Naan Mudhalvan Project
MONGODB With MERN STACK

Project Title: House Rent App

Submitted By :

AKSHAYA G K – 212421243003

AMMU B – 212421243004

JECINTHA C -212421243019

VEDHA DHARSHINI K -212421243037

SREE SASTHA INSTITUTE OF

ENGINEERING AND TECHNOLOGY

CHEMBARAMBAKKAM

CHENNAI - 6000123

ANNA UNIVERSITY CHENNAI – 600025

NOV/DEC 2024

BONAFIDE CERTIFICATE

Certified that this report titled "HOUSE RENT APP" for the Naan mudhalvan project is a bonafide work of (AKSHAYA GK ,JECINTHA C, VEDHA DHARSHINI K, AMMU B) in MERN stack by Mongo DB -NM1016 who carried out the work under my supervision.

Certified further that to the best of my knowledge, the work reported here does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

FACULTY MENTOR **HEAD OF DEPARTMENT** **SPOC**

Submitted for the University Practical Examination held on ________

Internal Examiner **External Examiner**

Index

Abstract

The House Rent App is a comprehensive web application designed to streamline the property rental process for renters, property owners, and administrators. Developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), this application provides a centralized platform where renters can browse available properties, inquire about rental options, and secure bookings, while property owners can efficiently create, manage, and update their property listings. The application supports real-time updates on property availability, ensuring that users are always informed of the latest rental options.

A central feature of the House Rent App is its role-based access model, which distinguishes between three primary user roles—Renter, Owner, and Admin. Renters have access to a dashboard displaying available properties and a search and filter system to narrow down options based on price, location, and property type. They can also inquire about specific properties and manage their booking requests. Property owners, once approved by the admin, can create and manage property listings, including adding details like location, rental price, property type, and availability status. They can respond to inquiries and update booking statuses, ensuring effective property management.

The admin role is integral to the platform's governance and security. Admins are responsible for validating new owner accounts, overseeing property listings, and monitoring platform activities to maintain a safe and well-regulated environment. They have the authority to review, approve, or deny owner applications and resolve any user-reported issues, thereby ensuring the platform's quality and trustworthiness.

Security is a critical component of the app, implemented through JSON Web Tokens (JWT) for secure user authentication and session management. This ensures that only authorized users can access specific functionalities based on their roles. Additional technologies like Bootstrap, Material UI, and Ant Design are used to create a responsive, aesthetically pleasing user interface across devices, while MongoDB's NoSQL database structure supports efficient data storage and retrieval, enabling quick access to property details, inquiries, and bookings.

Future enhancements are planned to expand the app's functionality and improve user experience. Proposed features include mobile compatibility through a dedicated mobile app, advanced filtering options that allow renters to search based on amenities and proximity to landmarks, integrated secure payment options for booking confirmations, and an in-app messaging system for direct communication between renters and property owners. These enhancements aim to increase the app's accessibility, usability, and appeal in the property rental market, making the House Rent App a versatile and valuable tool for users and property managers alike.

CHAPTER 1. Project Overview

The House Rent App is a MERN-based web application designed to simplify the property rental process for both renters and property owners. This platform allows renters to browse property listings, inquire about rental options, and secure rentals with ease. On the other hand, property owners can manage their listings, respond to inquiries, and track bookings efficiently.

The app introduces an admin panel to ensure a smooth user experience, offering oversight and governance over user activities and platform security. Admins can validate owner accounts, monitor and approve property listings, and resolve any issues to maintain a high-quality and secure platform for everyone involved.

The main goal of the House Rent App is to create a centralized space where renters can quickly find suitable properties and property owners can efficiently manage their rental listings and interactions with potential tenants. By focusing on simplifying the rental process and incorporating essential features like real-time updates, advanced filtering, and secure user authentication, the app enhances the rental experience for all users.

The House Rent App also aims to contribute to the growth of the online property rental market by providing a modern and easy-to-use solution for property management and rental transactions.

