Frontend Development with React.js

Project Documentation format

1. **Introduction**
   - **Project Title**: ***rentease***
   - **Team Members**:

   Sanyukta korade: backend development and testing

   Vaishnavi pawale: frontend development and UI UX

   Srushti keskar: database management

   Vaishnavi nichit: project architecture, documentation and reporting.

2. **Project Overview**

Purpose: This project is for a house rental application built on the **MERN stack** (MongoDB, Express, React, Node). Its primary purpose is to simplify and improve the rental process for tenants, owners, and administrators.

- The core of the project is to solve key problems in the rental market, such as:
- **Inaccurate listings** and inefficient search for tenants.
- **Manual and time-consuming application process** for owners.
- **Lack of transparency and trust** among all parties.
- The proposed **solution** is a unified digital platform featuring:
- A streamlined property search with accurate, up-to-date listings.
- A secure and simplified rental application process.
- An integrated communication system and a reliable payment gateway.

Goals: The house rent project is a digital solution built on the **MERN stack** (MongoDB, Express, React, and Node) designed to streamline the rental process. Its primary purpose is to simplify and improve the experience for tenants, property owners, and administrators.

The project's key **goal** is to create a more efficient, transparent, and trustworthy rental ecosystem. It achieves this by addressing common market problems, such as:

- **For Tenants:** Overcoming the frustration of inaccurate property listings and a complex application process.

- **For Owners:** Solving the challenges of inefficient applicant screening and manual property management.

Features:

For All Users

- **User Management:** Secure registration and login, profile management, and role-based access control.
- **Notifications:** Real-time alerts for important events like new messages, application status changes, or payment reminders.

For Tenants

- **Property Search & Filters:** Advanced search functionality to find properties by location, price, amenities, and number of bedrooms.
- **Detailed Listings:** Access to comprehensive property information, including high-quality photos, descriptions, and location on a map.
- **Rental Application:** A streamlined process for submitting rental applications and uploading necessary documents (e.g., ID, income proof).
- **Application Tracking:** A dashboard to monitor the real-time status of submitted applications.
- **Payment Gateway:** A secure system for making rent payments and viewing payment history.

For Owners

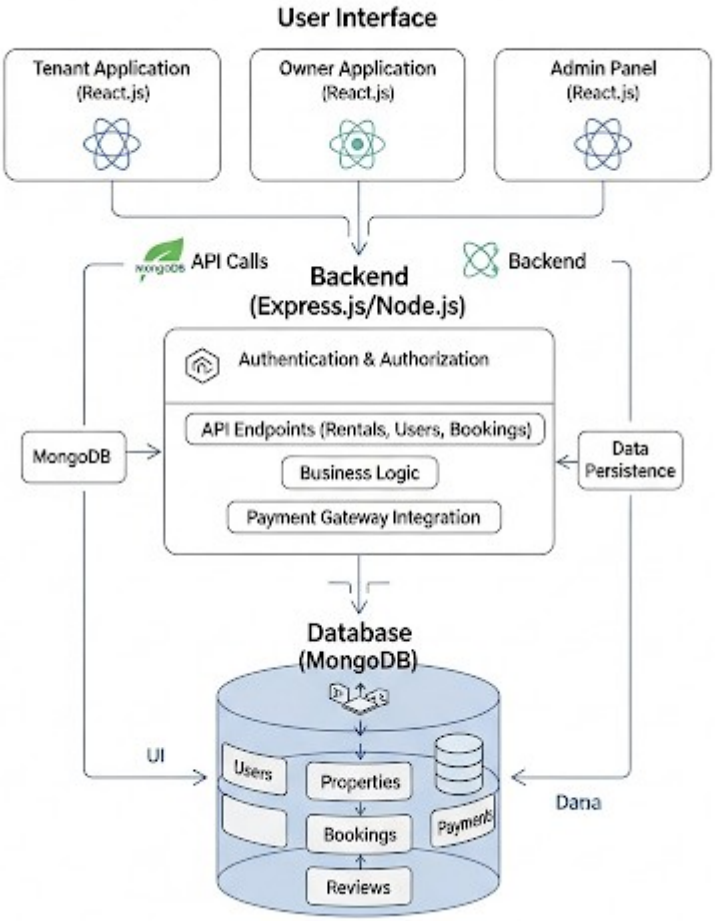
- **Property Listing Management:** A dashboard to create, edit, update, and delete property listings with ease.
- **Applicant Management:** Tools to review, approve, or reject rental applications.
- **Tenant Communication:** An in-app messaging system for direct communication with prospective and current tenants.
- **Payment Tracking:** A dashboard to track rent payments and generate financial reports.

