webpack, Babel, and ESLint for efficient bundling, transpiling, and code quality.

Backend:

**1.Project Structure:** Organized in folders (e.g., controllers, models, routes, middlewares, services), promoting modularity and scalability.

**2.Core Features**:

* + **User Management**: User registration/login (JWT-based) and role-based access (Renters, Listers).
  + **Property Listings**: CRUD for property listings, with search and filter options.
  + **Bookings**: Renters can request bookings, and Listers can approve/manage them.
  + **Reviews**: Renters can review properties; reviews are shown on listings.

Database:

MongoDB with Mongoose, collections for User, Property, Booking, and Review.

**Endpoints**:

* + User routes for signup, login, profile.
  + Property routes for adding, updating, searching properties.
  + Booking routes to handle bookings.
  + Review routes for property reviews.

**Middleware**:

* + **Auth Middleware**: JWT verification for protected routes.
  + **Role-Based Access**: Restrict certain actions to Listers only.
  + **Error Handling**: Standardized error responses.

**Security and Deployment**:

* + **Authentication**: JWT for sessions.
  + **Deployment**: Cloud-hosted Node.js server , MongoDB Atlas for the database.

This setup provides a robust and secure backend structure for a rental app, covering essential operations and scalability.

**5.Setup Instructions:**

**Core Software Requirements:**

1. **Node.js** - JavaScript runtime for running the backend code.
2. **npm** (or **yarn**) - Node package manager to manage dependencies.
3. **MongoDB** - Database to store user, property, booking, and review data. Use MongoDB Atlas if a cloud database is needed.
4. **Postman** (optional) - For testing API endpoints.

**Backend Framework and Libraries**

1. **Express.js** - Web framework for routing and handling HTTP requests.
2. **Mongoose** - MongoDB Object Data Modeling (ODM) library for schema management.
3. **dotenv** - Environment variable management to securely store configuration.
4. **JWT (jsonwebtoken)** - For handling JSON Web Tokens in authentication.
5. **bcrypt** - To hash passwords securely.

**Middleware and Utility Libraries:**

1. **morgan** - HTTP request logging middleware for better development insights.
2. **express-validator** - Validation middleware for validating user input.
3. **nodemailer** - For sending email notifications (e.g., booking confirmations).

**Installation:**

Step 1:create a folder and open a folder in vscode

**Step 2:** Install Dependencies

Make sure Node.js and npm are installed. You can check with:

node -v

npm -v

If not, download and install Node.js (which includes npm) from <https://nodejs.org/>.

Install project dependencies using npm (or yarn if preferred):

npm install

**Step 3:** Set Up MongoDB

Install MongoDB (if using a local database):

Download MongoDB from https://www.mongodb.com/try/download/community and follow the installation instructions.

Start the MongoDB service locally:

mongodb

Alternatively, set up a cloud database using MongoDB Atlas.

Step 4: Configure Environment Variables

In the project’s root directory, create a .env file:

touch .env

Open the .env file in a text editor and add the following environment variables. Replace placeholder values with your actual configuration:

PORT=8000

MONGODB\_URI=mongodb://localhost:27017/mydatabase # or your MongoDB compass URI

JWT\_SECRET=your\_jwt\_secret\_key # Choose a strong secret key

EMAIL\_USER=your\_email@example.com # Used for nodemailer setup if sending emails

EMAIL\_PASS=your\_email\_password

PORT: Port number for the server to listen on (e.g., 5000).

MONGODB\_URI: MongoDB connection URI. Use your MongoDB URI here.

JWT\_SECRET: Secret key for JWT token encryption. Use a strong, unique key.

EMAIL\_USER and EMAIL\_PASS: Email credentials for sending notifications (only if email functionality is needed).

**Step 5:** Run the Server

Start the server:

npm start

Or, for development with automatic restarts (requires nodemon):

npm install -g nodemon

nodemonindex.js

The server should now be running at http://localhost:3000 (or your specified port).

**Step 6:** Test the API Endpoints

Use Postman or curl to test endpoints. For example, to check if the server is running, send a GET request to:

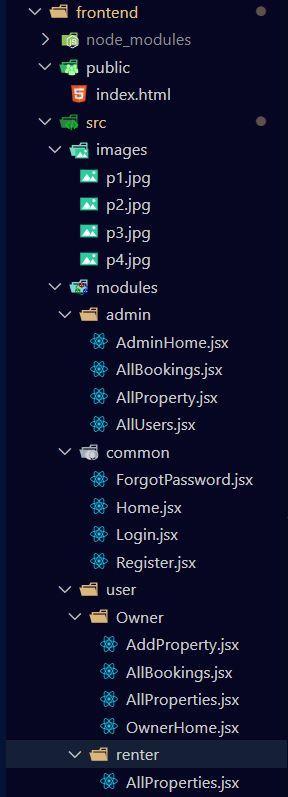
http://localhost:3000/api/login

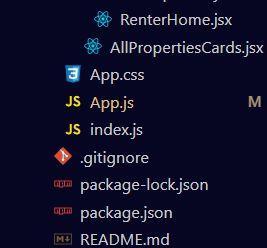
With this setup, your environment should be ready to develop and test the backend for HouseRentApp.