Objectives

The primary objectives of the House Rent App are to:

1. Creating a Unified Platform: Integrating renters and property owners in a single application where both parties can seamlessly interact.
2. Facilitating Property Management: Allowing property owners to easily add, edit, and remove listings, respond to inquiries, and track their rental activity.
3. Simplifying Search for Renters: Renters can browse properties with ease and filter based on preferences like rent range, property type, and location.
4. Ensuring Platform Security: With admin oversight, the application ensures a safe user experience by monitoring activities, validating owner accounts, and managing user permissions.
5. Improving Communication: Enabling direct communication channels for renters and owners to discuss property details, availability, and booking terms.

CHAPTER 2. Technology Stack & System Requirements

The House Rent App is built using the MERN stack, a powerful combination of technologies that enable the development of dynamic, scalable web applications. The core components of the stack include:

- **MongoDB**: A NoSQL database used for storing user information, property listings, inquiries, and booking data.
- **Express.js**: A flexible backend framework that powers the API and handles server-side logic for the application.
- **React.js**: A frontend framework used to build a dynamic, responsive user interface for seamless user interaction.
- **Node.js**: A runtime environment that executes backend JavaScript and handles server requests efficiently.

In addition to the MERN stack, several other technologies are utilized to enhance the user experience and application security:

- **Bootstrap, Material UI, and Ant Design**: CSS frameworks used to design responsive and aesthetically pleasing user interfaces.
- **JSON Web Token (JWT)**: For secure user authentication and session management, ensuring that user data is protected.

System Requirements:

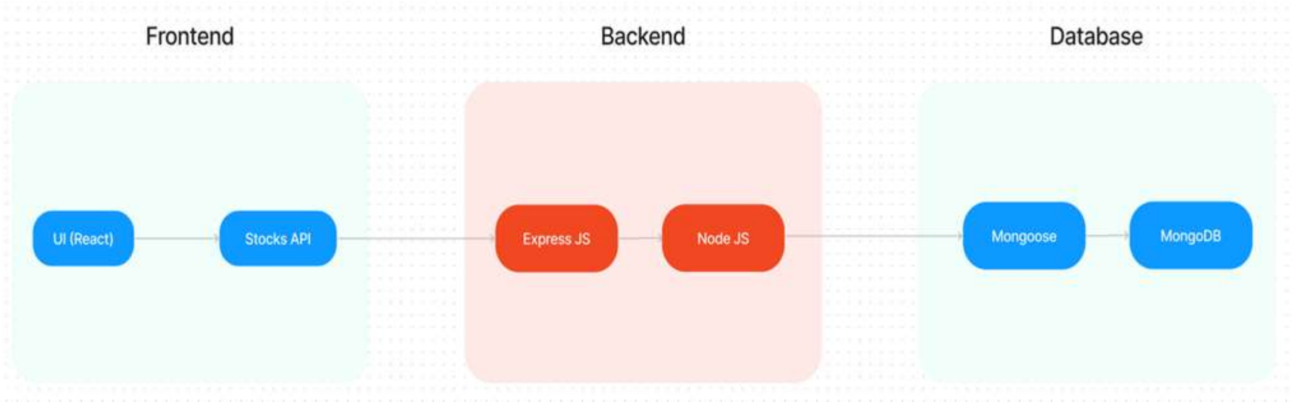
To run the House Rent App, the following hardware and software are required:

- **Hardware**:
  - Windows 8 or higher machine with a stable internet connection (30 Mbps recommended).
- **Software**:
  - **Node.js**: The latest version of Node.js should be installed.
  - **MongoDB Community Server**: Required for database management.
  - Two web browsers (for testing purposes) are recommended for cross-browser compatibility.

CHAPTER 3. Project Architecture

The application follows a modular architecture, split between frontend (client-side) and backend (server-side) components:

1. **Frontend (Client-Side)**:
   - Built with React.js for dynamic rendering, creating an intuitive and interactive experience for users.
   - CSS frameworks like Bootstrap, Material UI, and Ant Design are used to style components, ensuring consistency in design.
   - Communicates with backend services using API endpoints, enabling real-time data handling and smooth user interactions.
2. **Backend (Server-Side)**:
   - Express.js handles the core API and server functions, managing routes and ensuring that each request is processed efficiently.
   - MongoDB serves as the database, storing information related to users, properties, and booking transactions.
   - JWT tokens secure user authentication and authorization, preventing unauthorized access.
3. **Database (MongoDB)**:
   - MongoDB collections store all app-related data, including user profiles, property details, inquiries, and bookings.
   - The database is designed for optimal retrieval, supporting fast queries and handling real-time updates efficiently.