3. **Architecture :**

The technical architecture of the house rental application is built on the **MERN stack**, which consists of four key components:

1. **Frontend (React.js):** This is the user interface that tenants, owners, and admins see and interact with. It's responsible for presenting property listings, application forms, and dashboards in a user-friendly way.
2. **Backend (Express.js & Node.js):** This is the server-side part of the application. It handles all the business logic, API calls, and user authentication. It acts as the bridge between the frontend and the database.

3. **Database (MongoDB):** This is where all the application data is stored. It's a flexible, non-relational database that holds information about user profiles, property listings, rental applications, and payments.



4. **Setup Instructions**

Prerequisites:

VS code :

Code editor installation for MERN stack project making , it runs the application.

Git and Github :

It is for code resources and references or store it securely. Github is used to collaborate with the project workspace repository .

Node.js and npm:

The runtime and package manager for JS (backend). to run backend JavaScript server efficiently download the Nodejs and install npm init.

Express.js:

It is used to handle routes and APIs easily. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

React.js:

This is to build dynamic frontend components and the useful and working user experience (UI). React.js making it easier to build responsive web applications. Command for frontend react.js installation –> npx create-react-app .

MongoDB:

Install MongoDB then used the MongoDB Atlas for GUI or cloud database to view or manage data, storing and analysing it for rentease application. MongoDB Atlas cloud-based database service provided by MongoDB that allows you to run, manage, and scale MongoDB databases.

Connecting database:

Use a MongoDB library like Mongoose to connect the Node.js server with the MongoDB database which created differently for rentease application. Later perform CRUD (Create, Read, Update, Delete) operations on it to work.

HTML, CSS, and JavaScript:

HTML for creating the structure of the application,

CSS for styling and designing frontend layout,

JavaScript for frontend logic and backend development.

Installation:

- VS code installation.
- folder setup = make two different folders for frontend and backend.
- confirm the download of node.js package. Node.js and npm.
- Git and github for code backup and references.

Frontend = react.js installation.

Backend =In backend folder

- cors
- express

- dotenv
- mongoose
- Nodemon
- Jsonwebtoken

Database =

- Configuring MongooDB.
- Import mongoose.

Folder Structure :

Frontend and backend folders

5. **Running the Application**
    - Provide commands to start the frontend server locally.
        - **Frontend**: npm start in the client directory.
        - **Backend**: npm start on go live server of system.

6. **Component Documentation:**

1. Frontend Components (React.js)

The frontend is built with React and is organized into reusable components for a modular and maintainable user interface.

- **Authentication Components:**
    - Login: A form for users to enter their credentials and access the application.
    - Register: A form for new users to create an account, including options for social sign-in.
    - Profile: A component allowing users to view and update their personal information and contact details.
- **Property Components:**
    - HomePage: The landing page with a search bar and a list of featured properties.
    - PropertyList: Displays a list of properties based on search results or filters.
    - PropertyCard: A reusable component representing a single property in the list, showing key details like photos and price.
    - PropertyDetail: The page showing all comprehensive information about a specific property, including a photo gallery, description, and amenities.
    - AddEditPropertyForm: A form used by owners to create a new property listing or edit an existing one.

2. Backend Components (Node.js / Express.js)

The backend is built with Node.js and Express.js and handles all business logic, API routing, and database interactions.

- **API Routes:**
  - /api/auth: Handles all user authentication endpoints like /register, /login, and /reset-password.
  - /api/properties: Manages property listings with endpoints for creating (POST), retrieving (GET), updating (PUT), and deleting (DELETE) listings. It includes a /search endpoint for filtering.
  - /api/applications: Manages rental applications with endpoints for submitting (POST), viewing (GET), and updating the status (PUT).
- **Models (Mongoose Schemas):**
  - User.js: Defines the schema for user data, including fields for name, email, password, and role (tenant, owner, admin).
  - Property.js: Defines the schema for a property, with fields like title, description, address, rent, photos, and a reference to the owner's ID.
- **Middleware:**
  - **Authentication Middleware:** A function to protect routes by verifying a JSON Web Token (JWT), ensuring only authenticated users can access certain resources.
  - **Validation Middleware:** Functions to validate incoming data from the frontend (e.g., ensuring a password meets length requirements before saving).