**6.Folder structure:**

Client:

houserentapp-frontend/

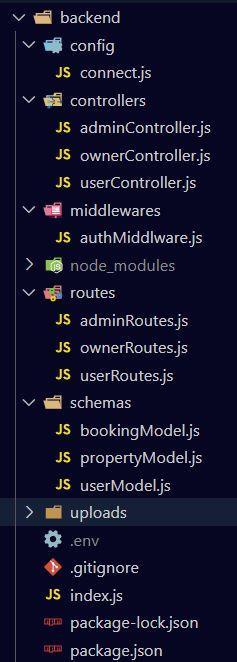




**Key Features**

* **Routing**: react-router-dom for navigation (home, property, search, etc.).
* **Global State**: Context API for authentication and data sharing.
* **Services**: Centralized API requests with Axios in services/.
* **Responsive Design**: Ensures mobile-friendly UI.
* **Environment Config**: .env file for API base URL and keys.

Server:



**summary**

* **Modularity**: Each feature has its own controller, route, and service.
* **Middleware**: Handles auth and error responses.
* **Environment Config**: Uses .env for secure and flexible setup.

**7.Running the application:**

**Frontend:**

**1.Navigate to the client directory**

cd frontend  
**2.Install dependencies** (if not already done):

npm install

**3.Start the development server**:

npm start

**Backend:**

**1.Navigate to the client directory**

Cdbackend  
**2.Install dependencies** (if not already done):

npm install

**3.Start the development server**:

npm start

**8.Authentication:**

1. Authentication and authorization in the HouseRentApp are handled using JWT (JSON Web Tokens) and role-based access control.
   1. Authentication:
      * Users sign up with their credentials (passwords are hashed).
      * Upon login, the backend generates a JWT that includes the user’s ID and is sent to the client.
      * For subsequent requests, the client sends the token in the Authorization header.
      * The backend verifies the token to authenticate the user.
   2. Authorization:
      * Role-based: Users have roles (e.g., user, admin).
      * Specific routes are protected based on the user's role (e.g., only admin can manage properties).
      * Middleware checks the user’s role and grants access accordingly.

Top of Form

Bottom of Form

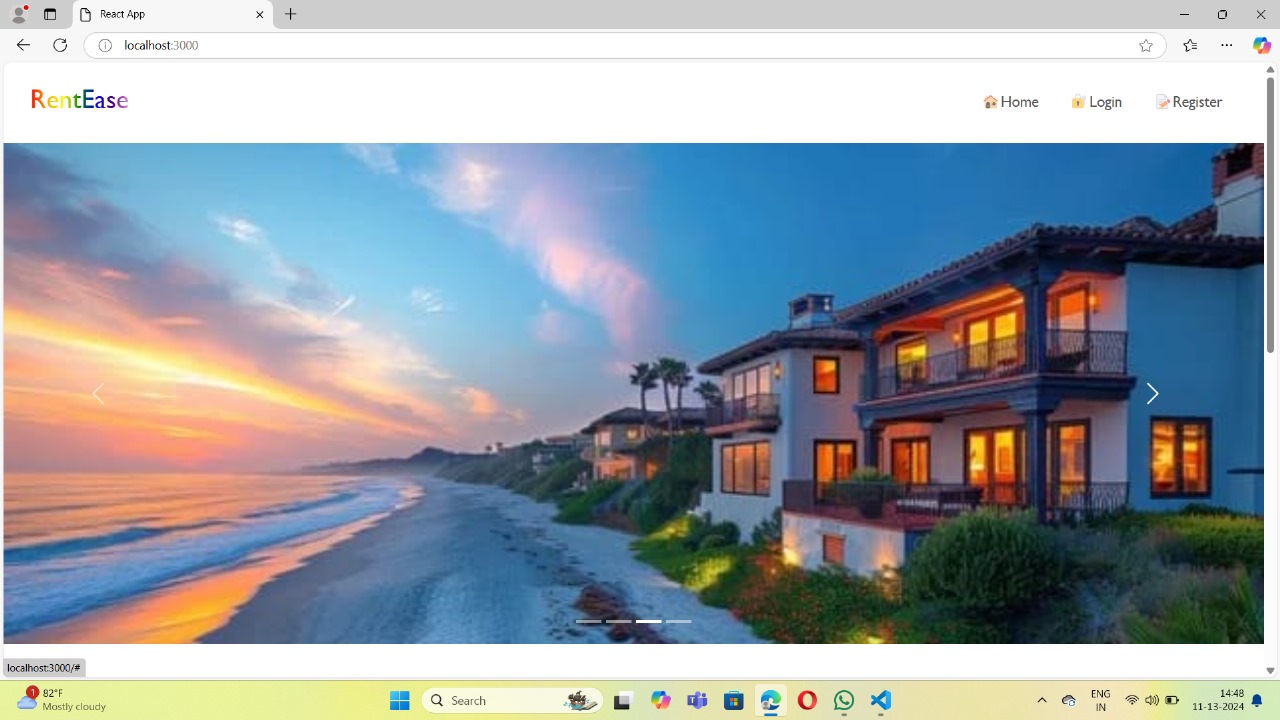
2.In the HouseRentApp, **JWT (JSON Web Tokens)** are used for **authentication** and **session management**.

