



NM1042-MERN STACK POWERED BY MONGO DB

HOUSE RENT APPLICATION PROJECT

A Project Report

SUBMITTED BY

SWETHASRI A - 310821104098

SUGANYA A - 310821104095

VIGOSH P - 310821104109

SHIBIVARMAN M - 310821104089

IN PARTIAL FULFILLMENT OF THE AWARD FOR THE DEGREE

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

JEPPIAAR ENGINEERING COLLEGE

ANNA UNIVERSITY : CHENNAI - 600 025

NOVEMBER 2024



ANNA UNIVERSITY : CHENNAI - 600025

NOVEMBER 2024

ANNA UNIVERSITY : CHENNAI - 600025

BONAFIDE CERTIFICATE

Certified that this project report “**HOUSE RENT APPLICATION**” is the bonafide work of “**SWETHASRI A (310821104098)**” who carried out the project work for Naan Mudhalvan.

MENTOR

HEAD OF DEPARTMENT

DATE: _____

INTERNAL:

EXTERNAL:

TABLE OF CONTENTS

S.NO .	CONTENTS	PAGE NO
1.	INTRODUCTION	4
2.	PROJECT OVERVIEW	5
3.	ARCHITECTURE	10
4.	SETUP INSTRUCTIONS	12
5.	FOLDER STRUCTURE	16
6.	RUNNING THE APPLICATION	18
7.	API DOCUMENTATION	20
8.	AUTHENTICATION	24
9.	USER INTERFACE	27
10.	TESTING	31
11.	SCREENSHOTS	33
12.	KNOWN ISSUES	37
13.	FUTURE ENHANCEMENTS	38
14.	RESULT	42

1. Introduction

Project Title: House Rent Application with MERN Stack

This project provides an online platform for Home Rent is a web-based platform designed to streamline the rental process for both landlords and tenants. Built using the MERN stack (MongoDB, Express.js, React, Node.js), it enables landlords to list their properties and tenants to search, view, and apply for rentals. The platform is equipped with secure user authentication, an intuitive interface, and features like property management, application tracking, and advanced search options.

By leveraging modern technologies, the application aims to bridge the gap between landlords and tenants, making property rentals more accessible, transparent, and efficient. The goal of the project is to provide a seamless and efficient experience for both landlords and clients while ensuring secure transactions, job management, and real-time communication.

Team Members:

- Swethasri A (Frontend Developer): Responsible for building the user interface using React and ensuring the frontend's responsiveness and user experience.
- Suganya A (Full Stack Developer): Works on both the frontend and backend to ensure seamless integration and overall functionality of the application.
- Vigosh P (Backend Developer): Handles the backend API development with Node.js, Express.js, and MongoDB.
- Shibivarman M (UI & UX Design): Focused on designing the platform's interface and creating intuitive and user-friendly experiences.

The project is developed with the MERN (MongoDB, Express.js, React, Node.js) stack, utilizing Socket.io for real-time communication and Redux for state management on the frontend.

2. Project Overview

2.1 Purpose

The purpose of the house rent application is to modernize and simplify the property rental process by providing a centralized digital platform. It aims to:

1. **Simplify Search and Listings:** Help tenants find rental properties quickly and allow landlords to list and manage properties efficiently.
2. **Enhance Communication:** Enable seamless interaction between landlords and tenants through in-app messaging.
3. **Streamline Transactions:** Facilitate secure rent payments and lease management.
4. **Improve Transparency:** Build trust with features like user verification, reviews, and detailed property information.
5. **Save Time:** Reduce the effort and time spent on traditional rental processes.

2.2 Objective

To simplify the rental process by providing a centralized platform for landlords and tenants.

- **Key Features:**
 - User registration and login (landlord/tenant).
 - Property listing with descriptions, photos, and pricing.
 - Advanced search

and filter functionality. ○ Application submission and status tracking.

- Admin dashboard for managing users and properties.

For Tenants:

- Advanced property search with filters (location, price range, property type, etc.).
- View property details, photos, and virtual tours. ○ Book property visits or directly apply for a rental.
- Easy communication with landlords through in-app messaging.

For Landlords:

- Effortless property listing with detailed information and multimedia support.
- Track rental applications and communicate with potential tenants. ○ Payment management for rent collection.
- Analytics for property performance and tenant management.

Additional Features:

- Tenant and landlord profile verification for added security. ○ Integrated payment gateway for hassle-free transactions. ○ Notifications and reminders for rent due dates or property updates. ○ Reviews and ratings to enhance transparency.