3. Data Flow & Integration

The **React frontend** sends requests (e.g., GET, POST) to the **Express.js backend**'s API endpoints. The Express server, in turn, uses **Mongoose** to interact with the **MongoDB database** to perform CRUD (Create, Read, Update, Delete) operations. The data is transferred between the frontend and backend in **JSON** format.

7. **State Management:**

1. Component-level State

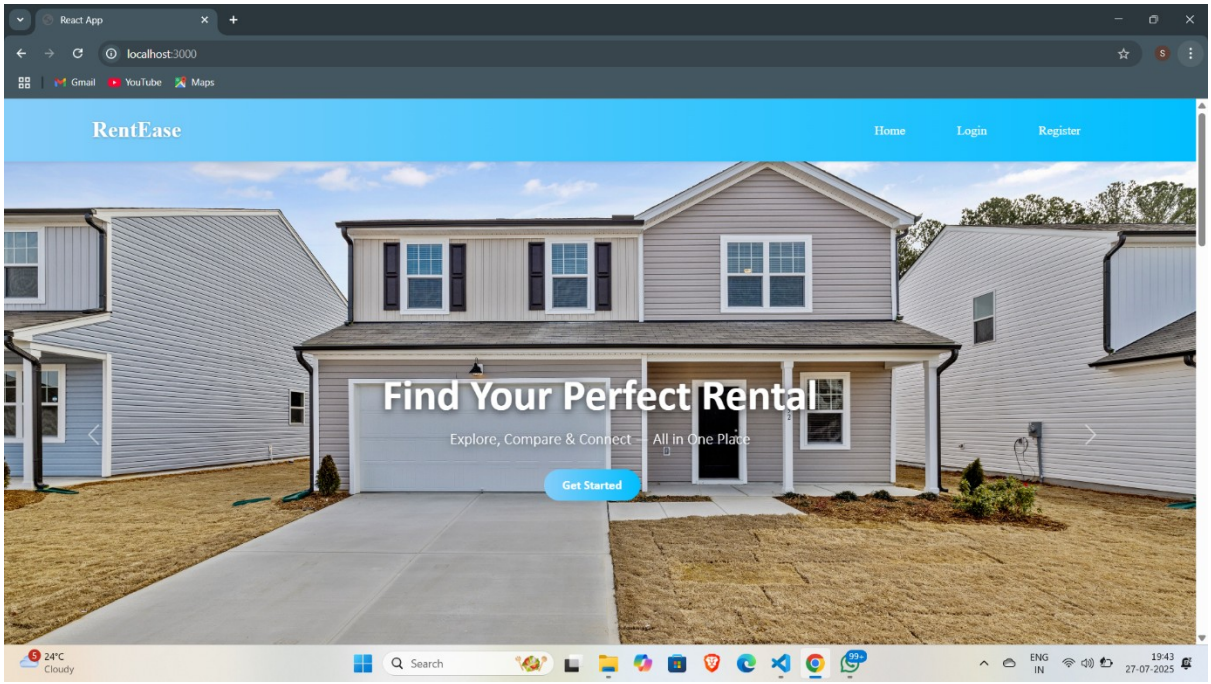
- **Purpose:** To manage temporary data that is only relevant to a single component.
- **Implementation:** This is handled using React's built-in useState and useEffect hooks.
- **Examples:** The text entered into a search bar, the visibility of a modal, or the loading state of a specific component's data.