CHAPTER 4. Installation and Setup

Prerequisites

1. Install **Node.js** and **npm**.
2. Install **MongoDB Community Server**.

Step-by-Step Setup

1. **Clone the Repository**

   https://github.com/AkshayaGopalakrishnan/HOUSE-RENT-APP-USING-MERN.git

2. **Backend Setup**:
   a. Navigate to the backend folder and install dependencies:
      - cd house-rent-app/code/backend
      - npm install

   b. Create an .env file with MongoDB connection and JWT key.

   c. Modify the MongoDB connection string in connect.js in config folder

   d. Start the backend server:
      - npm start

3. **Frontend Setup**:
   a. Navigate to the frontend folder and install dependencies:
      - cd house-rent-app/code/frontend
      - npm install

   b. Start the frontend server:
      - npm start

4. **Access the App**:
   a. Frontend: http://localhost:3000

   b. Backend: http://localhost:8000

CHAPTER 5. FOLDER STRUCTURE

The folder structure of the House Rent App is organized to separate frontend and backend components, allowing for a modular and maintainable codebase.

```
house-rent-app/
├── frontend/             # Frontend (client-side) application folder
│   ├── public/           # Static files accessible to users
│   │   ├── index.html      # Main HTML file for the React app
│   │   └── assets/        # Images, icons, and other static assets
│   ├── src/              # Source code for the React app
│   │   ├── components/     # Reusable React components
│   │   │   ├── Navbar.js   # Navigation bar component
│   │   │   ├── Footer.js   # Footer component
│   │   │   └── PropertyCard.js # Component for displaying property details
│   │   ├── pages/         # Page-level components (e.g., Home, Login)
│   │   ├── context/       # Context API setup for global state management
│   │   ├── App.js         # Main App component with routing
│   │   └── index.js       # Entry point for the React application
│   ├── package.json       # Project metadata and dependencies for frontend
│   └── .env              # Environment variables for frontend
```

```
├── backend/              # Backend (server-side) application folder
│   ├── config/           # Environment and database configurations
│   │   └── connect.js      # MongoDB connection setup
│   ├── controllers/       # Logic for handling API requests
│   │   ├── authController.js # Handles user authentication
│   │   ├── propertyController.js # CRUD operations for properties
│   │   └── bookingController.js  # Handles booking requests and status
│   ├── models/            # Mongoose schemas for MongoDB collections
│   │   ├── User.js        # Schema for user details (Renter, Owner, Admin)
│   │   ├── Property.js     # Schema for property listings
│   │   └── Booking.js      # Schema for booking details and status
│   ├── routes/            # API routes for different functionalities
│   │   ├── authRoutes.js   # Routes for authentication (login, signup)
│   │   ├── propertyRoutes.js # Routes for property management
│   │   └── bookingRoutes.js  # Routes for booking inquiries and updates
│   ├── middleware/        # Middleware functions for request validation
│   │   └── authMiddleware.js # JWT-based authentication middleware
│   ├── .env                 # Environment variables for backend (MongoDB URI, JWT
secret)
│   ├── server.js          # Entry point for the Express server
│   └── package.json       # Project metadata and dependencies for backend
└── README.md             # Documentation and instructions for the project
```

CHAPTER 6. Workflow and Usage

User Roles and Functionalities

The House Rent App includes three primary user roles: Renter, Owner, and Admin. Each role has specific responsibilities and functionalities to ensure smooth operation of the platform.

1. Renter (Tenant):

- **Account Creation and Login:** Renters create an account and log in using their email and password.
- **Dashboard Access:** Upon login, they are shown a list of available properties in their dashboard.
- **Property Details:** By clicking on the "Get Info" button for a property, renters can view detailed information about the property and the owner. A form will be generated for the renter to submit their details to the property owner.
- **Booking Status:** Renters can check the status of their booking in the "Booking" section, where the status will initially be displayed as "pending." The owner will later update the status (approve or reject).