• Tech Stack:

Frontend: React.js

- Designed for seamless user experience with responsive interfaces.
- Technologies: HTML, CSS, JavaScript (or frameworks like React, Angular, or Vue.js).

Backend: Node.js with Express.js

- Manages data processing and user authentication.
- Technologies: Node.js, Python (Django/Flask), Ruby on Rails, etc.

Database: MongoDB

- Stores user information, property listings, and transaction details.
- Options: MySQL, PostgreSQL, MongoDB, etc.

Authentication: JSON Web Tokens (JWT)

- Cloud-based hosting for scalability (AWS, Azure, or Google Cloud).

2.3Goals:

The core goals of the platform include:

The core goal of the house rent application is to create a seamless, secure, and efficient ecosystem for renting properties, empowering both landlords and tenants by digitizing the entire rental process. It strives to:

1. Provide a user-friendly platform for property listing, searching, and management.
2. Ensure transparency and trust through verified profiles, reviews, and secure transactions.

3. Minimize time and effort involved in property rentals by streamlining communication and payment processes.
4. Enhance accessibility by enabling users to manage rentals anytime, anywhere.

2.4 Features:

1. Advanced Property Search:

- Filter properties by location, budget, property type, and more.
- View detailed listings with photos, descriptions, and virtual tours.

2. Property Listings for Landlords:

- Easily upload property details, images, and rental terms.
- Manage multiple listings and track tenant inquiries.

3. In-App Communication:

- Direct messaging between landlords and tenants for quick responses.
- Notifications for updates on applications, payments, and reminders.

4. Secure Payment Integration:

- Enable tenants to pay rent digitally through secure payment gateways.
- Track payment history and send automated due date reminders.

5. Profile Verification:

- Ensure security with verified landlord and tenant profiles.
- Build trust and credibility in transactions.

6. Reviews and Ratings:

- Tenants can review properties and landlords, while landlords can rate tenants.
- Foster a transparent rental ecosystem.

7. Document Management:

- Upload and manage lease agreements and ID proofs within the app.
- Simplify access to important documents for both parties.

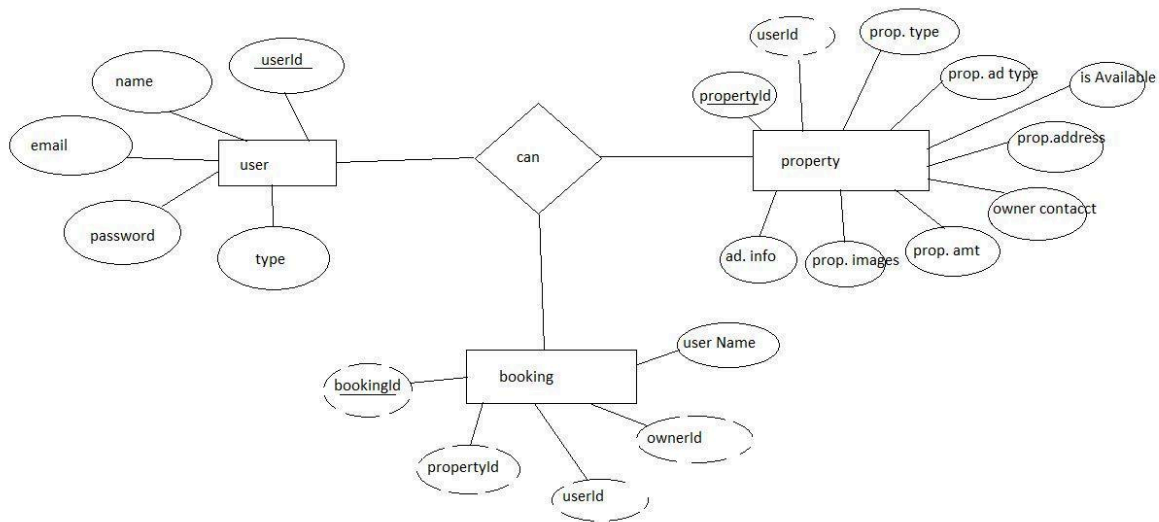
8. Analytics and Insights:

- Landlords can track property performance, occupancy rates, and payments.

Tenants can monitor application status and manage rental budget.

3. Architecture

DIAGRAM:



3.1 System Architecture

- **Frontend:** React handles UI/UX, communicates with the backend via API calls.
- **Backend:** Node.js with Express.js manages APIs and business logic.
- **Database:** MongoDB stores user, property, and application data.

3.2 Components

3.2.1 Frontend (React.js) ○ **Purpose:** Handles the user interface, routing, and interactions.

- **Key Tools/Technologies:** ○ **React Router:** For navigation between pages.
- **Axios/Fetch API:** For

making API calls. ○ **Redux/Context API:** For state management (optional).