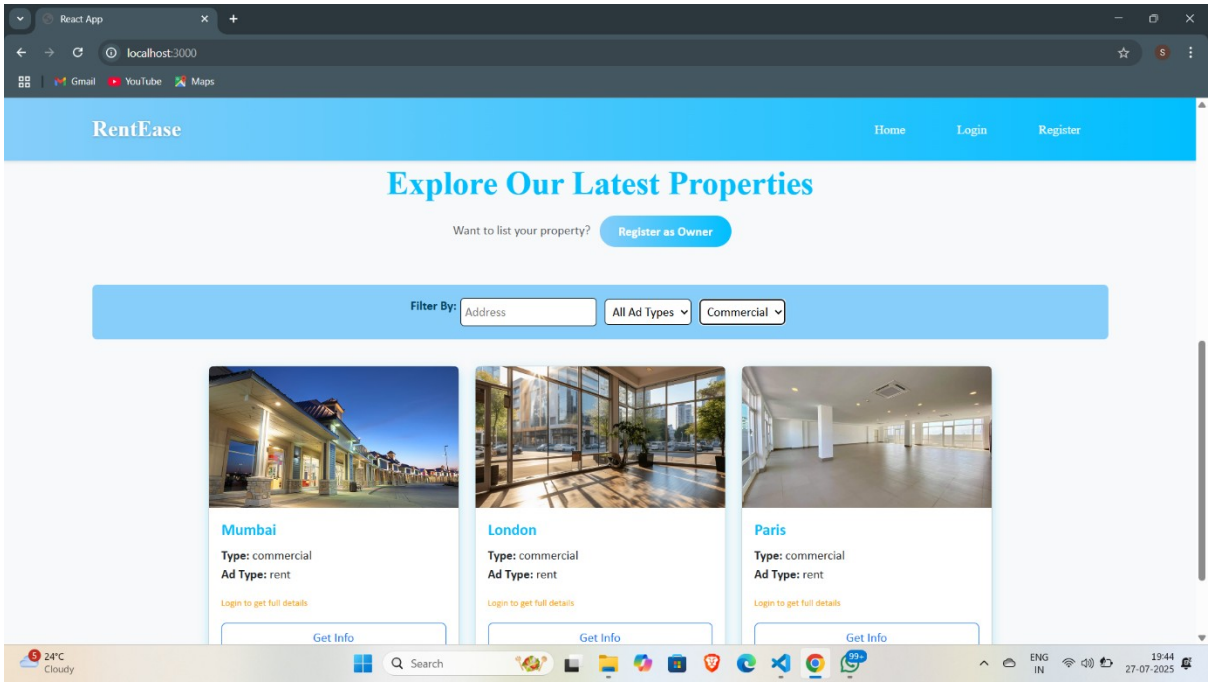
2. Global State

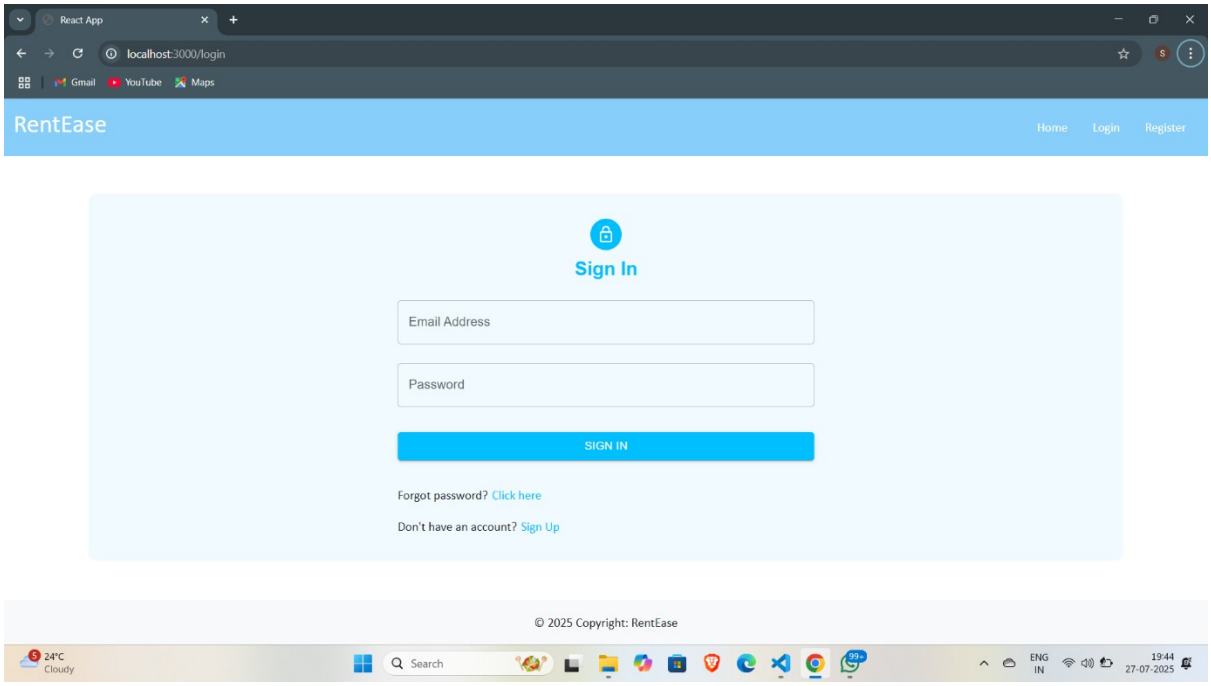
- **Purpose:** To manage data that needs to be accessed and shared across many different components in the application without having to pass it down manually through props ("prop drilling").
- **Implementation:** For a project of this complexity, the most robust solution would be **Redux Toolkit**. It provides a centralized, predictable state container and simplifies the management of complex data flows. The alternative for a simpler app would be React's **Context API**.