2. Owner:

- **Admin Approval:** Owners must first receive approval from the admin to create an owner account.
- **CRUD Operations for Properties:** Once approved, owners can perform all CRUD (Create, Read, Update, Delete) operations for their property listings. They can add, edit, or remove properties as needed.
- **Status and Availability Management:** Owners can update the status and availability of their properties (e.g., available, booked, unavailable) to reflect current conditions.

3. Admin:

- **Owner Approval:** Admins review and approve legitimate users as owners. Only after approval can an owner add and manage properties.
- **User Monitoring:** Admins monitor all user activities on the platform, ensuring compliance with platform rules.
- **Enforcing Policies:** Admins implement and enforce platform policies, terms of service, and privacy regulations to ensure a secure and trustworthy environment for all users.

CHAPTER 7. Testing

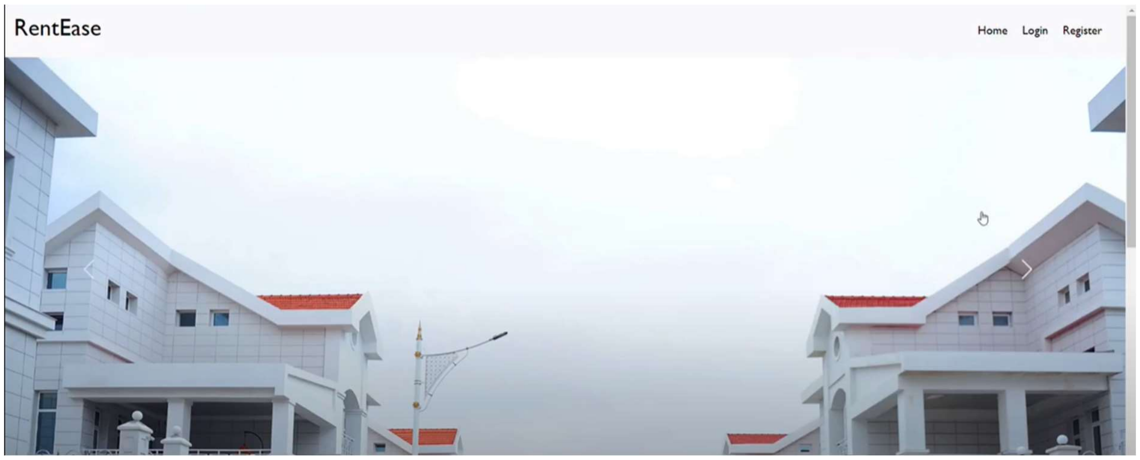
Manual testing was conducted to ensure that the House Rent App functions as expected and provides a seamless experience for users. The focus of manual testing included:

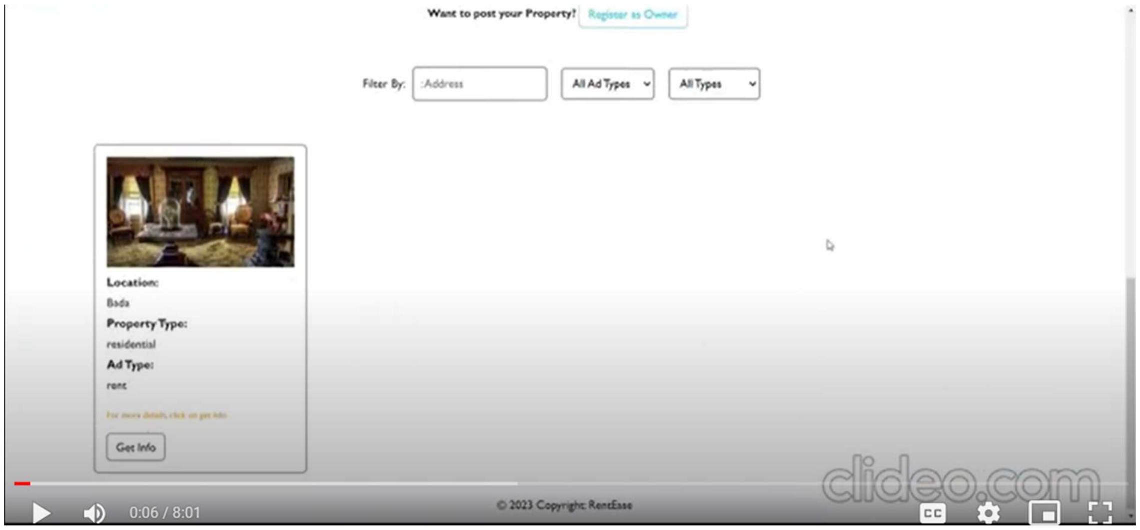
1. **User Registration and Login:**
    - Tested the registration and login functionality for both renters and property owners, ensuring that the correct validation messages appear for invalid inputs (e.g., empty fields, incorrect credentials).Verified that only authorized users (with admin approval) could register as owners.
2. **Property Listings:**
    - Verified that property owners could add new listings, edit existing ones, and delete properties successfully. Checked if property details such as location, rent, type, and availability were correctly displayed on the frontend.
3. **Search and Filter Functionality:**
    - Tested the search and filter options to ensure that properties could be filtered by various criteria like location, rent range, and property type. Ensured that applying filters resulted in relevant property listings being displayed.
4. **Booking and Inquiry System:**
    - Tested the inquiry and booking system by submitting inquiries as renters and confirming that property owners could receive and respond to these inquiries.Verified that the booking status correctly changes from "pending" to "confirmed" or "rejected" by the owner.
5. **Admin Functionality:**
    - Checked if the admin could approve owner accounts and ensure that only authorized owners could add or manage listings.Verified the admin's ability to monitor user activities and review owner registrations.
6. **Responsiveness:**
    - Tested the user interface on various devices (desktop, tablet, mobile) to ensure that the layout adapts appropriately to different screen sizes and that key functionalities remain accessible.
7. **Error Handling:**
    - Tested invalid inputs and error scenarios (e.g., incorrect property details, unauthorized access) to ensure that the system provides appropriate error messages and does not crash or behave unexpectedly.
8. **Real-Time Updates:**
    - Verified that property availability and booking statuses were updated in real-time and reflected correctly on both the renter's and owner's dashboards.

CHAPTER 8. Project Implementation & Execution

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one's provided below.