- **Responsive Design:** Ensures mobile and desktop compatibility.

Functionality:

- Property search, listing creation, profile management. ○ Interactive dashboards for landlords and tenants. ○ Mobile and web-responsive design.

3.2.2 Backend (Node.js with Express.js)

- **Purpose:** Serves as the intermediary between the frontend and the database.

Key Features:

- **RESTful APIs:** Modular endpoints for authentication, property management, and applications.
- **Middleware:** Handles authentication (JWT), error handling, and request validation.
- **Secure Data Handling:** Encrypts passwords using libraries like bcrypt.

Functionality:

- API endpoints for frontend communication.
- Processing rental applications and payments.
- Managing user sessions and security protocols.

3.2.3 Database (MongoDB)

- **Purpose:** Stores structured data for users, properties, and rental applications.
- **Schema Design:**

- **User:** Stores user credentials, roles (landlord/tenant), and profile info.
- **Property:** Stores property details (address, price, images, description, etc.).
- **Application:** Tracks tenant applications and statuses for specific properties.

3.2.4 Data Types:

- User data (profiles, verifications).
- Property data (listings, media).
- Transaction data (rent payments, history).

3.2.5 Integration Layer:

Purpose: Facilitates interaction with third-party services.

Components:

- Payment Gateway APIs.
- Notification services (email, SMS, push notifications).
- Map integration (Google Maps for property location).

3.2.6 Security Layer:

- **Purpose:** Protects user data and transactions.
- **Features:**
 - Encryption (SSL/TLS).
 - Role-based access control (RBAC).
 - Two-factor authentication (2FA).

3.2.7 Deployment Layer: ○ **Purpose:** Hosts

and delivers the application to users. ○

Technology:

- Cloud services (AWS, Azure, Google Cloud).
- Load balancers and auto-scaling for performance.

4. Setup Instructions

#Prerequisites:

Before setting up the project locally, make sure you have the following software installed on your machine:

- **Node.js:** Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is required to run both the frontend and backend of this project. You can download Node.js from the official website [here](<https://nodejs.org/>).
- **MongoDB:** MongoDB is a NoSQL database used to store user data, job postings, bids, and messages. You can either install MongoDB locally or use MongoDB Atlas, a cloud-based solution, for easier setup and management.
- **npm (Node Package Manager):** npm is used to install the necessary dependencies for both the frontend and backend. npm comes pre-installed with Node.js, so you will have it once you install Node.js.

#Installation:

1. Clone the Repository

Begin by cloning the project repository to your local machine using the following command:

```
git clone <repository-url>
```

Replace ``<repository-url>`` with the actual URL of your project repository.

2. Install Dependencies

After cloning the repository, navigate to the following directories to install the necessary npm packages:

- For the Frontend (Client):

Navigate to the `client` directory, which contains all the frontend code built with React. Run the following command to install the required dependencies:

```
cd client  
npm install
```

This will install all the packages listed in the `package.json` for the frontend.

- For the Backend (API):

Similarly, navigate to the api directory, which contains the backend code built with Node.js and Express. Run the following command: `api/ npm install`

This will install all the necessary backend dependencies such as Express, MongoDB, and other packages required for API functionality.

3. Setup Environment Variables

To ensure the frontend, backend, and socket server communicate securely, you need to set up the environment variables:

- Create a `.env` file in the root directory for both the frontend (client), backend (api), and socket server (socket).

- In the backend `.env` file (`api/.env`), include the following:

`env`

`MONGODB_URI=<your-mongodb-uri>`

`JWT_SECRET=<your-secret-key>`

- `MONGODB_URI`: Connection string for MongoDB. If using MongoDB

Atlas, you'll find this in your dashboard. For local setups, it could look like `'mongodb://localhost:27017/your-database-name'`.

- `JWT_SECRET`: A secret key used for generating JWT tokens for authentication. You can generate a random string for this key.

- In the socket `.env` file (`socket/.env`), you may want to include configurations for WebSocket authentication, if applicable.

- Frontend Environment Variables: While the frontend does not typically need environment variables for authentication, you may want to set up some for API URLs, e.g., `env`

`REACT_APP_API_URL=http://localhost:3001`

This variable can be used to interact with the backend API from the frontend.

4. Run the Server and Client

After installing all dependencies and setting up your environment variables, you are ready to run the project. Open three separate terminal windows (or tabs) to run the servers simultaneously:

- For the Backend (API):

In the terminal, navigate to the api directory and run the following command to start the backend server: `cd api/ npm start`

The backend will now be running at ``http://localhost:3001``.