* **Login**: When the user logs in, the backend generates a **JWT** with the user's ID and role, and sends it to the client.
* **Token Storage**: The token is stored in **localStorage**, **sessionStorage**, or **HTTP-only cookies** on the client-side.
* **Token Usage**: For every request, the client includes the token in the Authorization header (Bearer <token>).
* **Token Verification**: The backend verifies the token on each request to authenticate the user.
* **Expiration**: Tokens expire after a set time (e.g., 1 hour). If expired, the user must log in again.

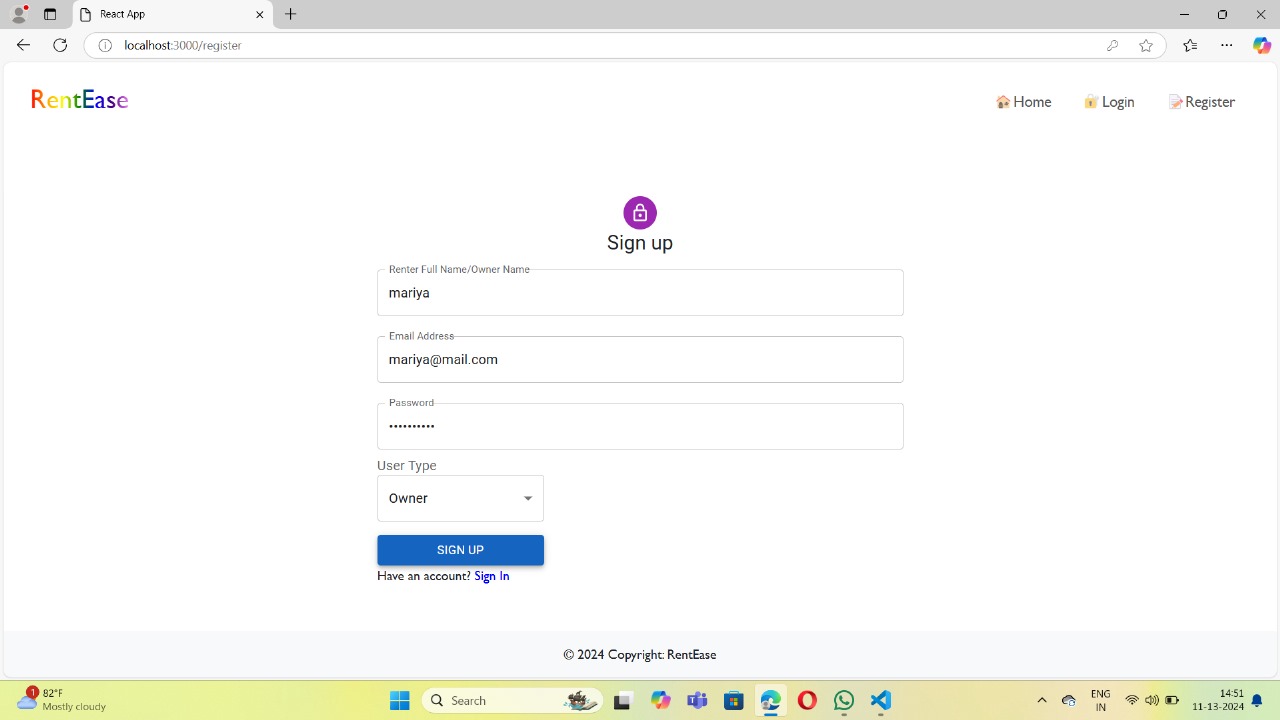
**Sessions** are not used in this app, as JWT is stateless, meaning no server-side session storage is needed. Optionally, **refresh tokens** can be used to extend sessions without forcing the user to log in again.

