Full Stack Development with MERN

Fast Rent - House Rental Application

Introduction

- Project Title : Fast Rent - House Rental Application
- Team Members

Name	Roles
1. Leo Franklin John - 311121104033	Frontend, Backend Dev
2. Shalwin Sanju - 311121104051	Frontend, Backend Dev
3. Melvina Rajvee - 311121104036	Frontend, Design
4. Anto Sherly - 311121104009	Frontend, Design

Project Overview

- Purpose :
    - The primary goal of FastRent is to provide a user-friendly platform where landlords can list their rental properties, and tenants can search, view, and rent homes seamlessly.
- Features :
    - User authentication and authorization.
    - Search and filter properties by location, price, and other criteria.
    - Property listing with image uploads.
    - Secure backend to handle user data.
    - Responsive frontend for an excellent user experience.

Architecture

- Frontend :
    - The frontend is built using React, styled with Material-UI, Ant Design, and Bootstrap.
    - Key libraries include: -
        - `react-router-dom`: For navigation.
        - `axios`: For API requests.
- Backend :
    - The backend uses Node.js with Express.js to provide RESTful APIs.

    - Key functionalities include:
        - Authentication using JWT.
        - Secure password encryption with bcrypt.js.
        - File handling with Multer for image uploads.
- Database :
    - The database is MongoDB, managed via Mongoose.
    - The schema includes collections for:
        - Users: Storing user details like name, email, and password.
        - Properties: Storing property details like title, location, price, and images.

Setup Instruction

- Prerequisites :
    - Ensure the following are installed: -
        - Node.js
        - MongoDB (local or cloud instance)
- Installation :
    - Clone the repository:
        - git clone https://github.com/shalwin04/FastRent.git
        - cd FastRent
    - Frontend Setup
        - cd frontend
        - npm install
    - Backend Setup
        - cd ../backend
        - npm install
    - Create a new .env file in the backend directory
        - PORT=5000
        - MONGO_URI=<your_mongodb_connection_string>
        - JWT_SECRET=<your_jwt_secret>

Folder Structure

- Frontend :
    - frontend/
    - ├── public/ # Static assets like HTML files and favicon
    - ├── src/ # Main application code
    - │ ├── images/ # Contains image assets
    - │ ├── modules/ # Organized by roles and shared components
    - │ │ ├── admin/ # Admin-related components and logic
    - │ │ ├── common/ # Shared utilities or components
    - │ │ └── user/

- │ │ ├── owner/ # Logic and components for property owners
- │ │ └── renter/ # Logic and components for renters
- │ ├── App.js # Application entry point
- │ └── index.js # React DOM renderer
- ├── package.json # Frontend dependencies
- └── README.md # Project description for the frontend

- Backend :
  - backend/
  - ├── models/ # Mongoose schemas
  - │ ├── User.js
  - │ ├── Property.js
  - ├── routes/ # API route handlers
  - │ ├── auth.js
  - │ ├── property.js
  - ├── controllers/ # Business logic
  - │ ├── authController.js
  - │ ├── propertyController.js
  - ├── index.js # Server entry point
  - └── package.json # Backend dependencies

Running the Application

- Start the Frontend
  - `cd frontend`
  - `npm start`

- Start the Backend
  - `cd backend`
  - `npm start`

  The application will be accessible at:

- Frontend: `http://localhost:3000`
- Backend: `http://localhost:5000`

API Documentation

- User Management
  - GET /getallusers - Fetch all users (auth required).
  - POST /register - User registration.
  - POST /login - User login.
  - POST /forgotpassword - Handle forgotten passwords.
  - POST /getuserdata - Fetch user data (auth required).
- Properties Management
  - GET /getallproperties - Fetch all properties (auth required).
  - POST /postproperty - Add a property (auth, file upload required).
  - DELETE /deleteproperty/:propertyid - Delete a property by ID (auth required).
  - PATCH /updateproperty/:propertyid - Update property details (auth, file upload required).
- Booking Management
  - GET /getallbookings - Fetch all bookings (auth required).
  - POST /handlebookingstatus - Update booking status (auth required).
  - POST /bookinghandle/:propertyid - Handle a booking for a property (auth required).
- Admin Features
  - POST /handlestatus - Handle user status (auth required).
- Eg Requests & Responses
  - REQUESTS FOR REG,
    - {
    - "name": "John Doe",
    - "email": "john@example.com",
    - "password": "password123"
    - }
  - RESPONSE FOR REG
    - { "message": "Registration successful!" }