- **Landing page:**



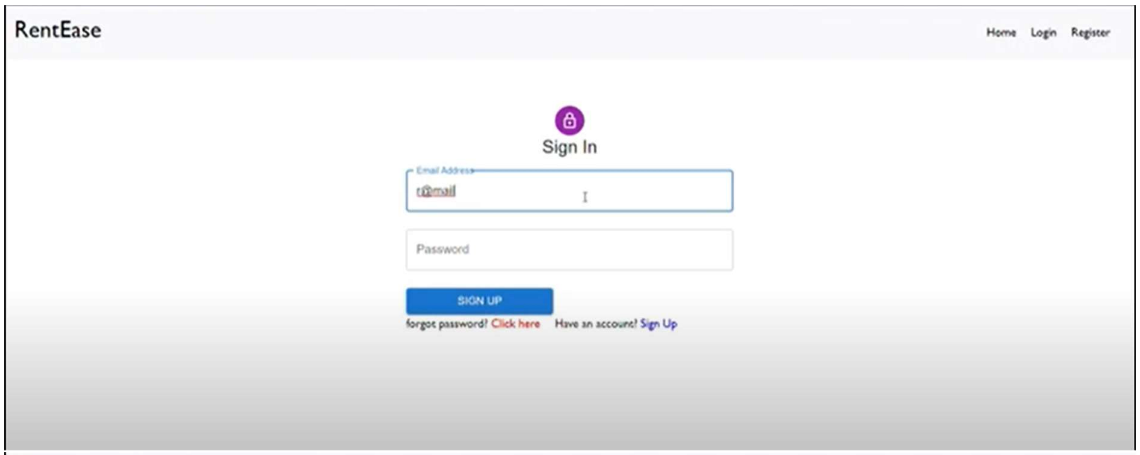


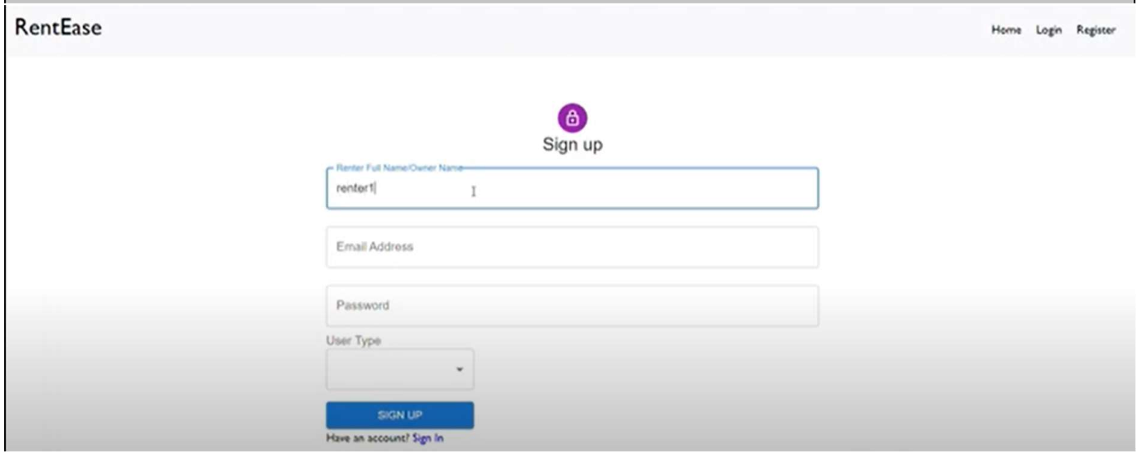
Location:
Bada
Property Type:
residential
Ad Type:
rent

