**FULL STACK DEVELOPMENT WITH MERN**

**MERN STACK BY MONGODB**

**PROJECT:**

**HOUSE RENTAL APPLICATION USING MERN**

**Team Members:**

KARTHIGAI KANNAN K (311521104024)

CHARAN M (311521104008)

ISAAC SAMUEL J (311521104018)

PRADEEP KUMAR G (311521104032)

**1. INTRODUCTION**

* **Project Title:** House Rent Application
* **Team Members:**

Karthigai Kannan K – Team Lead

Charan M- Member

Isaac Samuel J- Member

Pradeep Kumar G- Member

**2. PROJECT OVERVIEW**

### Technology Stack

* **Frontend:** React.js
* **Backend:** Node.js with Express
* **Database:** MongoDB
* **Authentication:** JWT (JSON Web Tokens) or similar

The goal of this project is to develop a web-based House Rent Application that allows users to search, list, and manage rental properties. The application will enable landlords to post property listings, and prospective tenants to search, view details, and inquire about properties. The MERN stack (MongoDB, Express.js, React, Node.js) will be used to build this application.

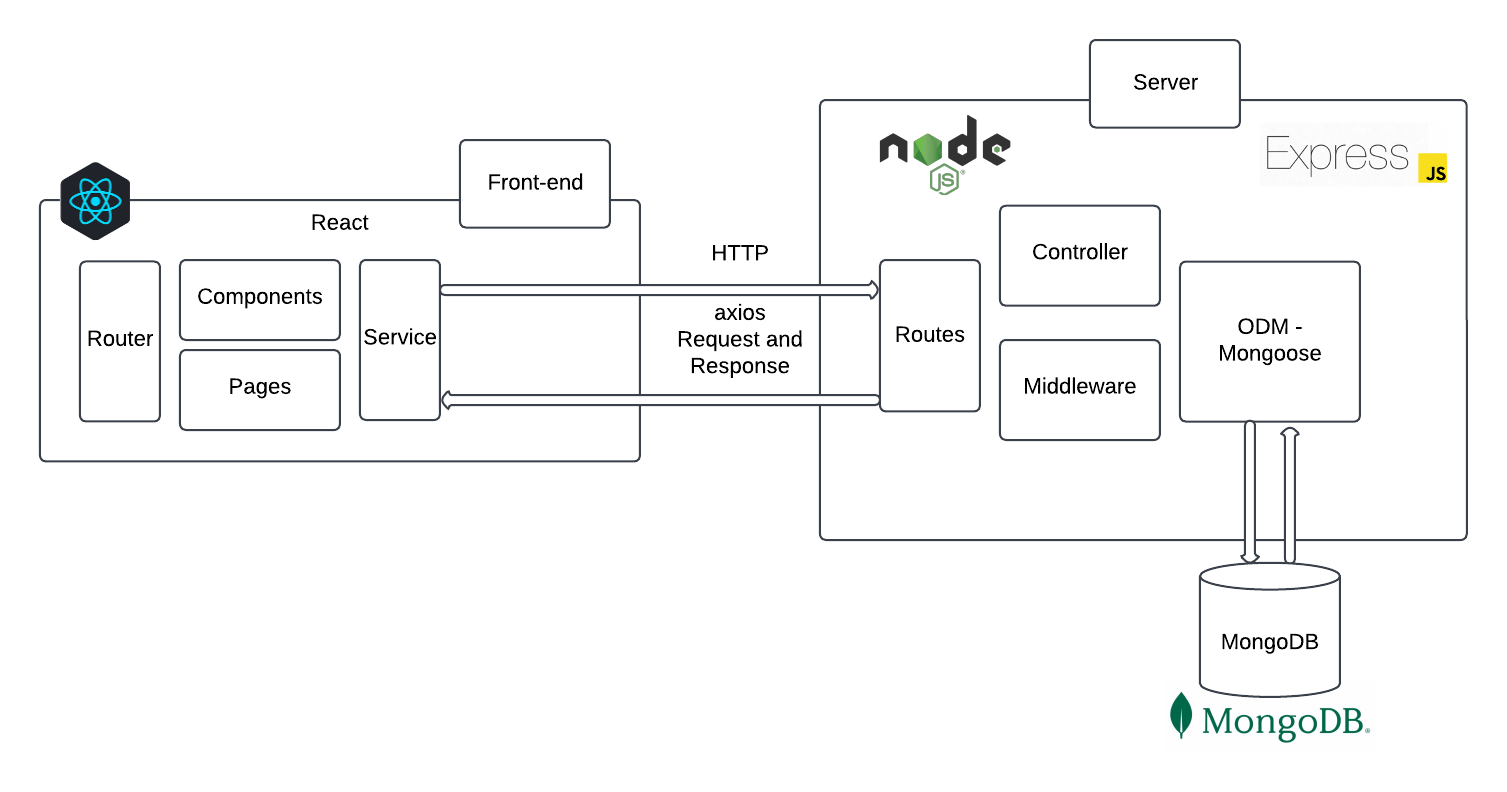
**PURPOSE:**

The purpose of the House Rent Application is to create a digital platform that connects property owners and potential tenants. The platform simplifies the process of renting houses by providing a secure and user-friendly interface for searching, listing, and managing rental properties.

### KEY FEATURES

1. **User Authentication**
   * Sign up, login, and logout features for users (tenants and property owners).
   * Role-based access control to distinguish between tenants and property owners.
2. **Property Listings**
   * Property owners can add new property listings, including details like rent amount, property location, photos, and amenities.
   * Tenants can browse and filter listings based on location, price, property type, and other criteria.
3. **Search and Filter**
   * A search and filter option for tenants to find listings based on specific criteria (e.g., address, Ad type(rent/sale), Property type(residential/commercial/plots)).
4. **Detailed Property Page**
   * A dedicated page for each property with all relevant details, photos, a description, and owners contact information.
   * An option for tenants to save or bookmark properties for later.
5. **Booking or Viewing Requests**
   * Tenants can send requests to view the property or book it.
   * Property owners can manage these requests and respond to tenants.
6. **Admin Dashboard**
   * A dashboard for admins to oversee all activity on the platform, including user management, property listings, and resolving disputes.

**3. ARCHITECTURE**

****

*FIGURE 1.ARCHITECTURE*

**FRONTEND:**

* **React.js:** The frontend will be developed using React to create a responsive, single-page application (SPA). Key components will include:
  + **Home Component:** Displays featured properties and search filters.
  + **Property Listing Component:** Shows all available properties with filter options.
* **Property Details Component:** Provides detailed information about a specific property.
* **User Profile Component:** Allows users to manage their personal information, listed properties, and saved rentals.
* **Admin Dashboard Component:** Restricted access dashboard for administrators to monitor activity and manage content.
* **State Management:** Options include using React Context API or Redux for managing the application state and API responses, such as user data, property listings, and notifications.
* **Authentication:** Use JWT for handling user authentication, where tokens are stored in local storage or HTTP-only cookies.
* **Routing:** React Router for managing navigation between pages without reloading the entire app.

**BACKEND:**

**Node.js and Express.js:** The backend will be built using Node and Express to handle HTTP requests, API routes, and business logic.

* **RESTFUL API:** Build a set of APIs for the following functionalities:
  + **User Authentication:** Routes for signup, login, logout, and password reset.
  + **Property Management:** CRUD operations for property listings (create, read, update, delete).
  + **Messaging System:** API routes for sending and receiving messages.
  + **Admin Operations:** Routes for managing users, properties, and reports.
* **Security and Authorization:** Use JWT for authorization, with middleware for protecting routes based on user roles (e.g., admin, tenant, landlord).
* **File Uploads:** Set up an endpoint for image uploads for property photos, potentially using a service like Cloudinary for storing images.

**DATABASE:**

The database uses MongoDB with a schema defined through Mongoose. Collections are created to manage Users, Products, Orders, and Cart items, ensuring data integrity and efficient querying.

**MongoDB:**

Use MongoDB to store data related to users, properties, reviews, messages, and more.

**Database Models:**

* + - **User Model:** Stores user information such as name, email, password (hashed), role (tenant, landlord, admin), and profile details.
    - **Property Model:** Stores information about each property, including address, description, rent price, owner ID, images, and other details.
    - **Message Model:** Stores message threads between users.
    - **Booking or Viewing Requests Model:** Stores tenant requests for viewing or booking properties.
    - **Review Model:** Stores reviews submitted by tenants for properties and landlords.
* **Indexes:** Ensure frequently searched fields, like location and price, are indexed for faster queries

**4. SETUP INSTRUCTIONS**

**BACKEND PREREQUISITES:**

**1. Prerequisites:**

* Node.js and npm: Download and install from [Node.js](https://nodejs.org).
* MongoDB: You can set up a local MongoDB instance or use MongoDB Atlas for a cloud database.
* Git: For version control (optional but recommended).

**2 .Installation:**

**Navigate to the backend directory:** cd backend

1. **Install the backend dependencies:  
   bash**npm install express mongoose dotenv jsonwebtoken bcryptjs cors

**2.Install development dependencies for easier development:  
 bash** npm install --save-dev nodemon

**3.Create a .env file for environment variables in the backend directory and add the following variables (with appropriate values for your setup):**

**makefile** MONGO\_URI=your\_mongodb\_connection\_string

JWT\_SECRET=your\_jwt\_secret

PORT=3000

**4.Start the backend server in development mode:** npm start

**5.This will run the backend on**  [**http://localhost:3000**](http://localhost:3000) **.**

**FRONTEND PREREQUISITES:**

**1.Install the frontend dependencies:**npm install react react-dom react-router-dom axios

**2.Start the React application:**npm start

This will start the frontend server on <http://localhost:3000>.