**9.User Interface:**

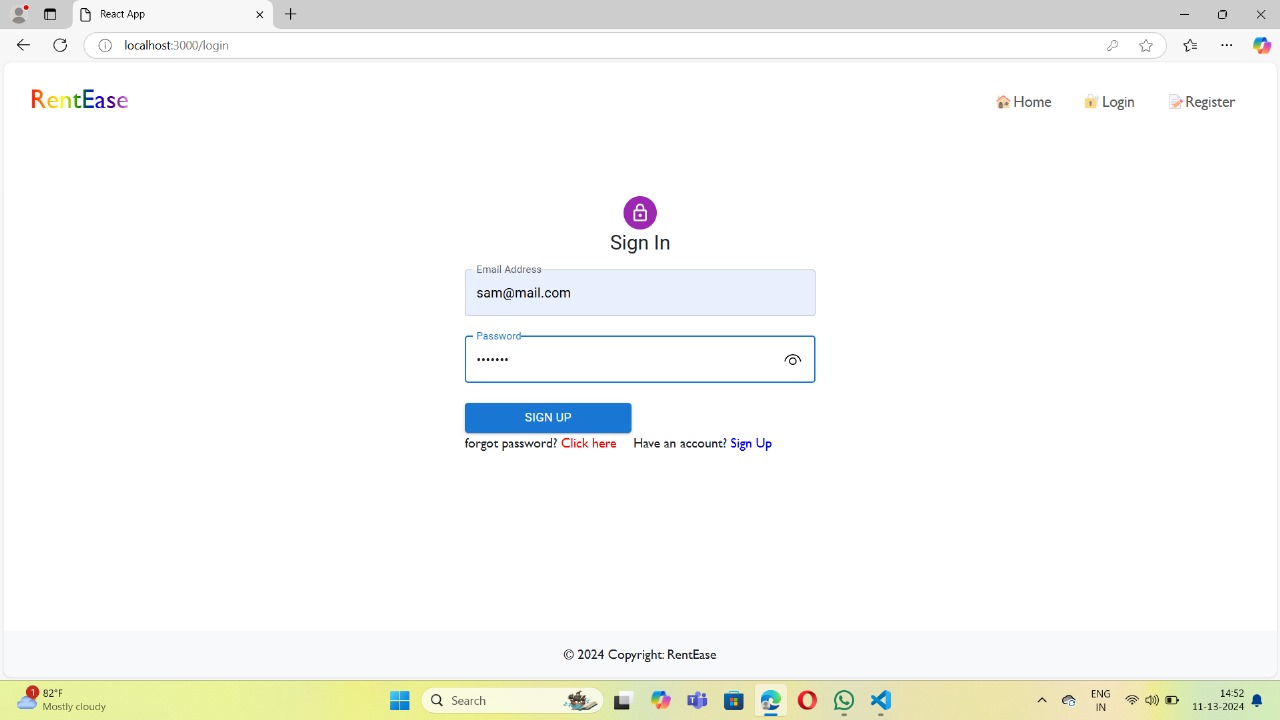
**Home Page:**

****

**Login Page:**

****

**Register Page:**

****

**10.Testing:**

Backend Testing:

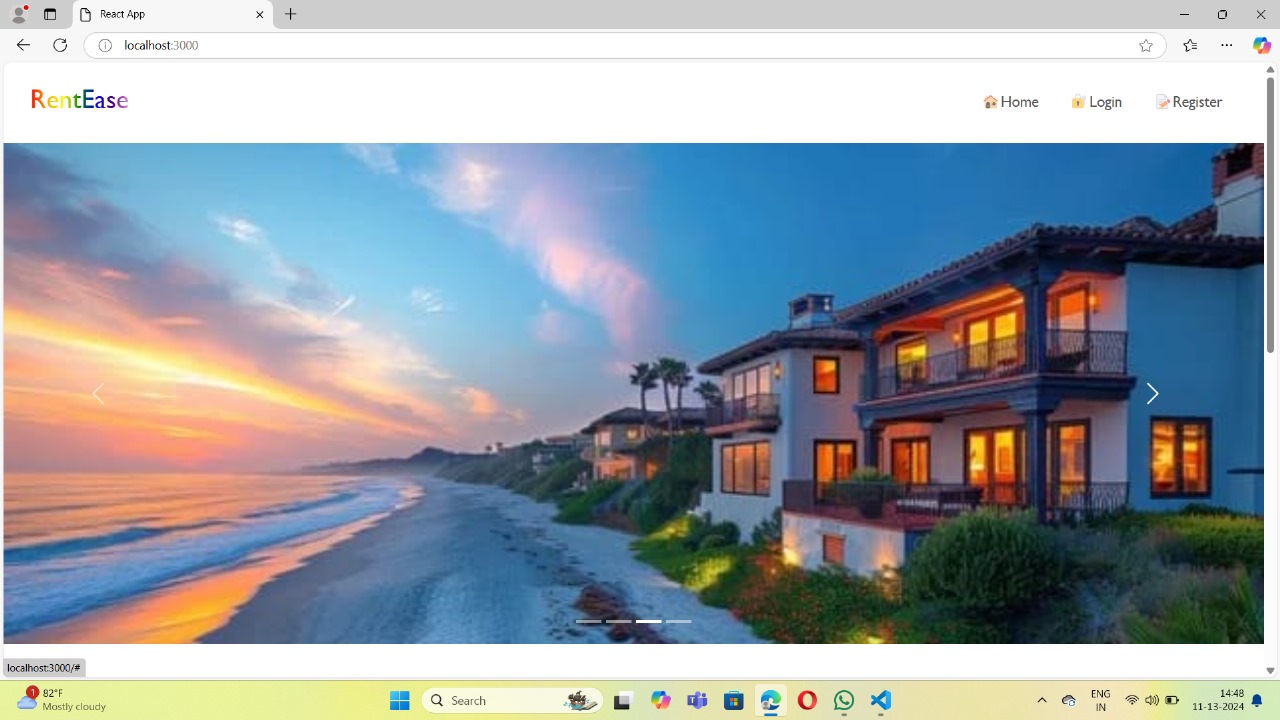
* Tools:
  + Jest: For unit and integration tests.
  + Supertest: For API testing (testing HTTP requests and responses).
  + MongoDB Memory Server: For in-memory database testing.
* Focus: Unit, integration, and API endpoint testing (e.g., login, property management).