For more details, click on get info

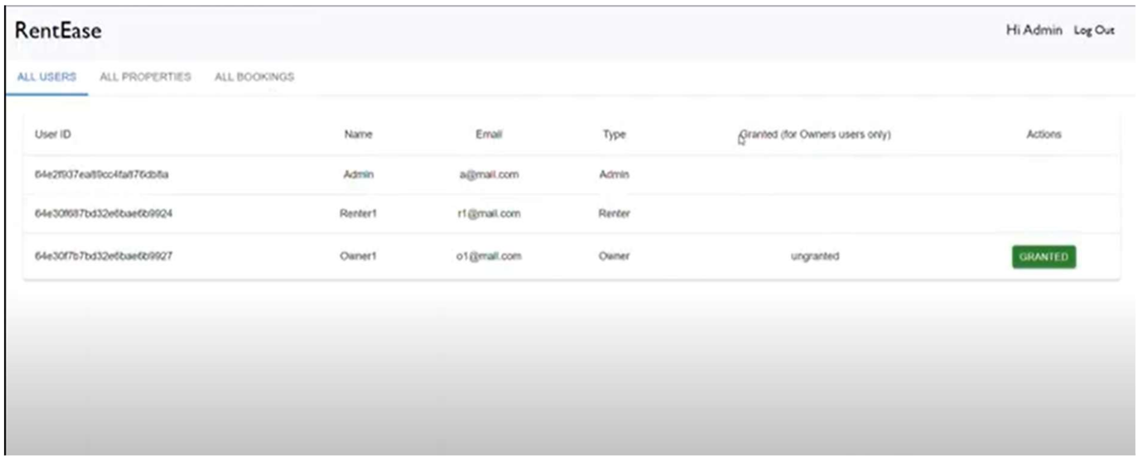
© 2023 Copyright: RentEase

- **Login and register page:**



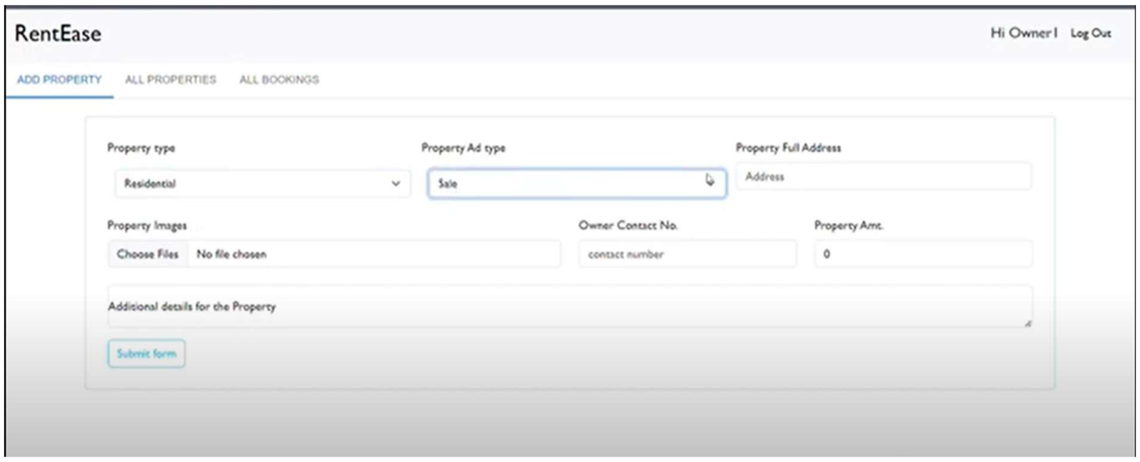


- **Admin Panel:**



User ID	Name	Email	Type	Granted (for Owners users only)	Actions
64e2f937ea89cc4fa876db8a	Admin	a@mail.com	Admin		
64e30f687bd32e6bae6b9924	Renter1	r1@mail.com	Renter		
64e30f7b7bd32e6bae6b9927	Owner1	o1@mail.com	Owner	ungranted	GRANTED

- **Owner Panel:**



RentEase

Hi Owner1 Log Out

ADD PROPERTY ALL PROPERTIES ALL BOOKINGS

Property type: Residential

Property Ad type: Sale

Property Full Address: Address

Property Images: Choose Files No file chosen

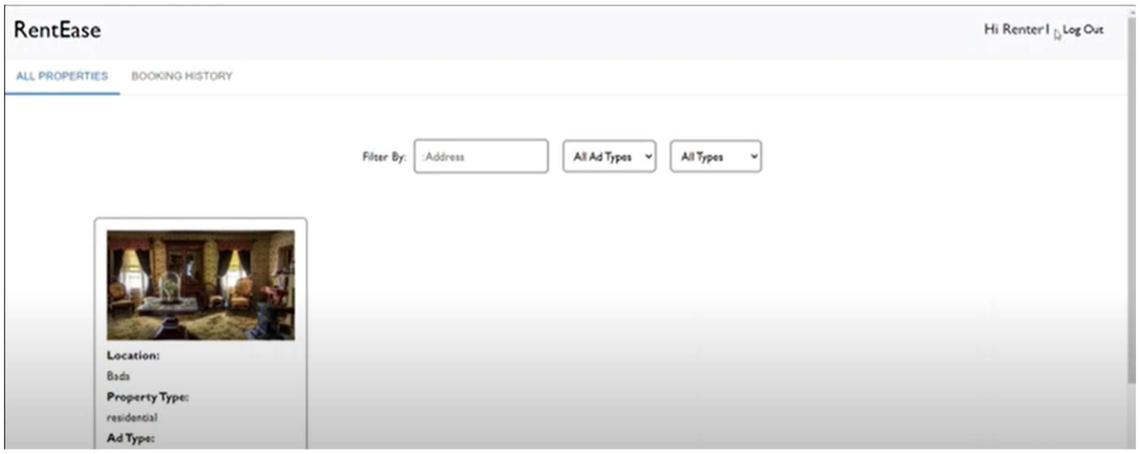
Owner Contact No.: contact number

Property Amt.: 0

Additional details for the Property

Submit form

- **Tenant panel:**



RentEase

Hi Renter1 Log Out

ALL PROPERTIES BOOKING HISTORY

Filter By: Address All Ad Types All Types

Location:
Bada
Property Type:
residential
Ad Type:

CHAPTER 9. Challenges Faced

1. **User Role Management**:
   - Developing a secure role-based access control system to ensure users only access their permitted features.
   - Handling edge cases where users might attempt to access features meant for other roles, requiring extensive testing and debugging.mplementing seamless transitions between roles (e.g., from Renter to Owner) without affecting existing user data.
2. **Database Design**:
   - Designing relationships between collections (e.g., Users, Properties, and Bookings) to minimize redundancy and improve scalability.
   - Balancing schema flexibility with query performance to accommodate varying property details and booking statuses.Managing database migrations and updates as the application evolved during development.
3. **Frontend Design**:
   - Creating a responsive design that works efficiently across multiple devices and screen sizes.
   - Ensuring the UI is intuitive and provides clear feedback to users during actions like submitting inquiries or viewing booking statuses. Handling performance issues when rendering large datasets (e.g., property listings) on the client side.
4. **Backend Optimization**:
   - Building APIs that handle high traffic efficiently, especially for property searches and booking inquiries.
   - Ensuring proper error handling and debugging for asynchronous operations involving MongoDB queries. Optimizing the backend to support future scalability as more users and properties are added.
5. **Testing and Debugging**:
   - Performing end-to-end testing for features like booking inquiries and owner approvals. Identifying and resolving bugs in edge cases (e.g., overlapping booking requests).Testing role-specific interfaces to ensure no accidental data exposure occurs.