- For the Frontend (Client):

In another terminal window, navigate to the client directory and run: `cd client/ npm start`

The frontend will be available at ``http://localhost:3000``.

Access the Application

Once all the servers are running, you can access the application in your web browser:

- Frontend: Open your browser and go to `http://localhost:3000` to interact with the user interface of the application.

- Backend: The backend API can be accessed at `http://localhost:3001`, though it is primarily used by the frontend.

Socket Server: The real-time communication can be tested using the frontend, which connects to the socket server to facilitate messaging between users.

5. Folder Structure

Frontend

The client folder contains all the frontend-related files, built using React.js.

client/

```
├── client/
│   ├── node_modules/ # Contains the dependencies installed for the
│   │   │               frontend (via npm install).
│   ├── public/ # Contains the public files like the `index.html` and favicon.
│   ├── src/ # Contains all the source code of the React frontend.
│   │   ├── App.js # The root component that contains the routing logic for
│   │   │           the app.
│   │   ├── data.js # A file for mock data, API endpoints, or static data used
│   │   │           in the app.
│   │   └── index.js # Entry point for the React app, rendering the root
│   │               component.
└── README.md
```

- assets/: Contains images, fonts, icons, or other static assets used throughout the frontend application.
- components/: Contains reusable React components that can be used across different pages, such as bookingpage, Profile and host.
- ClientComponents/: Components that are primarily used by the clients

- ServerComponents/: Components tailored for profile update, property management, viewing project listings and login.
- redux/: Contains Redux slices, actions, and reducers that manage global state such as user data, new property and booking updates.

Backend

The server folder contains the backend-related files, which include the API routes, controllers, models, and other necessary server-side logic.

```

├── server/
│   ├── models/ # Mongoose schemas for MongoDB collections (users,
│   │           jobs, bids).
│   ├── node-modules/ # Dependencies installed for the backend are stored
│   │               here.
│   ├── public/ # Define the public visible in the site.
│   ├── routes/ # Defines API routes for user authentication, job postings,
│   │           bidding, and messaging.
│   └── index.js/ #Logic of need index data.
└── README.md

```

- config/: This folder contains the configuration files like the database connection (MongoDB) and the environment variables used by the server.
- middleware/: Includes reusable middleware functions that are applied to routes, such as authentication verification and error handling.
- models/: Defines Mongoose schemas for the collections in the MongoDB database. Each model corresponds to a collection (users, jobs, bids).

- routes/: Contains Express.js route files that define the API endpoints for jobs, users, bids, etc.
- uploads/: Stores any uploaded files like images or documents related to jobs or user profiles.
- app.js: The entry point to the backend server. It sets up the Express app, connects to the database, and starts the server.

6. Running the Application

To run the House Rent Application locally, follow the steps below:

Frontend:

1. Navigate to the client directory: `cd client/`
2. Install the required dependencies: `cash npm install`
3. Start the frontend server:

`npm start`

4. The frontend will be available at
<http://localhost:3000>.

5. **# Backend:**

1. Navigate to the
server directory: `cd
server/`
2. Install the required
dependencies: `bash
npm install`
3. Start the backend
server:

`npm start`

4. The backend will
be available at :

[<http://localhost:3001>](<http://localhost:>]

By following these steps, you'll have the application running locally, with the frontend accessible on port `3000`, the backend on port `3001`, and the socket server enabling real-time communication.

7. API Documentation

The backend exposes several endpoints for managing users, jobs, bids, and messages. Below is the detailed API documentation for each endpoint:

User Authentication

POST auth/register

- Description: Register a new user (freelancer or client).
- Request Body:

```
```json
{
 "Name": "John",
 "email": "johndoe@example.com",
 "password": "password123",
 "type": ""
}
```
```

- Response:

```
```json
{
 "message": "User registered successfully"
}
```
```

POST auth/login

- Description: Login a user and return a JWT token.

- Request Body:

```
```json
{
 "email": "johndoe@example.com",
 "password": "password123"
}
```
```

- Response:

```
```json
{
 "token": "JWT_Token_Here"
}
```
```

Rental home listing

POST /properties /create

- Description: Create a new properties (client only).

- Request Body:

```
```json {
creator,
category,
type,
```

streetAddre

ss,

aptSuite,

city,

province,

country,

guestCount,

bedroomCo

unt,

bedCount,

bathroomC

ount,

amenities,

title,

description,

highlight,

highlightDe

sc, price,

}

``

- Response:

```json

```
{  
  "message": "newListing success"  
}  
``
```

GET /properties/{Id}

- Description: Retrieve the properties based in the ID - Response:

``json

```
[
```

```
{
```

creator, category, type, price

```
}
```

8. Authentication

8.1 Authentication strategy

1. User Registration:

- Allows new users (tenants or landlords) to create an account by providing necessary details like name, email, password, and role.
- Passwords are securely hashed before storage.