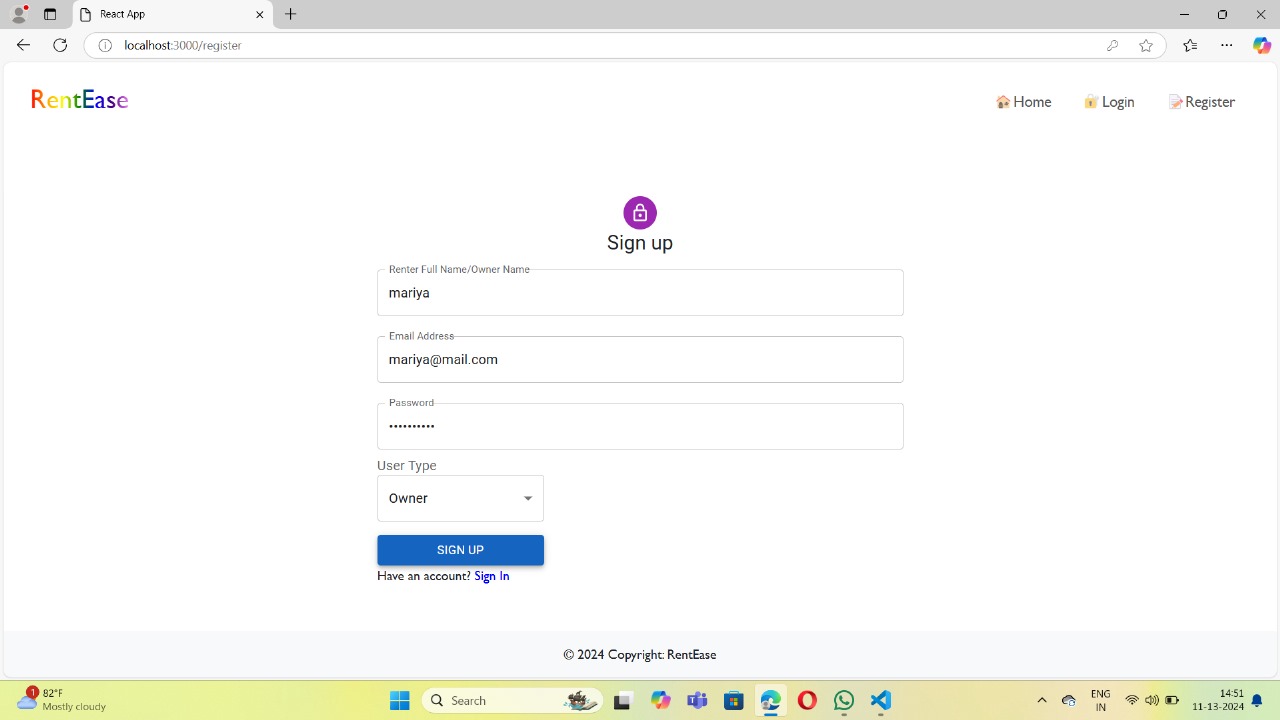
Frontend Testing:

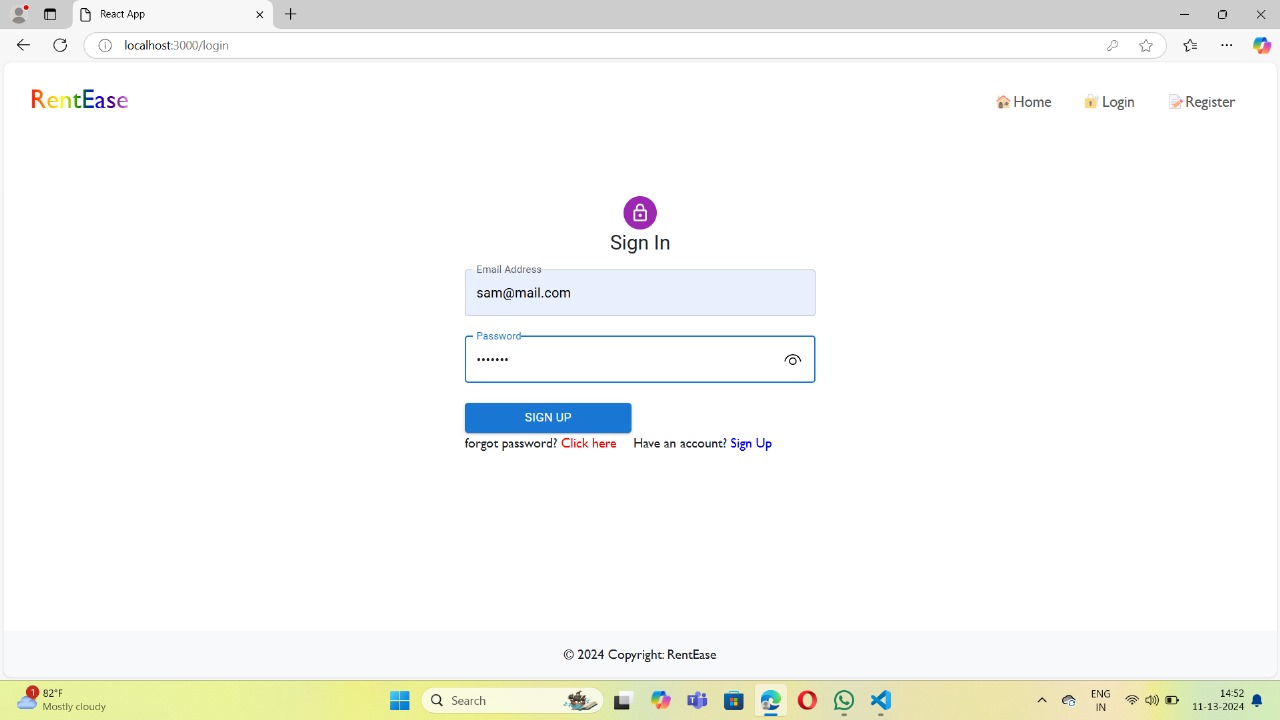
* Tools:
  + Jest: For unit and integration tests.
  + React Testing Library: For testing React components and user interactions.
  + Cypress: For end-to-end browser testing.