### 3. Install Frontend Dependencies

Navigate to the client directory:cd frontend

**5. FOLDER STRUCTURE**

**6. RUNNING THE APPLICATION**

**7. API DOCUMENTATION**

**8. AUTHENTICATION**

Authentication is the process of verifying the identity of a user or system to ensure that the individual or entity is who they claim to be. It is a critical aspect of web application security, ensuring that only authorized users can access certain resources or functionalities.

### Types of Authentication Methods

1. **Basic Authentication**:

* **Description**: Involves sending a username and password in the HTTP header. This method is simple but not secure on its own, as credentials are often transmitted in plaintext unless the connection is secured (e.g., over HTTPS).
* **Use Cases**: Often used for quick, low-level access where security isn’t a major concern or in legacy systems.

1. **Form-Based Authentication**:
   * **Description**: Involves a login form where users enter their credentials (username and password), which are then validated by the server. This method can be enhanced with additional features like session management (cookies) and encryption.
   * **Use Cases**: Commonly used in web applications, especially those that require user registration and login.
2. **Token-Based Authentication (e.g., JWT)**:
   * **Description**: The server generates a token (usually a JSON Web Token, JWT) after the user logs in. The token is then sent with each request, typically in the HTTP Authorization header. This method is stateless and doesn’t require the server to store session data.
   * **Use Cases**: Ideal for RESTful APIs and single-page applications (SPAs), especially for handling distributed, scalable systems.
3. **Session-Based Authentication**:
   * **Description**: After a successful login, the server creates a session and stores session information, typically in a cookie. The session is used to track user interactions and maintain login state without requiring the user to re-enter credentials.
   * **Use Cases**: Most traditional web applications use session-based authentication for maintaining user sessions.
4. **API Key Authentication**:
   * **Description**: Involves sending an API key, which is a unique identifier, in requests to authenticate API calls. The server verifies the key and grants access to the requested resources.
   * **Use Cases**: Common in APIs and microservices, especially for public or internal APIs.

### Authentication Flow

1. **User Login**:
   * The user submits credentials (username and password) via a login form or API endpoint.
2. **Server Validation**:
   * The server checks the credentials against a database. If the credentials are valid, the server generates a session or token.
3. **Session/Token Creation**:
   * For session-based authentication, the server creates a session and stores it in memory or a database. For token-based authentication (like JWT), a token is generated and sent to the client.
4. **Accessing Resources**:
   * The client sends the session ID or authentication token with each request to access protected resources.
5. **Logout/Session Expiry**:
   * On logout, the session is destroyed, or the token expires, requiring the user to authenticate again.

### Authentication Security Best Practices

1. **Use HTTPS**: Always use HTTPS to encrypt credentials during transmission, ensuring that sensitive data like usernames and passwords aren’t exposed during transit.
2. **Password Hashing**: Store passwords securely by hashing them (using algorithms like bcrypt or Argon2) instead of storing them in plaintext.
3. **Avoid Storing Plaintext Tokens**: If using tokens like JWT, ensure they are signed and optionally encrypted to prevent tampering.
4. **Limit Login Attempts**: Implement mechanisms to prevent brute-force attacks by limiting the number of login attempts per user.
5. **Session Timeout**: Implement session expiration to reduce the risk of unauthorized access in case a session is hijacked.

**Secure Cookie Attributes**: Set cookies with secure attributes such as HttpOnly, Secure, and SameSite to prevent cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.

**9. USER INTERFACE**

The User Interface (UI) plays a crucial role in allowing users to interact with a web application, particularly when it comes to logging in and authenticating their identity. In the context of a House Rent Application using the MERN stack, the login process and integration with a MongoDB database ensure that user credentials are stored and securely managed.

### User Login Process

1. **Login UI**:
   * The login page allows users to input their credentials (username and password). This page typically includes a form with fields for the user’s **username/email** and **password**, along with a button to submit the form and attempt to log in.
   * Additional options, such as a **forgot password** link or a **signup** link for new users, might also be available.
2. **Submitting Credentials**:
   * Once the user enters their information, the form data is sent to the backend for verification. The backend checks the credentials against the stored user data in the database (MongoDB).

### Backend Authentication Process

1. **MongoDB Database**:
   * MongoDB stores user data, including usernames, email addresses, and hashed passwords. The database is structured as a collection of documents, each representing a user’s profile.
   * When a user submits their login credentials, the backend queries the MongoDB database to find the document that matches the entered username or email.
2. **Password Verification**:
   * The backend compares the entered password with the hashed version stored in MongoDB. This ensures that passwords are not stored in plaintext, offering better security for the user’s sensitive data.
3. **Session or Token Creation**:
   * Once the credentials are validated, the server generates a session or an authentication token (e.g., JWT - JSON Web Token) that is sent back to the frontend. This token is used to authenticate future requests from the user without needing to re-enter credentials.

### Storing Data in MongoDB

1. **User Data Storage**:
   * After user registration (sign-up), the credentials (username, email, password) are stored in MongoDB. The password is hashed before it is saved, ensuring that sensitive data is protected.
   * MongoDB provides flexibility in storing various types of data, including user profiles, preferences, and other personalized settings.
2. **Data Retrieval**:
   * Once the user is logged in, their session or token allows them to access personalized resources, such as viewing rental listings, updating their profile, or managing other data specific to the application.
   * The backend queries MongoDB to retrieve the necessary data for the authenticated user.

### Session Management and Access Control

### Session Handling:

* + After successful login, a **session** is created on the server, or a **token** is issued. This token is stored on the client side (usually in local storage or cookies) and is included in subsequent requests to authenticate the user.
  + Sessions or tokens ensure that users remain logged in across different pages or actions within the application.

1. **Access Control**:
   * The backend checks the session or token with each request to confirm that the user is authorized to access certain resources. For example, a user can view their rental listings or update their profile, but they cannot access another user's data unless they have the appropriate permissions.

### User Interface Feedback

1. **Successful Login**:

* + Upon successful authentication, the UI typically redirects the user to the main dashboard or home page. The user can now interact with the application, such as browsing rental listings, submitting inquiries, or updating personal information.

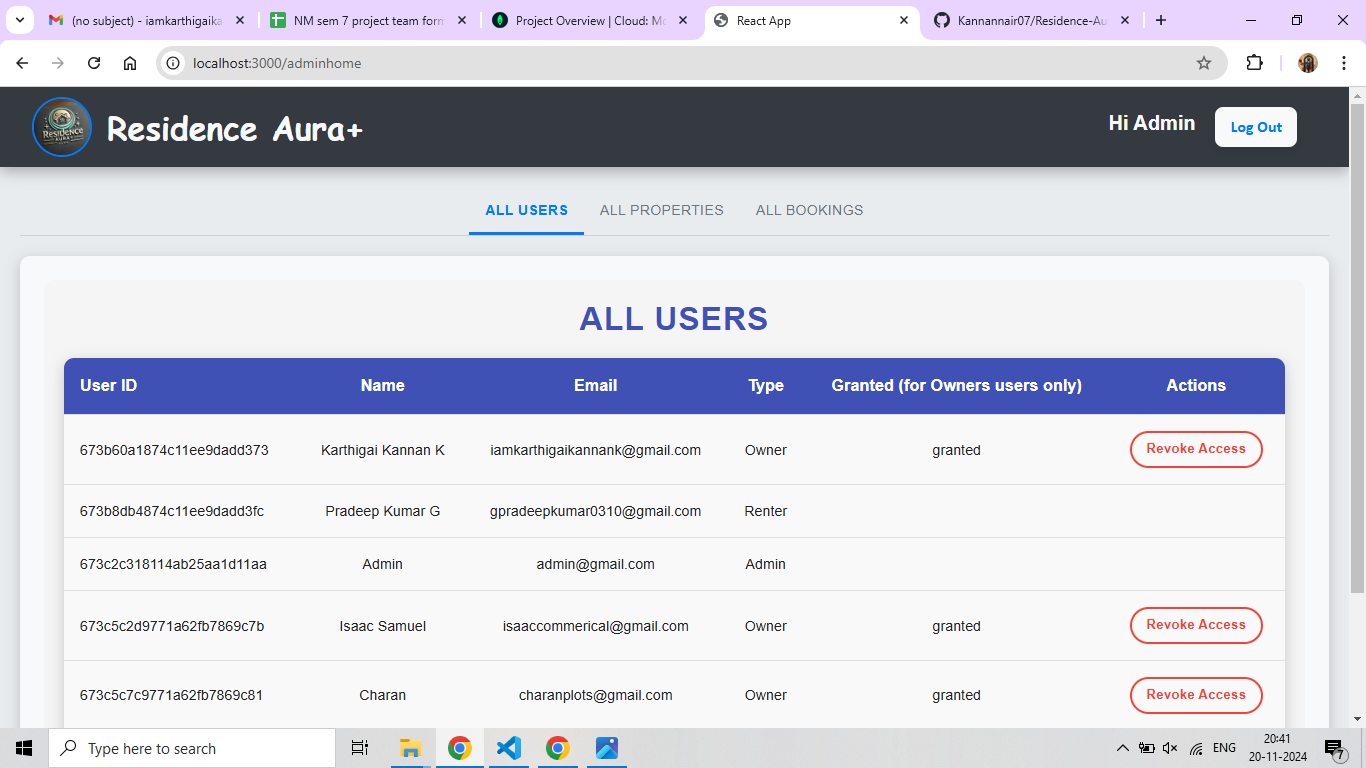
1. **Error Handling**:
   * If the login attempt fails (e.g., incorrect credentials), the UI will show an error message, prompting the user to check their username and password or try resetting their password.