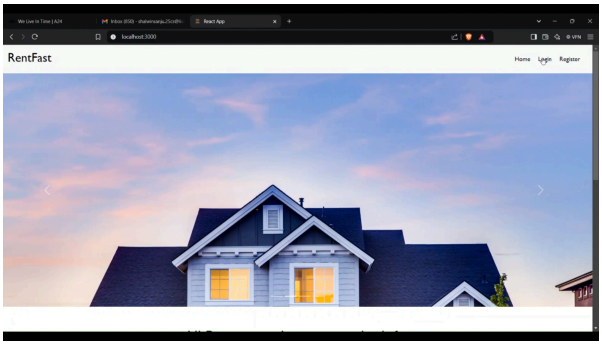
Authentication

- **Authentication**
  - **JWT-Based Authentication**:
    The project uses **JSON Web Tokens (JWT)** for authentication. After a user logs in or registers, a token is generated using a secret key (`process.env.JWT_KEY`) and sent to the client.

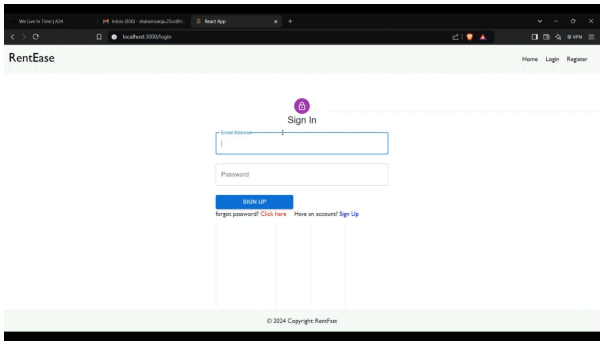
    - The client includes this token in the `Authorization` header for subsequent API requests.
- **Authorization**
    - **Middleware Validation**:
      The middleware checks for the `Authorization` header and validates the token using `jsonwebtoken`.
        - If valid, the user's `id` is extracted and appended to `req.body` for use in controllers.
        - If invalid or missing, appropriate error responses (`401` or `403`) are sent.
- **Session Management**
    - This implementation is stateless as tokens do not require server-side storage, making it scalable and efficient.

  This setup ensures secure access to protected routes based on user identity.