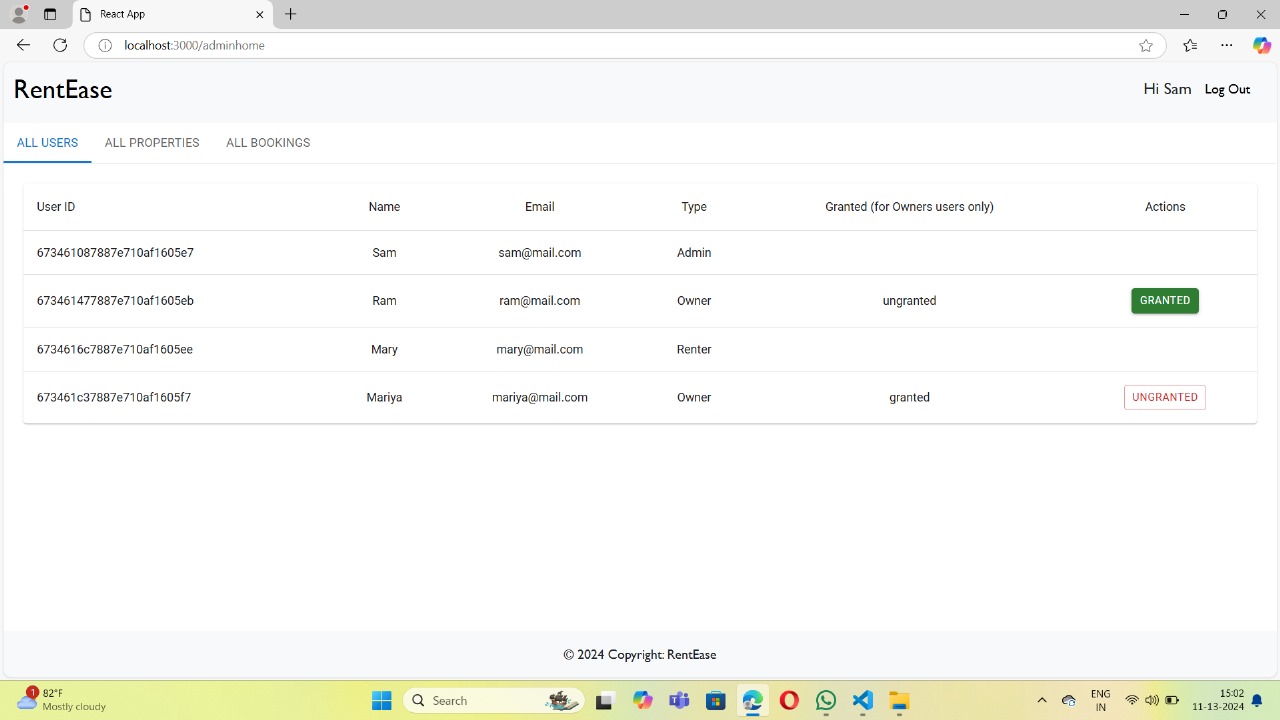
Focus: Component testing, form submission, and user flows.