2. User Login:

- Validates user credentials (email and password) to allow access.
- Generates a JSON Web Token (JWT) on successful authentication.

3. Role-Based Access Control (RBAC):

- Restricts access to certain features based on user roles (e.g., only landlords can list properties).

4. Password Reset:

- Enables users to reset their password through a secure link sent to their registered email.

5. Token-Based Authentication:

- Protects API endpoints with JWT tokens, ensuring secure communication.

8.2 Authentication Flow

1. Registration:

- User submits details via the registration form. ◦ Backend validates inputs and hashes the password using libraries like bcrypt.
- A new user record is saved to the database.

2. Login:

- User provides email and password. ◦ Backend verifies credentials and generates a JWT with user-specific claims (ID, role).
- Token is sent to the frontend for subsequent API calls.

3. Protected Routes:

- Middleware checks for the presence of a valid JWT in the request headers.

- If valid, the request proceeds; otherwise, access is denied.

4. Logout:

- Frontend deletes the stored token, effectively logging out the user.

8.3Frontend Integration

1. Login and Registration Forms:

- Capture user input and send API requests to the backend endpoints.

2. Token Storage:

- Store the JWT in local storage or cookies (with secure and HTTPOnly attributes).

3. API Calls with Tokens:

- Include the token in the Authorization header of protected API requests.

8.4Security Best Practices

- 1. Hash Passwords:** Use bcrypt to hash and salt passwords securely.

2. **Secure Tokens:** Store JWTs securely and implement short expiration times.
3. **Use HTTPS:** Protect data transmission with SSL/TLS encryption.
4. **Validate Inputs:** Sanitize and validate all user inputs to prevent injection attacks.

9. User Interface

UI Features:

9.1 Core Design Principles

1. **User-Centric Design:** ◦ Focus on simplicity, clarity, and usability.
2. **Responsive Design:** ◦ Ensure compatibility across mobile, tablet, and desktop screens.
3. **Modern Aesthetics:** ◦ Use clean layouts, intuitive navigation, and appealing color schemes.
4. **Accessibility:**
 - Provide support for screen readers, keyboard navigation, and proper contrast ratios.

9.2 UI Components and Screens

1. Landing Page

- **Purpose:** First impression, showcasing the application's features.
- **Key Elements:**

- Search bar for properties (e.g., "Find your next home").
- Call-to-action buttons (e.g., "Sign Up" or "Post a property").
- Highlights of key features (e.g., verified listings, secure payments).
- Testimonials or reviews.

2. Registration and Login

- **Purpose:** Enable users to register or log in securely.
- **Key Elements:**
 - Input fields for email, password, name, and role selection (tenant or landlord).
 - Error messages for validation issues (e.g., "Invalid email").

3. Dashboard

- **Purpose:** A personalized hub for tenants and landlords.
- **Key Elements:**
 - **Tenant View:**
 - Saved properties, payment status, and rental history.
 - **Landlord View:**
 - Property listings, tenant applications, and earnings overview.

4. Property Listings Page

- **Purpose:** Display available properties for tenants.
- **Key Elements:**
 - Filters (location, price range, property type, number of rooms).
 - Cards with property images, price, and brief details.
 - Sorting options (e.g., price low to high, most recent).

5. Property Details Page

- **Purpose:** Provide detailed information about a specific property.
- **Key Elements:**
 - High-quality images and videos.
 - Description, amenities, and location map.
 - Landlord contact button or request to book.

6. Add/Edit Property Page (For Landlords)

- **Purpose:** Enable landlords to create or update property listings.
- **Key Elements:**
 - Form fields for property name, description, price, location, and photos.
 - Preview option before posting.

7. Search Results Page

- **Purpose:** Display properties based on user search queries.
- **Key Elements:**
 - List/grid view toggle.
 - Interactive map showing property locations.
 - Pagination or infinite scrolling for results.

8. Payments and Receipts

- **Purpose:** Handle rent payment and view transaction history.
- **Key Elements:**
 - Payment status (paid, due, overdue).
 - Integration with payment gateways (e.g., PayPal, Stripe).
 - Downloadable rent receipts.

9. Help and Support

- **Purpose:** Provide users with assistance.
- **Key Elements:**

- FAQ section.
- Contact form for queries.
- Live chat or chatbot integration.