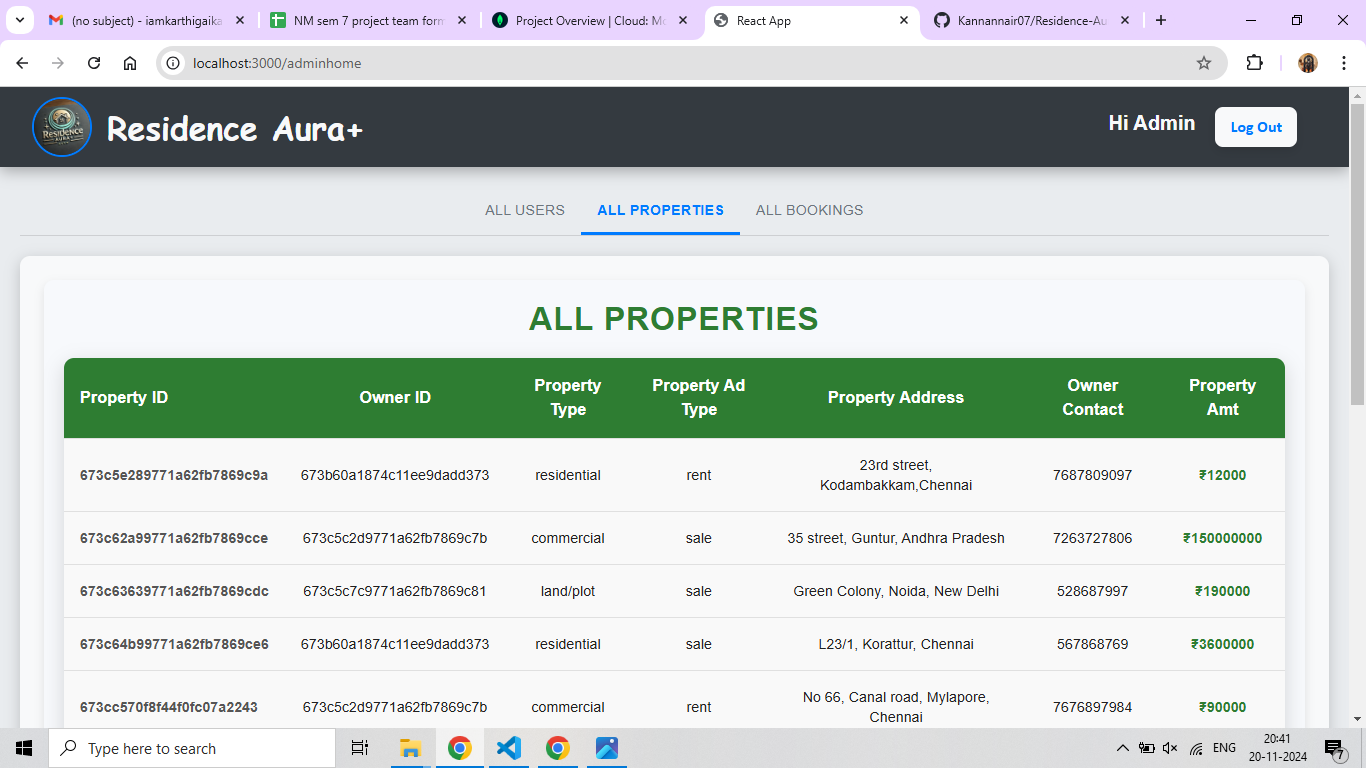
### Security Considerations

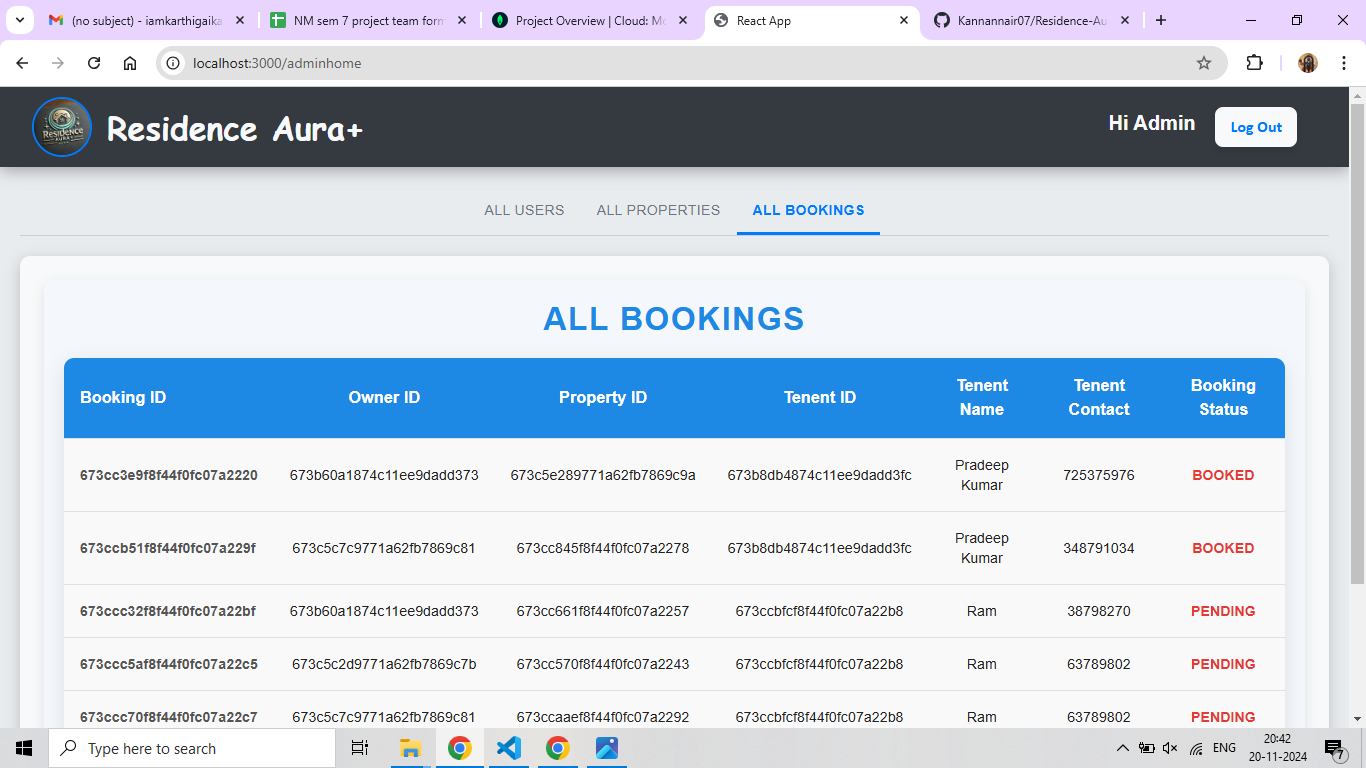
1. **Data Protection**:
   * Sensitive information such as passwords is never stored in plain text. Instead, they are securely hashed before being saved in MongoDB. This helps prevent unauthorized access if the database is compromised.
2. **Token Expiry**:
   * Tokens used for authentication often have an expiration time. This helps prevent unauthorized access if a session is left open for too long, enhancing security.