CHAPTER 10. Future Enhancements

1. Mobile Compatibility:
    - Develop a dedicated mobile app for iOS and Android using frameworks like React Native or Flutter to expand accessibility. Optimize the user interface for mobile screens, ensuring easy navigation and usability.
    - Add offline functionality to allow renters and owners to access saved data without an internet connection.
2. Advanced Filtering:
    - Allow renters to search by more specific criteria, such as property size, floor plan, or specific amenities (e.g., pet-friendly properties, parking spaces).
    - Incorporate geolocation-based filtering to show properties near the user's current location. Enable dynamic filtering to refine search results in real-time as users adjust criteria.
3. Payment Integration:
    - Implement secure payment gateways like Stripe or PayPal for renters to pay booking deposits directly through the platform.
    - Add support for multiple currencies to cater to international users. Provide automated invoice generation and payment confirmation emails for completed transactions.
4. In-App Chat:
    - Develop a chat feature with real-time messaging between renters and property owners using WebSocket or libraries like Socket.io. Allow file sharing (e.g., lease agreements, property photos) directly through the chat.
    - Incorporate chat moderation features to filter inappropriate messages and maintain platform quality.
5. Machine Learning Recommendations:
    - Add AI-based recommendations to suggest properties based on user preferences, search history, and location. Use machine learning to predict property demand trends and suggest optimal rental pricing for owners.

CHAPTER 11. Conclusion

The House Rent App successfully bridges the gap between renters and property owners, offering a seamless platform for property listing, browsing, and booking. By leveraging the MERN stack, the application ensures high performance, scalability, and user-friendly interactions. With features like secure authentication, real-time notifications, and role-based access control, the app provides a robust and secure environment for both users and admins.

Through comprehensive planning, implementation, and testing, the app delivers a streamlined property rental experience that reduces friction for both parties. The inclusion of advanced search filters and communication tools enhances user satisfaction, while the admin dashboard ensures platform integrity and security. The project exemplifies how modern web technologies can be used to solve real-world problems, providing value to users and improving the overall rental process.

Future enhancements, such as mobile compatibility and payment integration, are expected to further expand the app's functionality, making it even more accessible and efficient. Overall, the House Rent App is a successful implementation that lays the groundwork for future innovation in property rental services.

CHAPTER 12.Reference

1. **React.js Official Documentation**
   - https://reactjs.org/
     Comprehensive documentation for building user interfaces using React.
2. **Node.js Official Documentation**
   - https://nodejs.org/en/docs/
     Detailed guides for building server-side applications with Node.js.
3. **Express.js Guide**
   - https://expressjs.com/
     Documentation for the Express framework, essential for handling backend routing and middleware.
4. **MongoDB Manual**
   - https://www.mongodb.com/docs/manual/
     Database design principles and best practices for managing collections and queries.
5. **Mongoose.js Documentation**
   - https://mongoosejs.com/docs/

     ORM for MongoDB to simplify schema creation and CRUD operations.
6. **Material UI**
   - https://mui.com/
     A React component library for creating modern and responsive UIs.
7. **Ant Design**
   - https://ant.design/
     A design system and component library for React-based applications.
8. **Bootstrap**
   - https://getbootstrap.com/
     A popular CSS framework for responsive design and styling.
9. **JSON Web Tokens (JWT)**
   - https://jwt.io/introduction/
     Overview and use cases of JWT for secure authentication.
10. **W3Schools (HTML, CSS, JavaScript Basics)**
    - https://www.w3schools.com/
      Beginner-friendly resources for learning web development basics.