9.3 Tools and Technologies used:

1. **Frontend Framework:** React.js or Angular for dynamic and interactive components.
2. **Styling Libraries:** Tailwind CSS, Material-UI, or Bootstrap for consistency.
3. **Maps Integration:** Google Maps API for location-based features.
4. **Testing:** Use tools like Cypress or Jest to test UI responsiveness and functionality.

10. Testing

#Key Test Scenarios

10.1 Frontend Testing

- Registration Form:
 - Verify form validations (e.g., required fields, email format).
 - Ensure error messages are displayed correctly.
- Property Search:
 - Test filter combinations and sorting options.

- Validate the interactive map's functionality.
- Payment Processing:
 - Ensure successful transactions.
 - Check for proper error handling for failed payments.

10.2 Backend Testing

- API Endpoints:
 - Test CRUD operations for properties, users, and payments.
 - Validate token-based authentication for protected routes.
- Database:
 - Verify schema constraints (e.g., unique email for users).
 - Test relationships between users, properties, and payments.

10.3 Security Testing

- Check for SQL injection and XSS vulnerabilities.

- Validate secure storage of sensitive data (e.g., hashed passwords).
- Ensure proper implementation of role-based access control (RBAC).

10.4 Testing Tools

1. Frontend Testing:

- **Jest**: For unit and integration tests of React components.
- **Cypress**: For E2E testing of user flows.

2. Backend Testing:

- **Postman**: For API testing and request validation.
- **Supertest**: For automated API tests in Node.js.

3. Performance Testing: ◦ **JMeter**: For load and stress testing.

4. Security Testing: ◦ **OWASP ZAP**: For vulnerability scanning.

5. Database Testing:

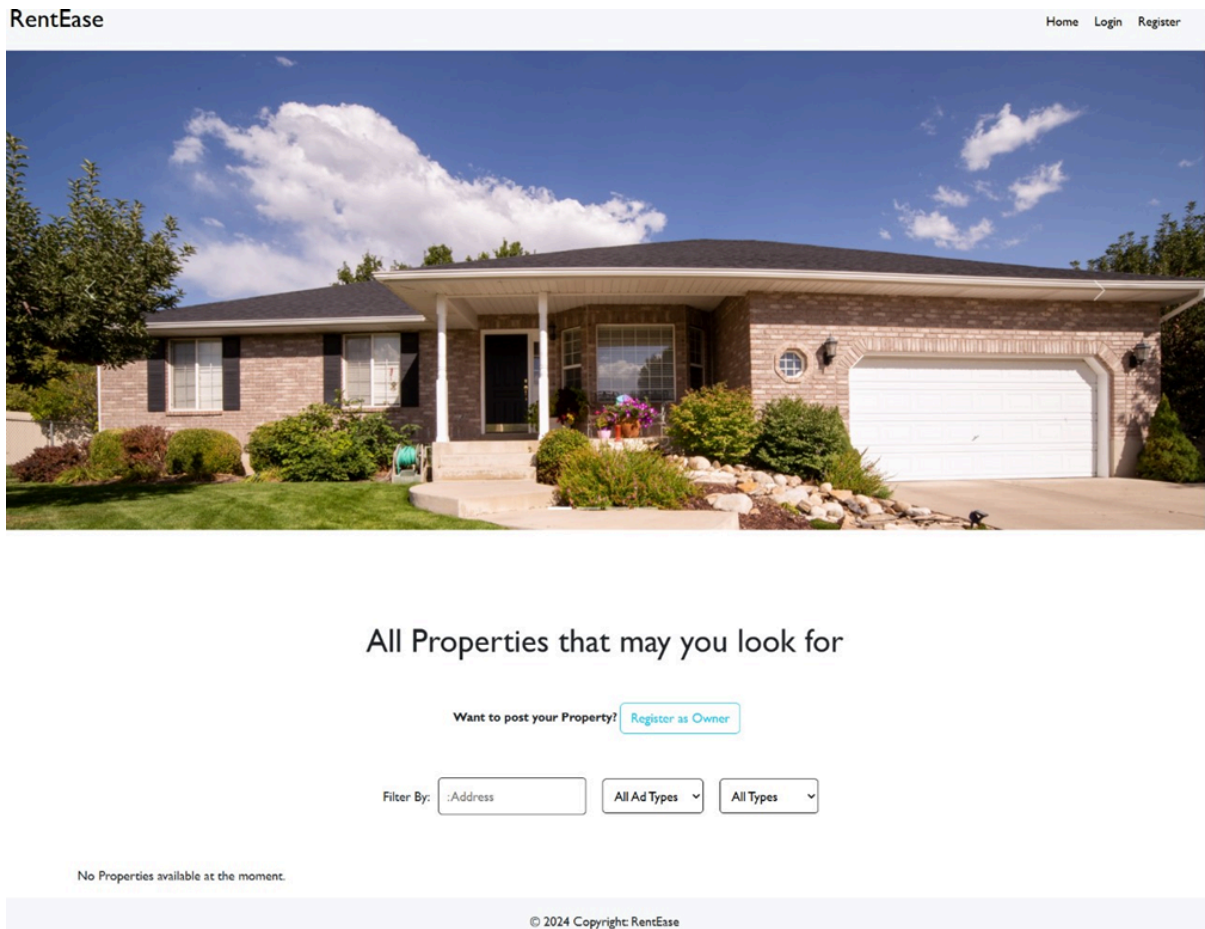
- Custom SQL scripts to validate data consistency.