**11.SCREENSHOTS & DEMO**

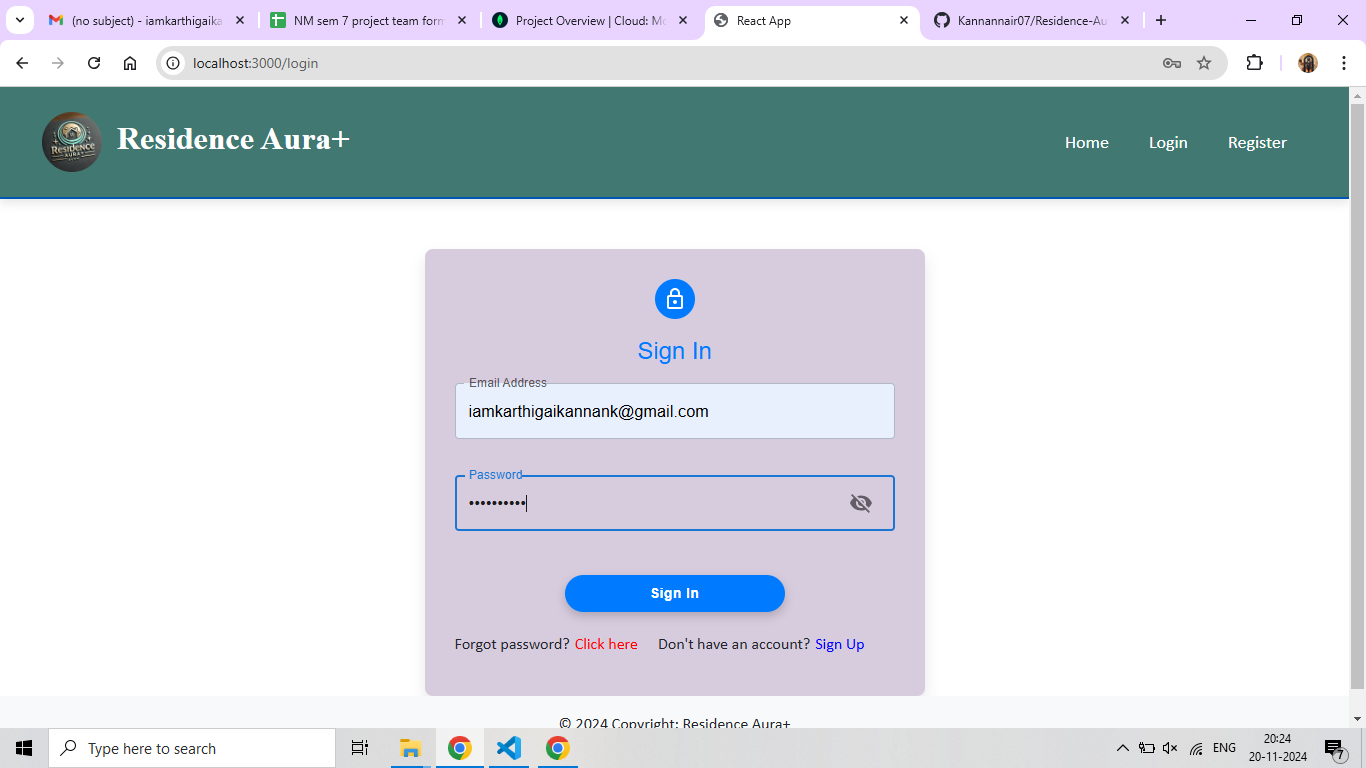
**Admin Page:**



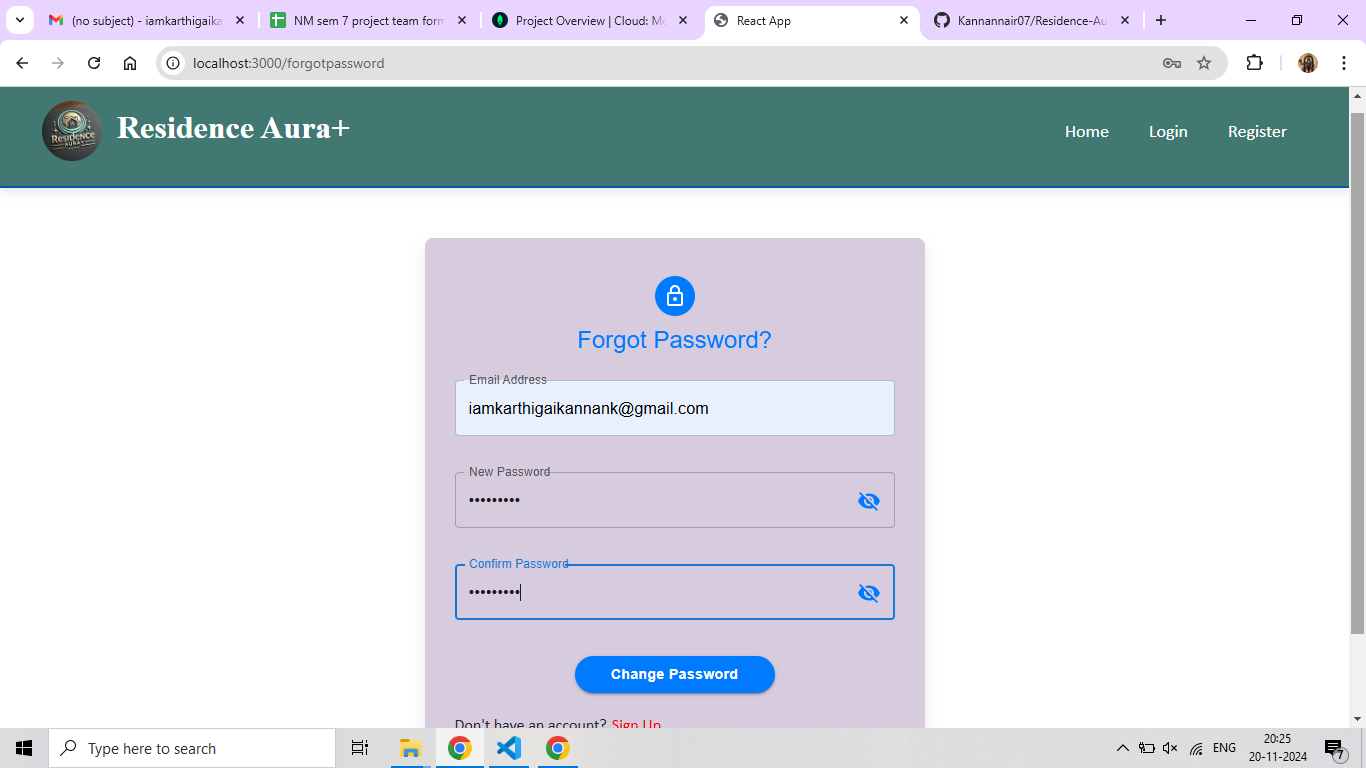




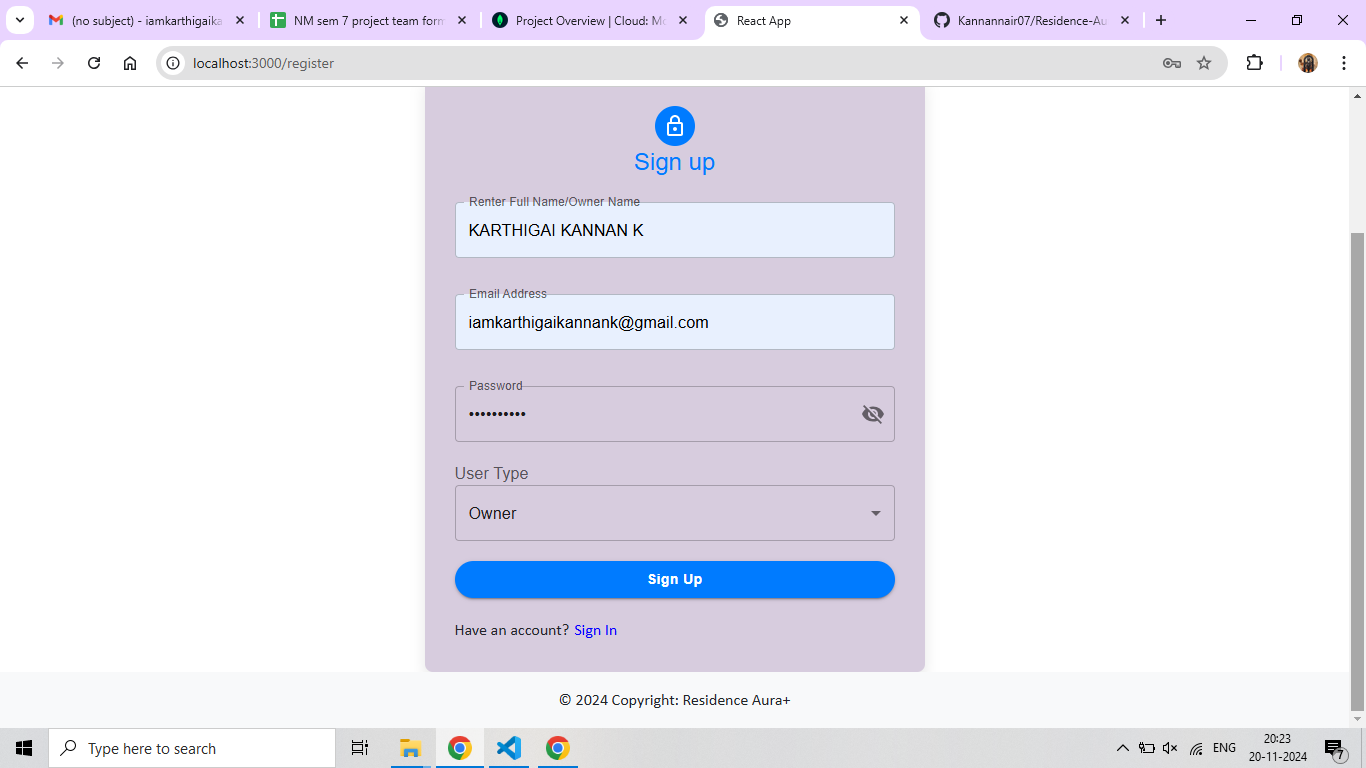
**Login:**



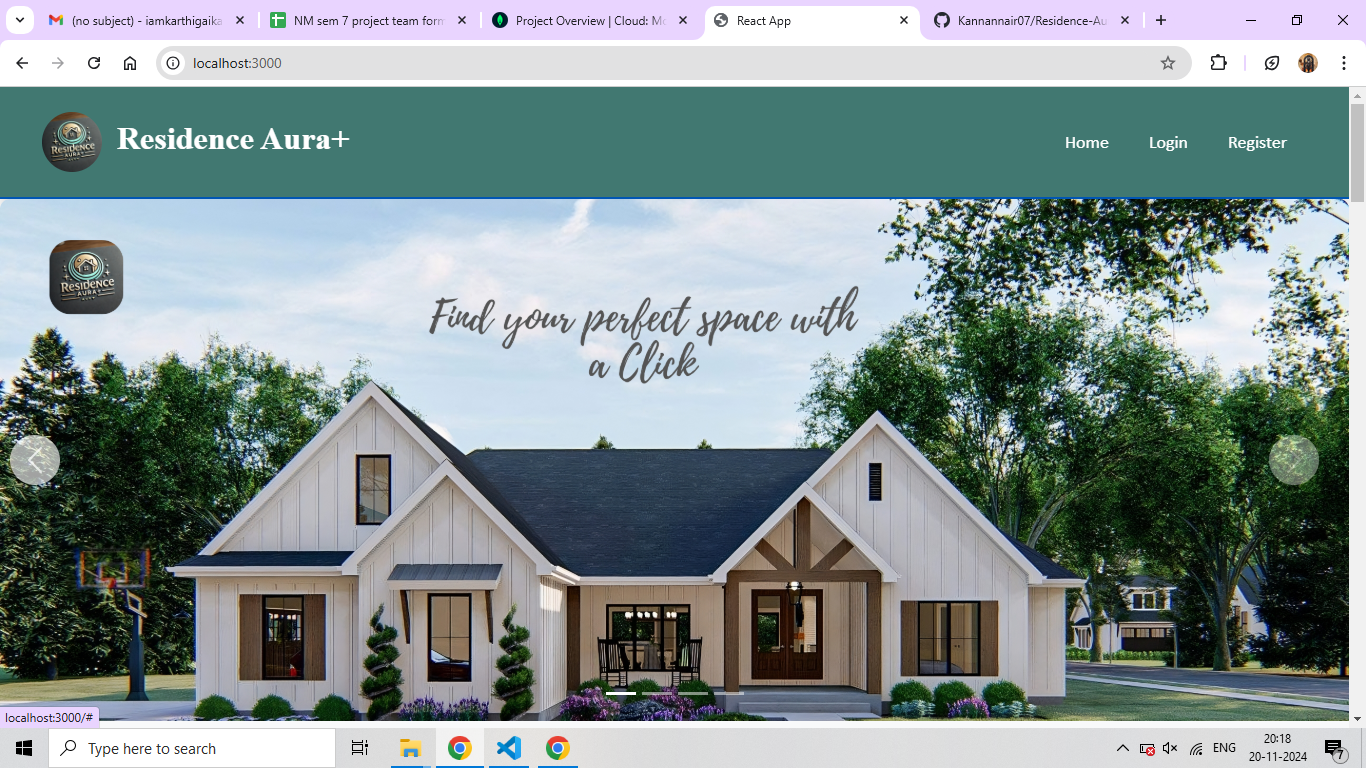
**Forgot Password:**

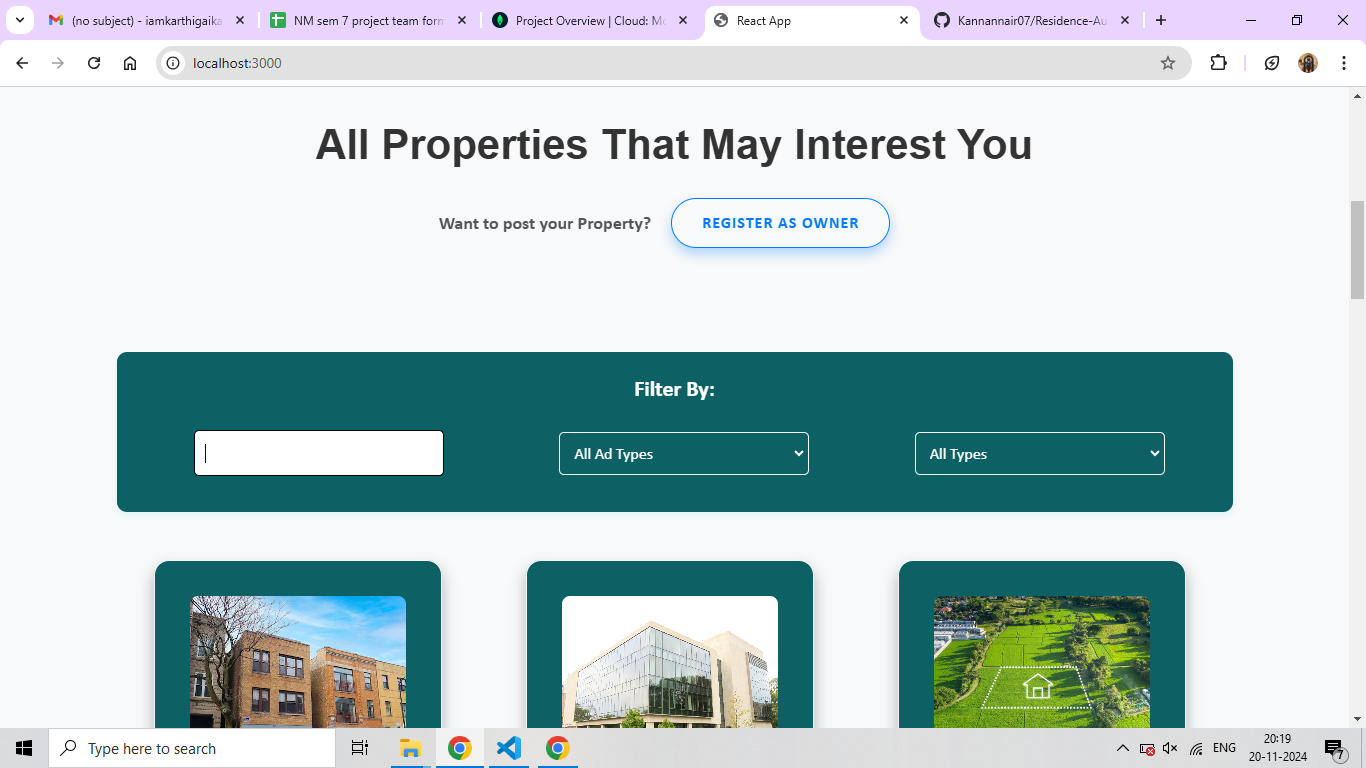
****

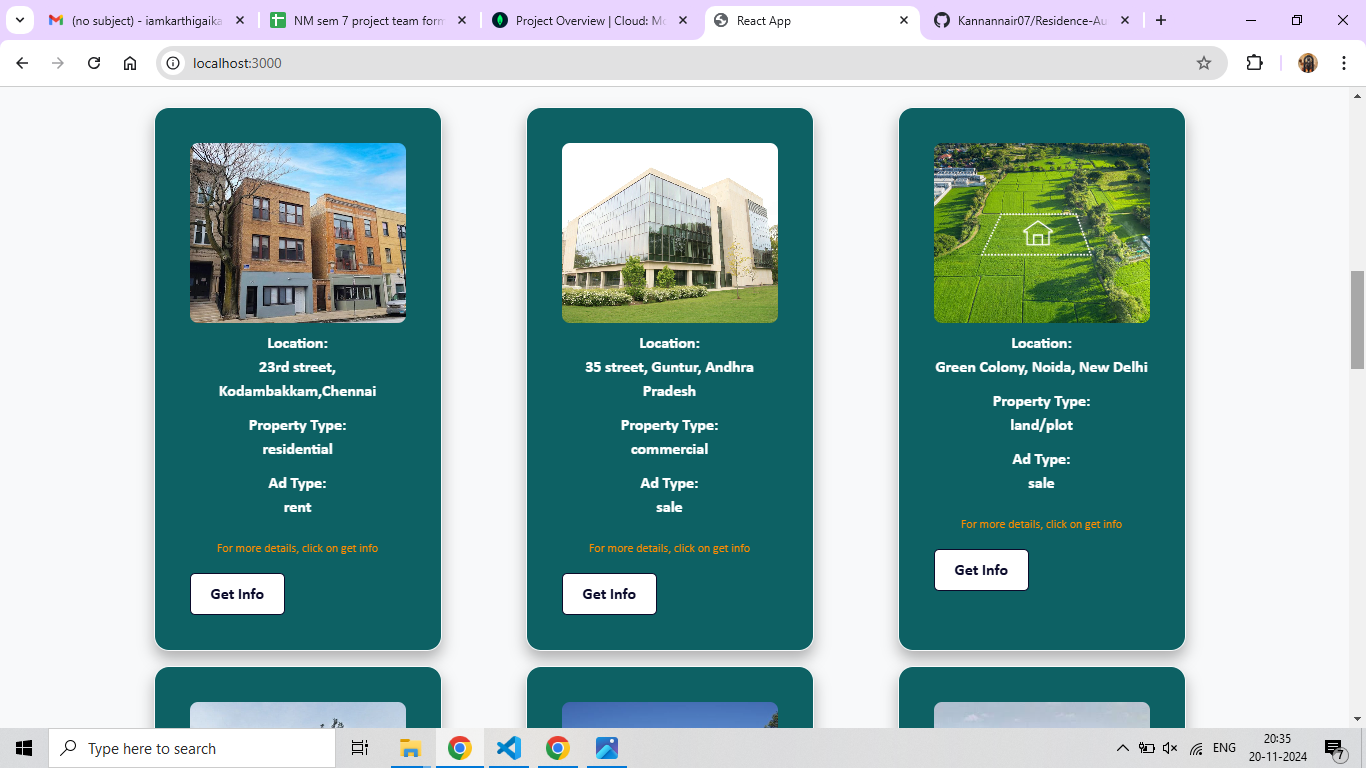
**Register:**

****

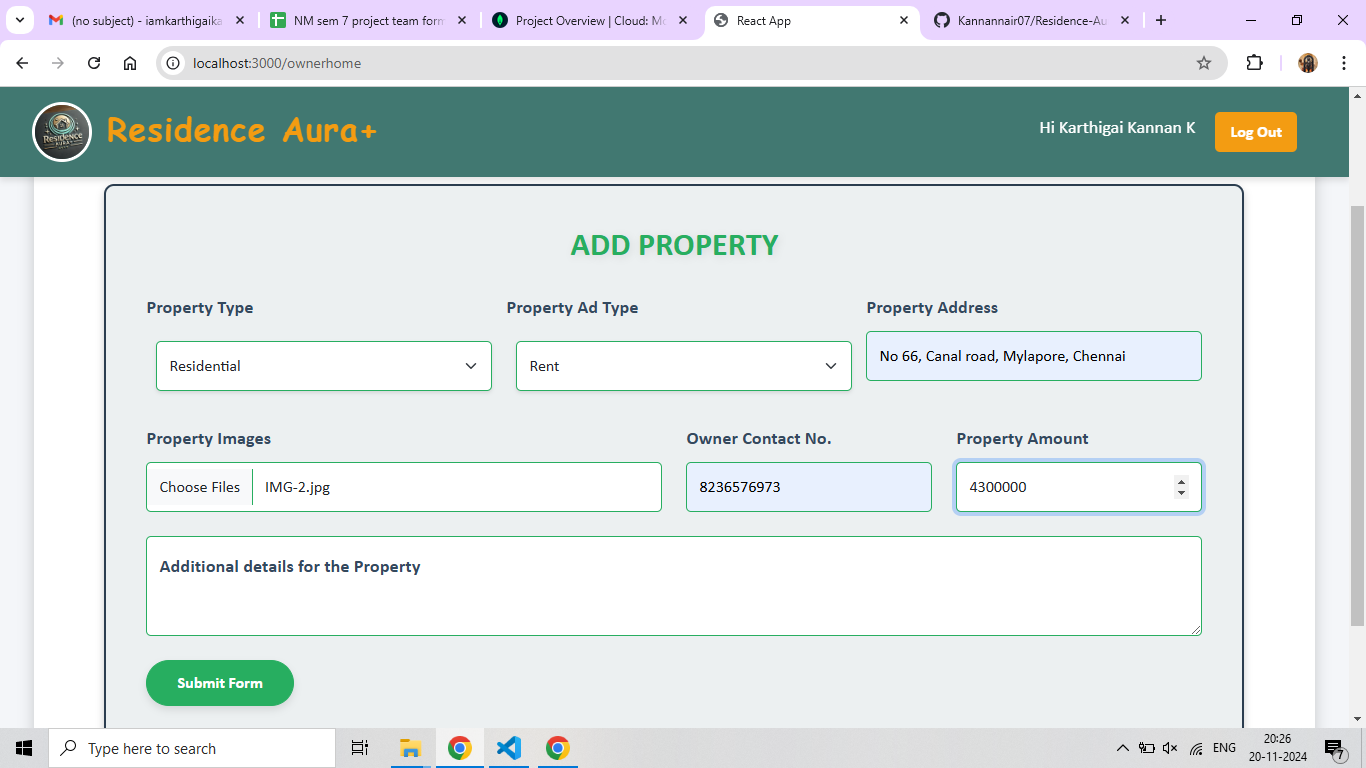
**Home Page:**

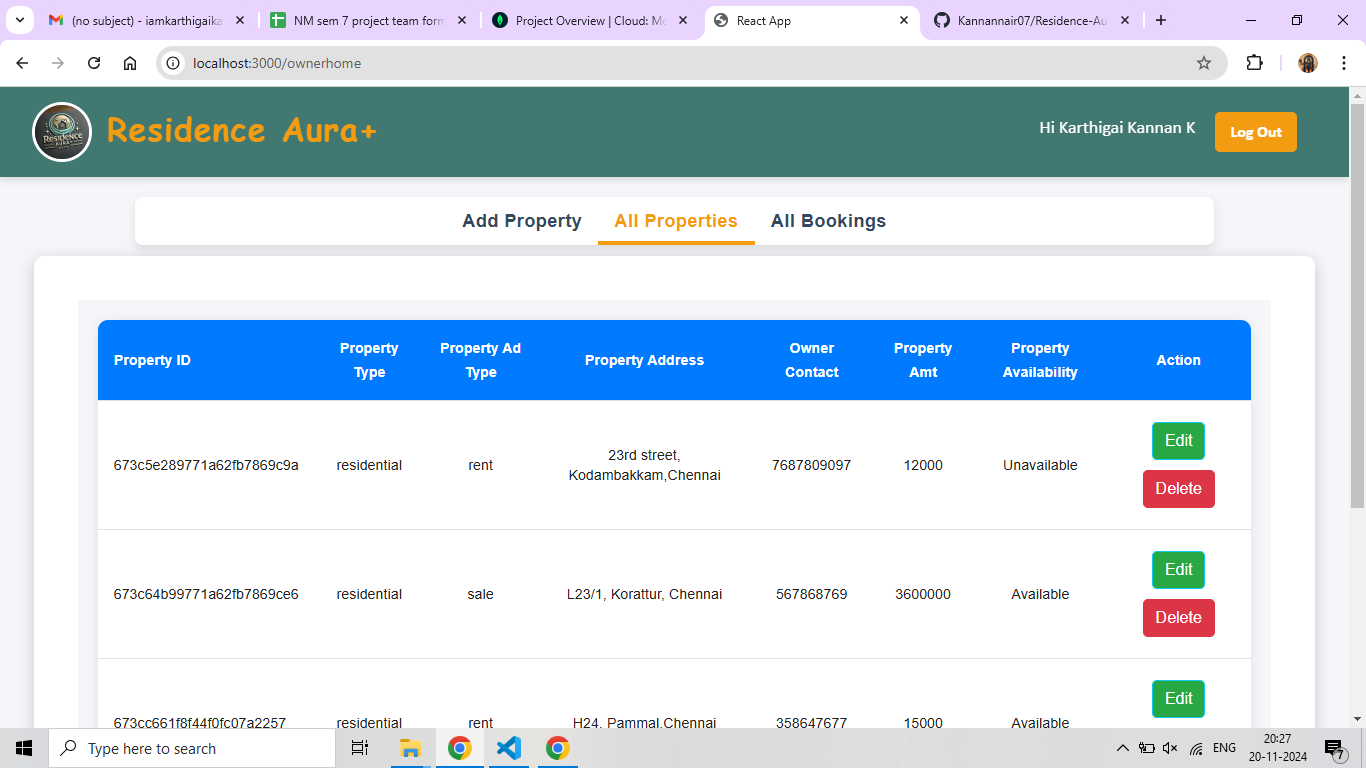
****

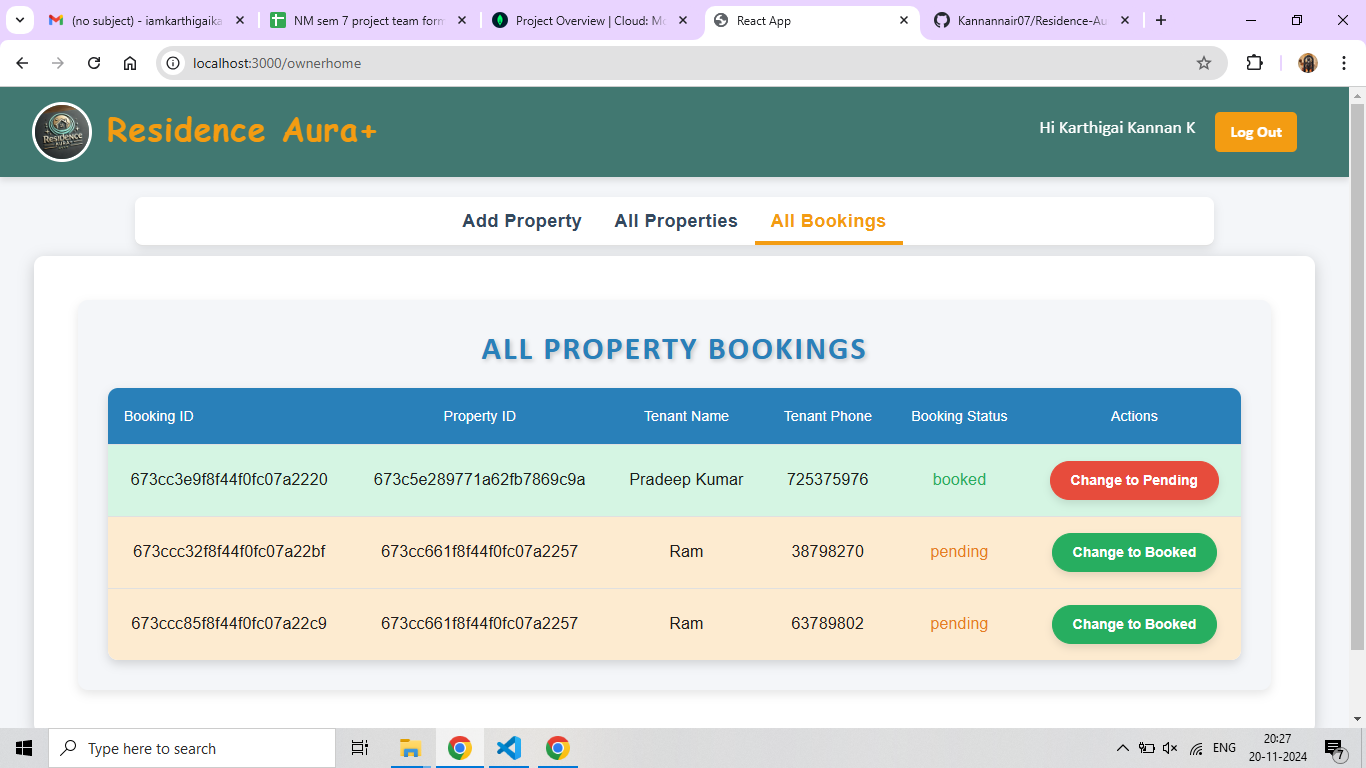
****

****

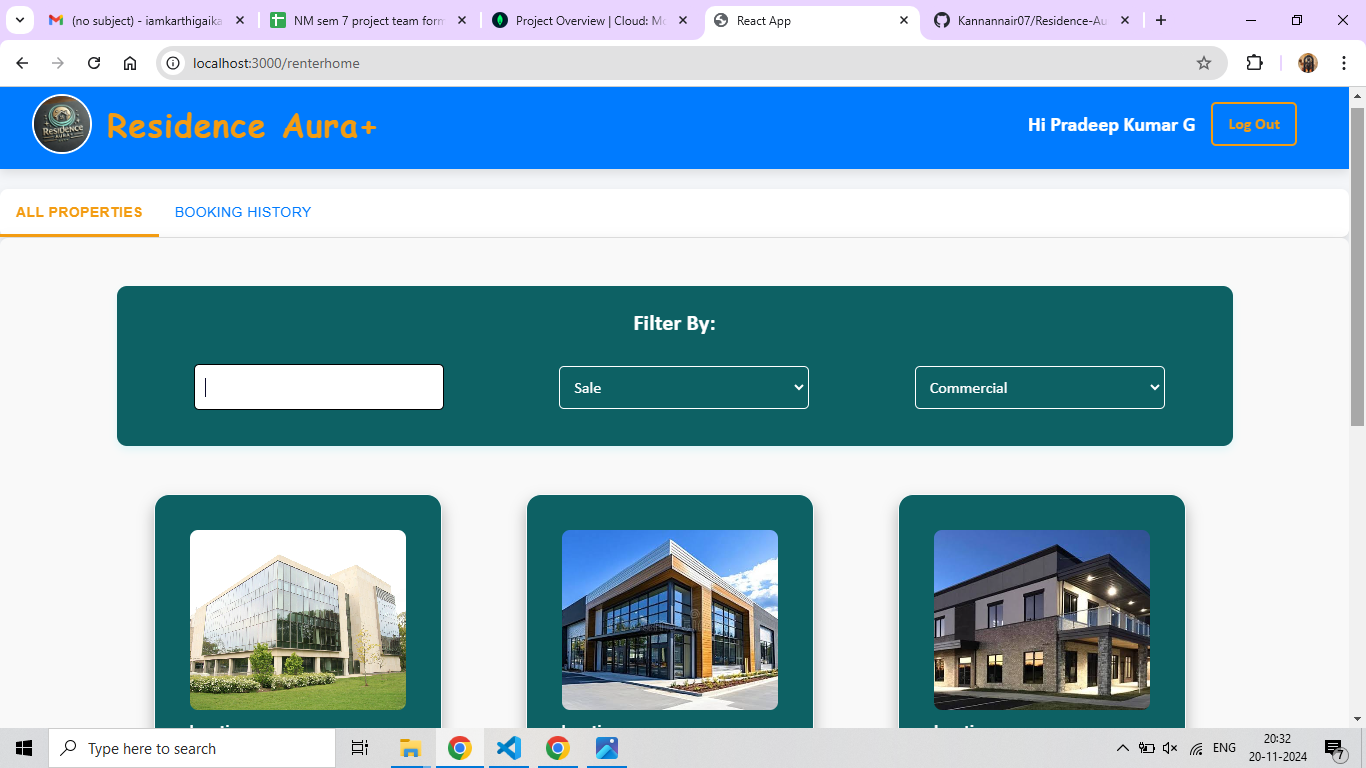
**Owner Page:**

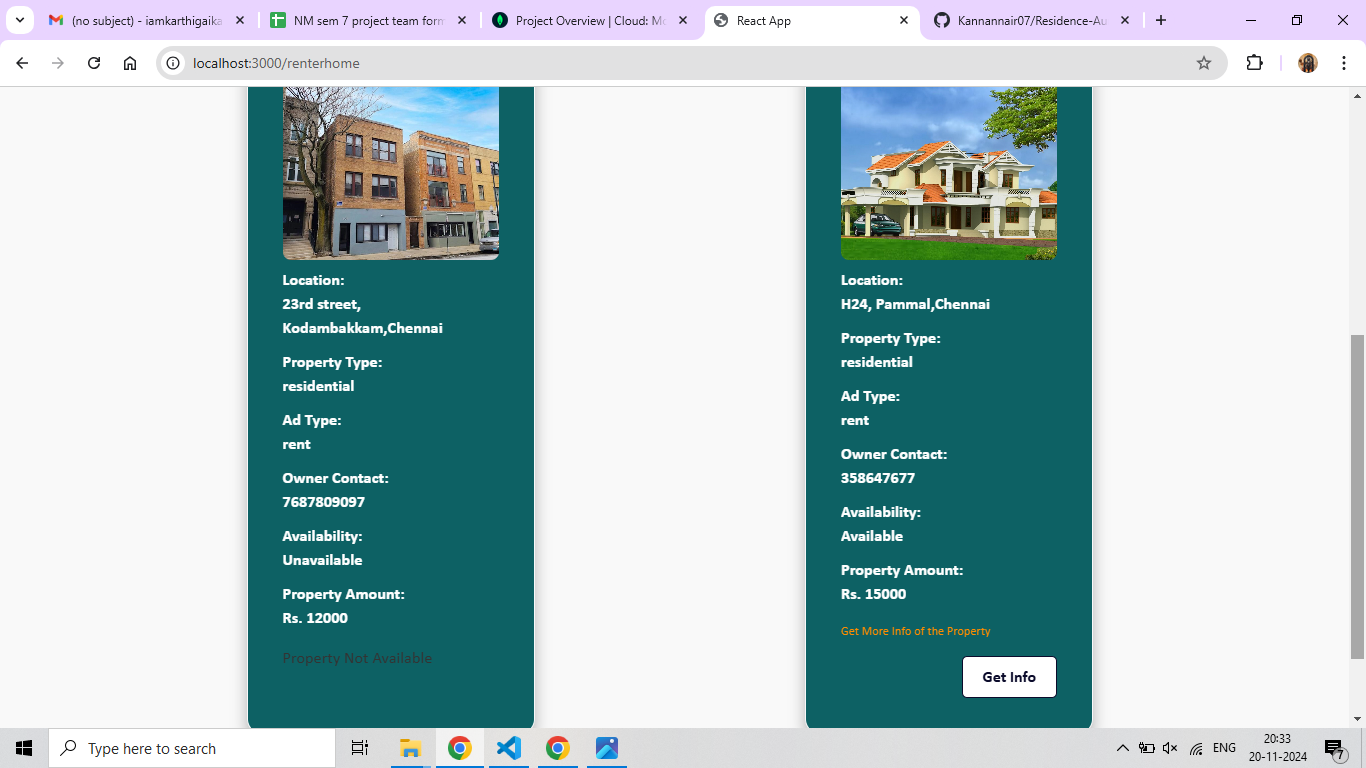
****

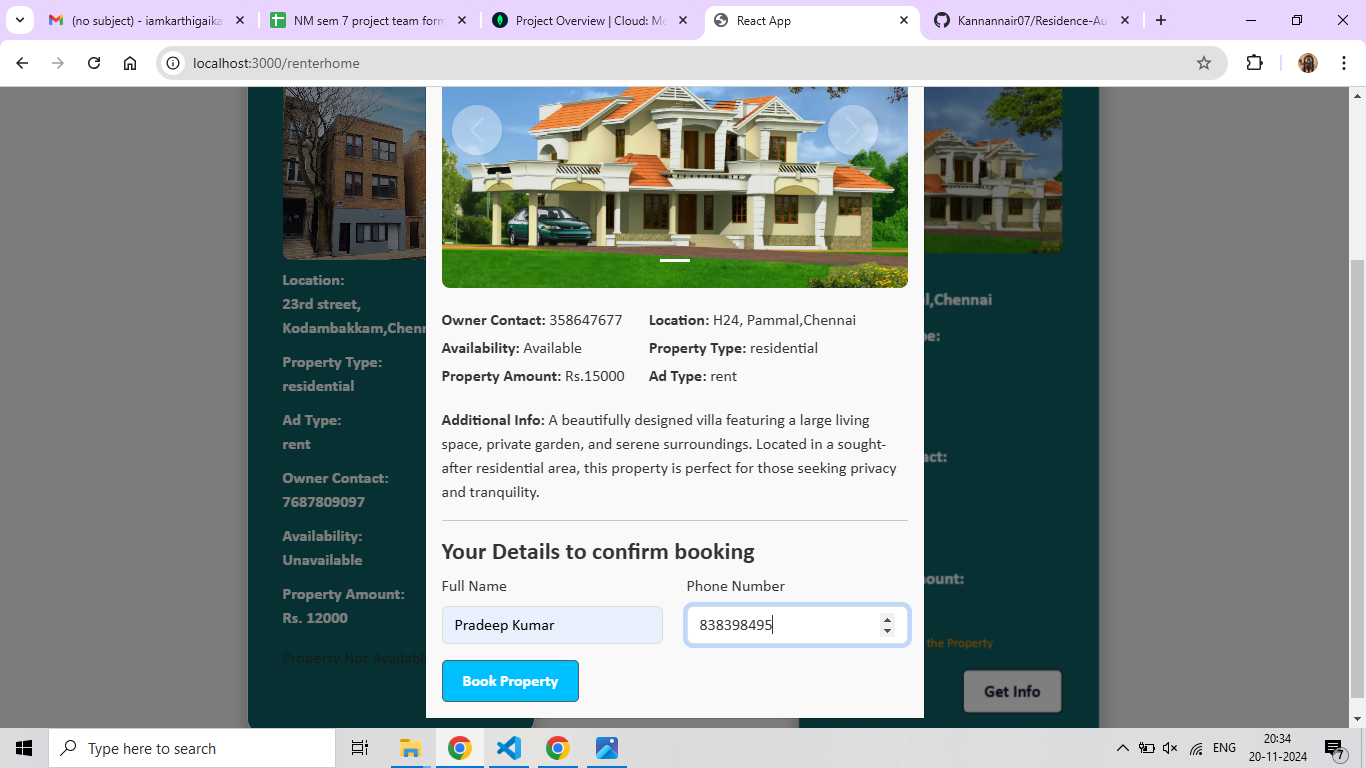
****

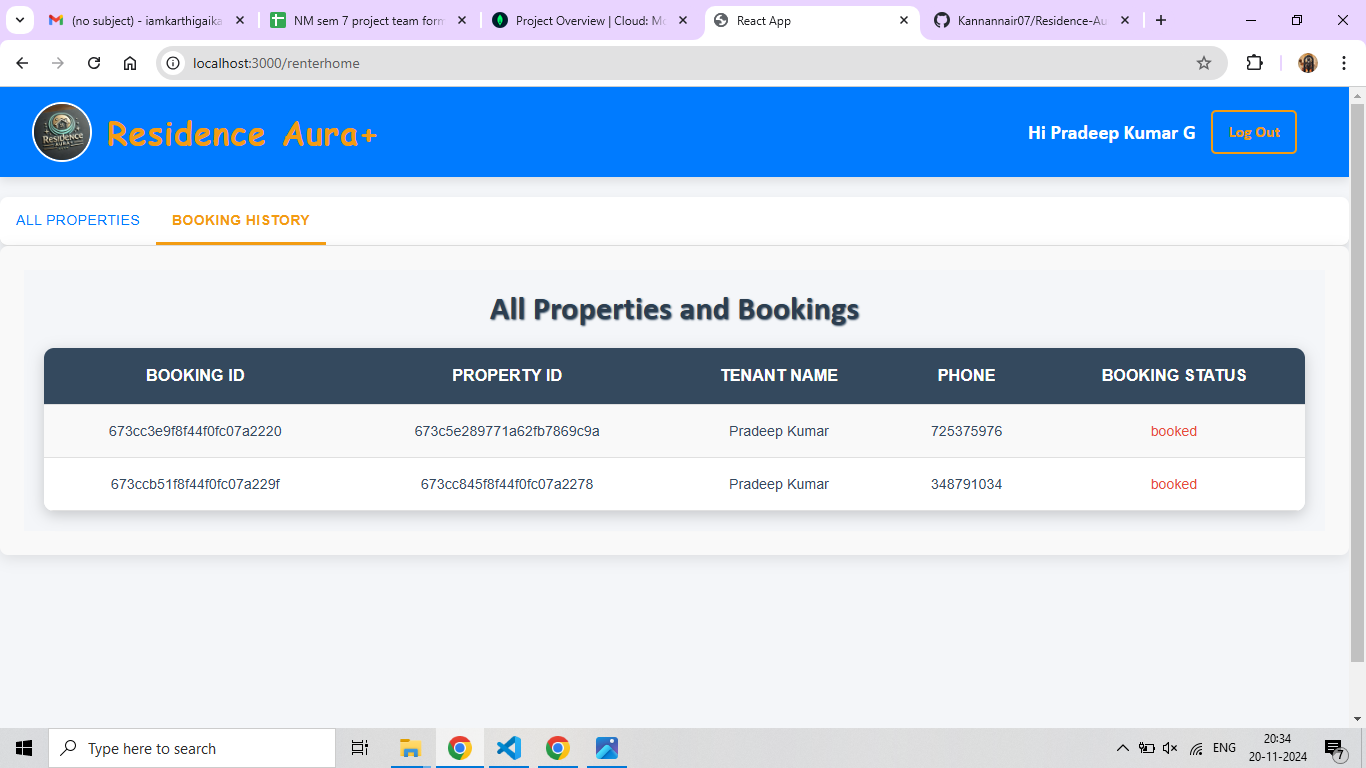
****

**Renter Page:**

****

****

****

****

**DEMO VIDEO LINK;** [**HOUSE RENT APP**](https://github.com/Kannannair07/Residence-Aura_Naan-Mudhalvan_Project)

**11. TESTING**

When developing a rental home application using the MERN stack (MongoDB, Express.js, React.js, Node.js), it's crucial to implement various testing strategies to ensure that the application works as expected and meets the requirements. Each part of the MERN stack has its own testing needs, and there are different tools and methodologies you can use for each layer of the stack. Below is a detailed breakdown of the testing strategies and the tools you can use for a rental home application.

### 1. Unit Testing

#### Objective:

* Ensure that individual units or components of your application (such as functions, classes, or React components) work as expected in isolation.