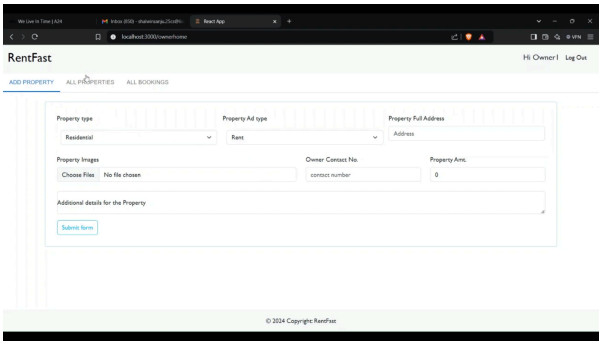
User Interface



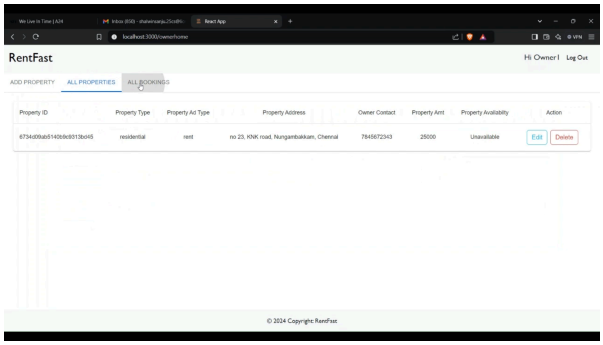
Home Page



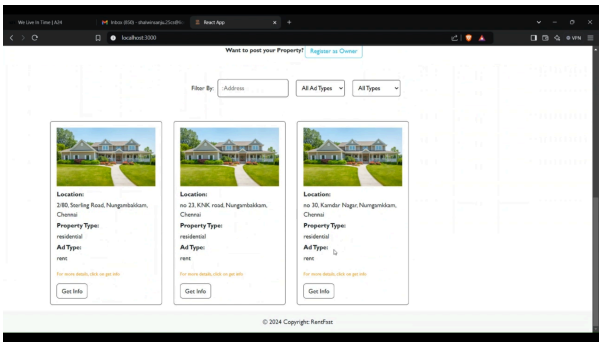
Login Page



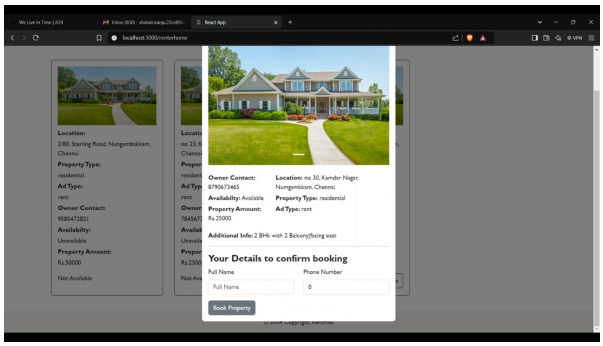
Owner Dashboard



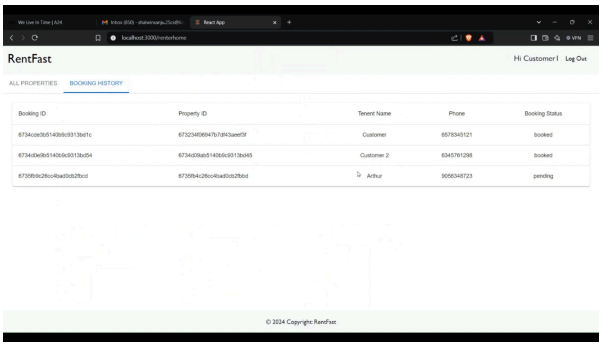
Owner Dashboard



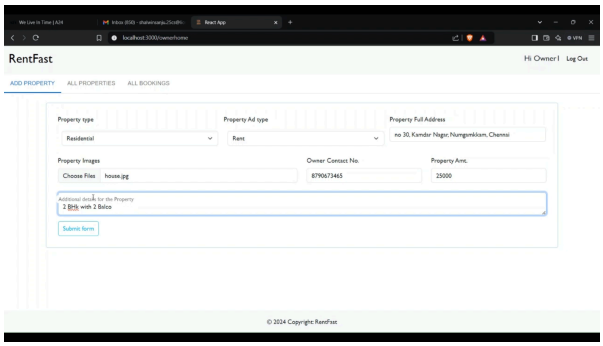
Property Listing in Home Page



Property Info



Customer Dashboard



New Property

Testing

The project employs the following testing tools and methodologies:

1. Frontend Testing

- **Tool Used:** React Testing Library
  - Ensures React components behave as expected.
  - Focus on testing UI interactions, component rendering, and state changes.
  - Example: Validating form submissions and error messages.

2. API Testing

- **Tool Used:** Postman
  - Used for testing backend API endpoints.
  - Verified request/response accuracy, status codes, and error handling.
  - Example: Testing endpoints like `/login`, `/postproperty`, and `/getallusers`.

This approach ensures functional reliability for both frontend and backend systems.

Demo

Demo Video link -
https://drive.google.com/file/d/1gGI4x7tZbHJuUblu8pQXc0yBIV7HtgS0/view?usp=drive_link