11.Screenshot:

#screenshots

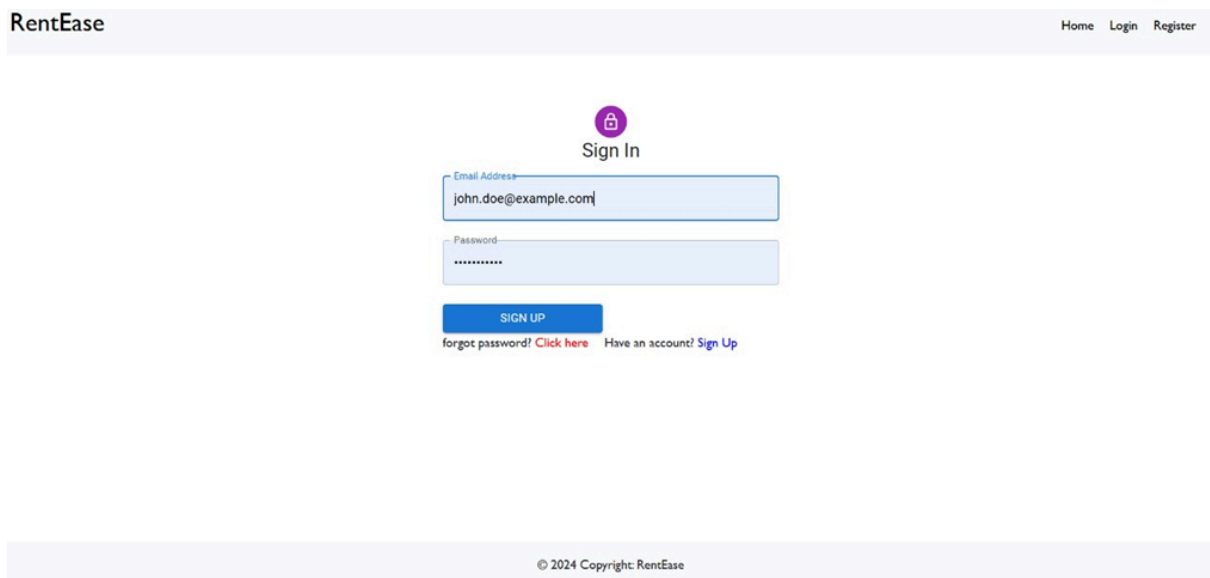
1.HomePage:

The screenshots below highlight key features of the house rent application platform:



2. Login Page:

- The login page provides users with a straightforward interface for signing into their accounts with their credentials.



The screenshot displays the login interface for 'RentEase'. At the top left is the 'RentEase' logo, and at the top right are links for 'Home', 'Login', and 'Register'. The main content area features a purple lock icon and the text 'Sign In'. Below this are two input fields: 'Email Address' containing 'john.doe@example.com' and 'Password' with masked characters. A blue 'SIGN UP' button is positioned below the password field. At the bottom of the form, there are two links: 'forgot password? Click here' and 'Have an account? Sign Up'. The footer contains the copyright notice '© 2024 Copyright: RentEase'.

3. User signup page:

- The **User Sign-Up Page** is where users (tenants or landlords) register to access the house rent application. It should be simple, intuitive, and secure, ensuring that all necessary information is collected efficiently.



Sign up

Renter Full Name/Owner Name
ram

Email Address
1234@gmail.com

Password

User Type
Owner

SIGN UP

Have an account? [Sign In](#)

4. Property Booking Page:

- Displays all the available place posted by clients. Allow users to search for properties by location, keywords, or property names.