#### Tools:

* **Frontend (React.js)**:
  + **Jest**: A popular testing framework for JavaScript that works well with React. It provides utilities to mock functions and check if a component behaves as expected.
  + **React Testing Library**: This helps you test React components by rendering them and interacting with them as a user would (e.g., clicking buttons, entering input).
  + **Enzyme** (Alternative to React Testing Library): Another testing utility for React that allows you to shallow render components, simulate events, and make assertions on component behavior

#### Backend (Node.js/Express.js):

* **Jest** or **Mocha + Chai**: These tools allow you to test API endpoints, business logic, and database interaction in isolation. You can mock dependencies like database calls to test your functions without interacting with the actual database.

### 2. Integration Testing

#### Objective:

* Ensure that multiple components or services work together as expected. This includes interactions between the frontend and backend, as well as between backend services like database operations.

#### Tools:

* **Supertest**: A popular library for testing HTTP endpoints. It can be used to test how well the frontend interacts with the backend.
* Jest or Mocha + Chai for integration testing.
* **MongoDB Memory Server**: For testing the backend in isolation without affecting the real database. It creates an in-memory MongoDB instance to run tests.

### 3. End-to-End (E2E) Testing

#### Objective:

* Simulate real-world user scenarios and test the entire application stack from start to finish. E2E tests verify the user experience across all layers of the app.

#### Tools:

* **Cypress**: A popular end-to-end testing framework that allows you to interact with your app in a real browser and simulate user interactions. It supports features like automatic waiting, debugging, and screenshot capturing.
* **Selenium**: An alternative to Cypress, useful for testing in multiple browsers, but more complex to set up.
* **Playwright**: Another end-to-end testing framework similar to Cypress, but it allows for testing across multiple browsers (Chrome, Firefox, WebKit).

**4. Security Testing**

#### Objective:

Identify vulnerabilities and ensure that the application handles sensitive data securely.

#### Tools:

* **OWASP ZAP** (Zed Attack Proxy): A tool for finding security vulnerabilities in web applications. It can be used for penetration testing and scanning for common vulnerabilities such as XSS, SQL injection, and CSRF.
* **Burp Suite**: Another tool for security testing and vulnerability scanning.
* **Jest** (for testing API authentication): Ensure that unauthorized users cannot access protected routes.

**Summary of Testing Strategies and Tools for MERN Stack:**

1. **Unit Testing**:
   * **Frontend**: Jest, React Testing Library, Enzyme
   * **Backend**: Jest, Mocha + Chai, Supertest
2. **Integration Testing**:
   * Supertest, MongoDB Memory Server, Jest
3. **End-to-End (E2E) Testing**:
   * Cypress, Selenium, Playwright
4. **Security Testing**:
   * OWASP ZAP, Burp Suite, Jest for authentication

**13. FUTURE ENHANCEMENTS**

**Browser Compatibility:**

* Different browsers may interpret web elements slightly differently, leading to inconsistencies. Tests may pass on one browser but fail on another, requiring extra handling for cross-browser compatibility.

**Dynamic Content Handling:**

* Selenium struggles with dynamic or JavaScript-heavy websites. Elements may not be immediately available or interactable, leading to errors like ElementNotVisibleException or ElementNotFoundException. Proper waits and handling of AJAX requests are necessary.

**Flaky Tests:**

* Tests might intermittently pass or fail due to timing issues, especially in asynchronous environments. This can be caused by race conditions or slow network responses. Proper synchronization using WebDriverWait can mitigate this.

**WebDriver Crashes:**

* Occasionally, WebDriver instances may crash or fail to initiate, often due to mismatches between browser versions and the corresponding WebDriver (e.g., ChromeDriver and Chrome).

**Slow Execution:**

* Selenium's real-browser interactions make it slower compared to headless testing tools or mocking frameworks. This can be an issue when running large test suites or when tests need to run quickly in CI environments.

**Limited Mobile Testing Support:**

* Although Appium extends Selenium's capabilities to mobile testing, setting up and executing mobile automation tests can be complex and less stable than web automation.

**14.CONCLUSION**

In conclusion, the House rent application offers a comprehensive, scalable, and user-friendly platform for managing house rentals, built using the MERN stack (MongoDB, Express.js, React, and Node.js). The platform caters to tenants, landlords, and admins, providing essential features such as property listings, advanced search and filtering, real-time messaging, and an intuitive admin panel. Tenants can easily search for and inquire about rental properties, while landlords can list, manage, and update their properties with ease. The inclusion of a secure authentication system, responsive design, and user-friendly interface ensures a smooth experience across devices.

The use of MongoDB as a NoSQL database, along with Express and Node.js for the backend, ensures scalability and flexibility, enabling the application to grow with added features like payment integration or machine learning-based recommendations. Additionally, the platform is designed with security in mind, implementing JWT-based authentication and robust data validation to protect user information. The admin panel enhances control, offering powerful tools for content moderation, analytics, and user management, ensuring smooth operations.

**GITHUB LINK:** [**HOUSE RENT APP**](https://github.com/Kannannair07/Residence-Aura_Naan-Mudhalvan_Project)