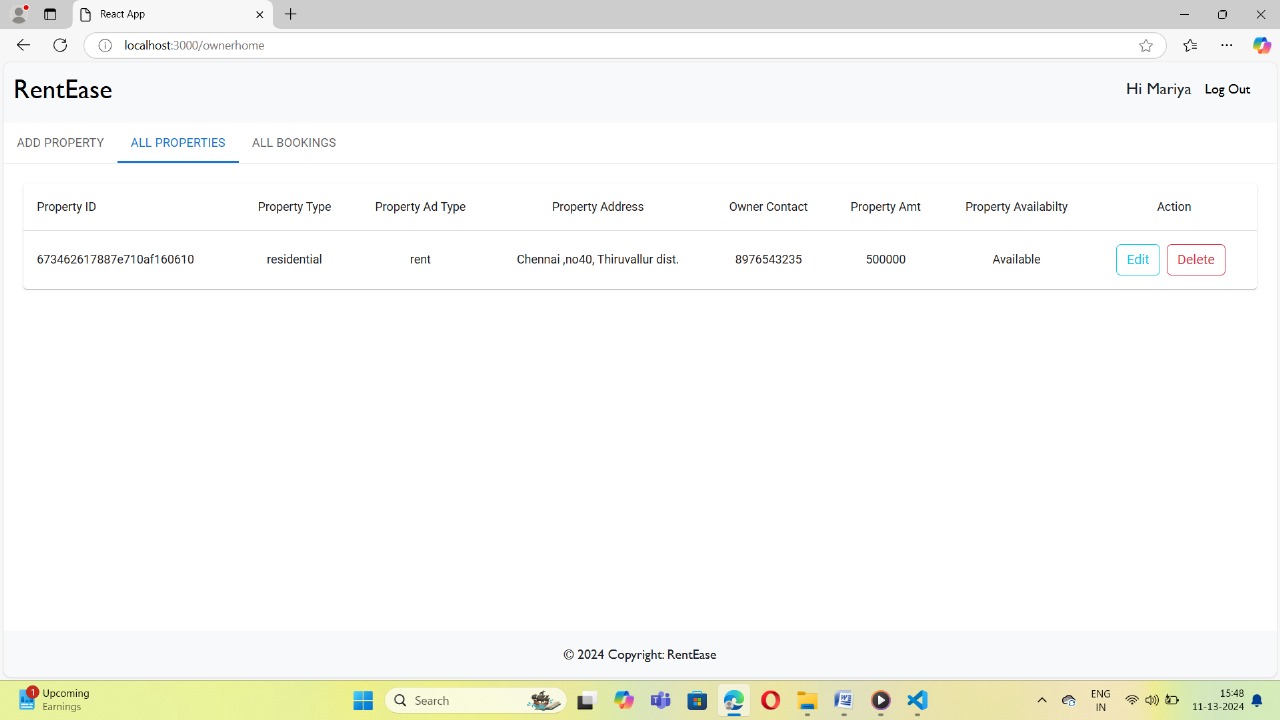
**11.Screenshots or Demo:**

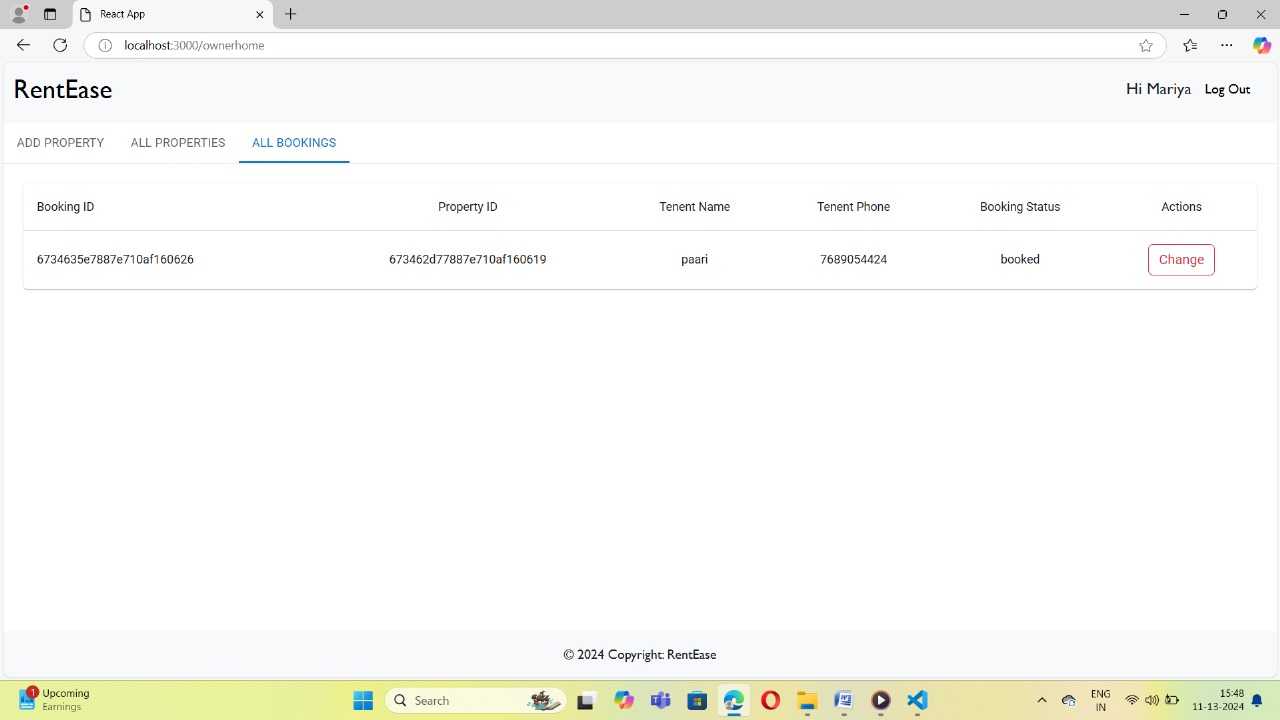
****

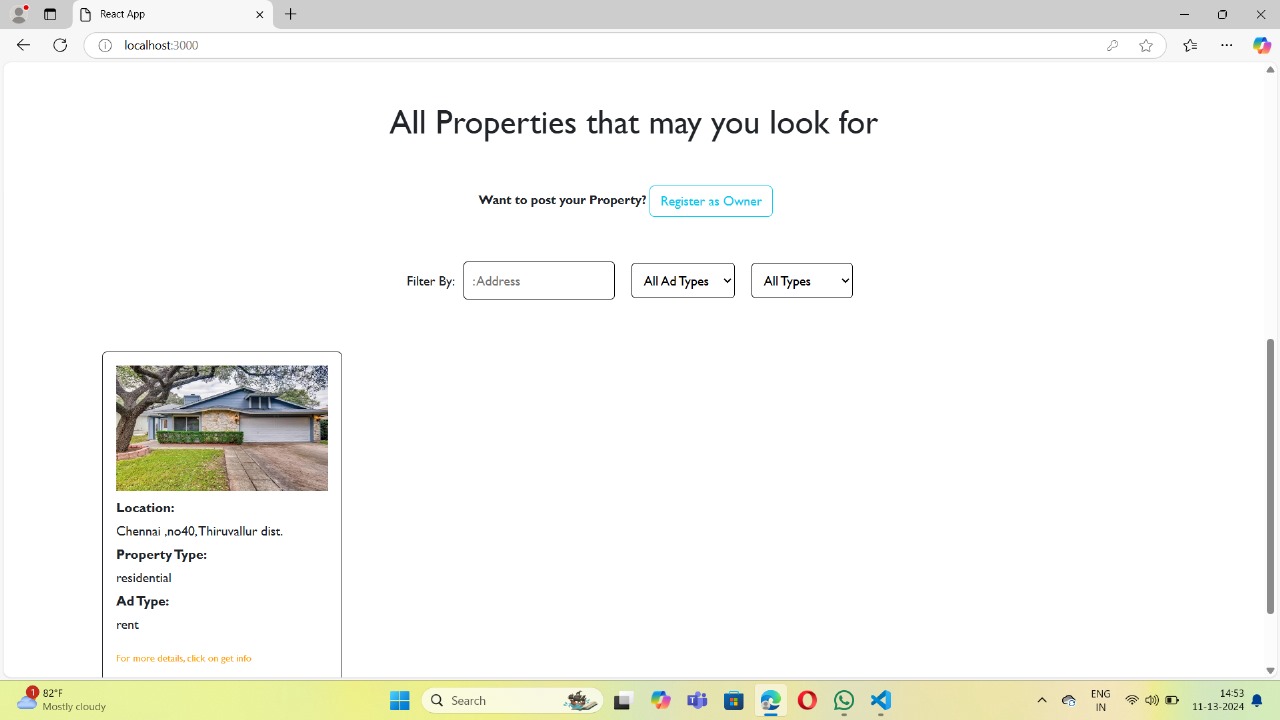
****

****

****

****

****

****

**12.Known Issues:**

1. Known Bugs/Issues
   1. Frontend:
      * UI glitches on mobile screens.
      * Inaccurate form validation.
      * Property search filter issues.
   2. Backend:
      * JWT expiry handling may cause unexpected logouts.
      * Occasional CORS issues between frontend and backend.
   3. Database:
      * Data syncing delays on concurrent updates.
      * Rare cases of duplicate entries for users or properties.
   4. Developer:
      * Missing or incorrect environment variables causing server failures.
      * Dependency version conflicts.
      * Testing requires mock data or in-memory database setup.
   5. Limitations:
      * Limited support for older browsers (e.g., Internet Explorer).
      * File upload may not handle large files or certain formats well.

Planned Fixes

* 1. Improve search filter accuracy and error handling.
  2. Optimize performance and address file upload issues.

Top of Form

Bottom of Form

**13.Future Enhancements:**

1. **User Features**:
   * Profile management, favorites/bookmarks, property ratings/reviews.
   * Booking system with payment gateway integration.
   * Push notifications for updates and new listings.
2. **Admin Features**:
   * Admin dashboard for property and user management.
   * Analytics to track usage and bookings.
3. **Backend**:
   * Real-time notifications, payment integration, multilingual support.
   * Optimized search algorithms.
4. **Security**:
   * Two-factor authentication, role-based permissions.
   * Rate limiting for API protection.
5. **Performance**:
   * Image optimization, caching strategies for faster load times.
6. **AI/ML**:
   * Property recommendation system and AI-driven chatbot for support.

These improvements would enhance user experience, app scalability, and security.