- **Global Data to be Managed:**
    - **Authentication State:** A central store would hold the user's login status (isLoggedIn), user role (isOwner, isTenant, isAdmin), and user profile information. This state would be accessible by components like the navigation bar and dashboards to show the correct content.
    - **Property Listings:** The list of all properties, including search results and filtered data, would be stored globally. This allows multiple components (e.g., the map view, the property list, and the search bar) to access the same data source.
    - **Rental Applications:** The list of pending, approved, or rejected applications for both owners and tenants would be managed in the global state, ensuring all relevant dashboards and notification components are always up to date.

8. **User Interface**
    - Provide screenshots or GIFs showcasing different UI features, such as pages, forms, or interactions.







9. **Styling**

1. Tailwind CSS

This is a **utility-first** framework that provides low-level CSS classes to build custom designs directly in your HTML or JSX. It's ideal for developers who want complete control over the design without being constrained by pre-defined components.

- **Pros:** Highly customizable, fast development, and excellent for unique, non-templated designs.
- **Cons:** Can lead to verbose code in JSX and requires a bit of a learning curve for beginners.

2. Bootstrap

Bootstrap is a classic **component-based** framework with a large collection of pre-built, responsive components like cards, forms, and modals. It's a great option for quick prototyping and ensuring a consistent, mobile-first design.

- **Pros:** Easy to learn, has a massive community, and is excellent for building an MVP quickly.
- **Cons:** Can result in a generic-looking design if not customized heavily.

11. **Testing**

Testing Scope:

- **Registration and Login:** Verify that users can register with email/password and social accounts (e.g., Google, LinkedIn). Ensure password reset and login functionality work correctly.
- **Profile Management:** Test that users can view, edit, and update their personal information, profile pictures, and contact details.
- **User Roles:** Verify that different permissions are enforced for each role (Tenant, Owner, Admin). For example, only Owners can list properties.

[**CRUD Operations (Create, Read, Update, Delete):** Test that Owners can successfully add new listings, view their properties, edit details, and remove listings.]

12. **Screenshots or Demo:**

Demo limk:
https://drive.google.com/file/d/1TGpD50ebdaJvKgyZLq9V7dZqc4KEPlc8/view?usp=drive_link

13. **Known Issues**

User Research and Requirements Gathering :

The house rental market is inefficient and frustrating. Tenants struggle to find accurate, up-to-date listings and face a complex, time-consuming application process. Owners have difficulty vetting

applicants and managing communication, while the entire process is often plagued by a lack of transparency and trust.

14. **Future Enhancements:**

1. Advanced Search and Personalization 🔍

- **AI-Powered Matching:** Implement a recommendation engine that suggests properties to tenants based on their search history, saved listings, and application data.
- **Neighborhood Insights:** Integrate with data sources to provide tenants with information on schools, crime rates, local amenities, and public transport options for each neighborhood.

2. Enhanced User Experience & Tools 🛠️

- **Integrated E-Signature:** Add a feature for digital signing of rental agreements and other documents, making the process fully paperless.
- **Tenant & Owner Review System:** Implement a transparent rating and review system where tenants can rate their experience with a property or landlord, and vice-versa.

3. Monetization & Business Expansion 📈

- **Premium Listings:** Offer a premium subscription for owners to boost their property listings, making them more visible to tenants.
- **In-App Insurance:** Partner with insurance providers to offer renters' insurance directly through the application.