Property Info

Owner Contact: 786543890

Location: padur

Availability: Available

Property Type: residential

Property Amount: Rs.25000

Ad Type: rent

Additional Info:

Your Details to confirm booking

Full Name

Phone Number

bansali

986543098

Book Property

5.Booking Lists:

- The **Booking Lists Page(is owners)** is a dedicated section where displays a summary of all information, User Id, Email, type. It provides an organized dashboard to streamline their tasks and improve usability.

| User ID | Name | Email | Type | Granted (for Owners users only) | Actions |
|--------------------------|-----------|---------------------------|--------|---------------------------------|-----------|
| 67371c7aa3f85b27ddf69767 | Suganya A | suganyaanand006@gmail.com | Owner | granted | UNGRANTED |
| 67371d12a3f85b27ddf6976d | Sugan | 1234@gmail.com | Renter | | |
| 67372093a3f85b27ddf69781 | John | john.doe@example.com | Admin | | |
| 67398e2431da1bbbadc62162 | Ram | 4532@gmail.com | Owner | granted | UNGRANTED |

12. Known Issues

1. Frontend Issues

- **Form Validation Problems:**
 - Certain edge cases bypass client-side validation, e.g., special characters in name fields or invalid email formats.
- **Cross-Browser Compatibility:**
 - Features or styles may not render consistently across different browsers (e.g., Safari vs. Chrome).

Responsiveness Issues: ◦ Some components may not adapt well to mobile or tablet screens.

- **Slow Loading Times:**
 - Property images take too long to load due to lack of image optimization.

1. Backend Issues

- **API Response Delays:**
 - High latency when fetching property details due to inefficient database queries.
- **Authentication Errors:**

- Token expiration not handled properly, causing users to be logged out unexpectedly.
- **Data Inconsistencies:**
 - Concurrent updates (e.g., multiple users editing the same property)

13. Future Enhancements

1. Advanced Search and Filters

- Enhancement: Add filters for amenities (e.g., pet-friendly, parking, furnished), neighbourhood ratings, and proximity to landmarks.
- Benefit: Improves the user experience by making property discovery more precise.

2. Real-Time Messaging

- Enhancement: Enable a chat feature for landlords and tenants to communicate directly within the app.

Benefit: Reduces dependency on external communication tools and enhances user engagement.

3. Virtual Tours

- Enhancement: Integrate 360° virtual property tours using VR technology or panoramic images.
- Benefit: Provides a better preview of properties, especially for remote users.

4. AI-Powered Recommendations

- Enhancement: Use machine learning to recommend properties based on user preferences and browsing history.
- Benefit: Personalizes the user experience and increases engagement.

5. Payment Integration

- Enhancement: Support recurring rent payments via credit cards, UPI, or digital wallets.
- Benefit: Simplifies rent collection for landlords and ensures timely payments for tenants.

6. Calendar Scheduling

- Enhancement: Introduce a booking system for property visits with available time slots.
- Benefit: Streamlines scheduling and reduces conflicts in property viewing.

7. Tenant and Landlord Reviews

- Enhancement: Add a review and rating system for tenants and landlords to build trust within the platform.
- Benefit: Encourages accountability and improves community credibility.

8. Multi-Language Support

- Enhancement: Offer localization with support for multiple languages and currencies.
- Benefit: Expands usability for international users.

9. Mobile App Development

Enhancement: Launch a dedicated mobile application for Android and iOS.

- Benefit: Improves accessibility and user convenience with push notifications and offline access.

10. Enhanced Security

- Enhancement: Introduce multi-factor authentication (MFA) and biometric logins for enhanced user account security.
- Benefit: Protects user data and reduces the risk of unauthorized access.

11. Analytics Dashboard

- Enhancement: Provide landlords with insights into tenant behaviour, rental trends, and earnings analytics.

- Benefit: Helps landlords make informed decisions and optimize property performance.

12. Maintenance Request Management

- Enhancement: Allow tenants to raise and track maintenance requests directly through the app.
- Benefit: Simplifies issue resolution and enhances tenant satisfaction.

13. Blockchain Integration

- Enhancement: Use blockchain for secure and transparent lease agreements.
- Benefit: Ensures tamper-proof contracts and builds trust between users.

14. Energy Efficiency Insights

- Enhancement: Display energy efficiency ratings for properties (e.g., water, electricity usage).
- Benefit: Attracts environmentally conscious tenants and promotes sustainable living

Conclusion:

In conclusion, the house rent application using MERN stack is designed to bridge the gap between landlords and tenants, offering a seamless platform for property management and rental operations. With a robust architecture, user-friendly interface, and secure backend, it addresses the core needs of both parties, streamlining the process of finding and managing rental properties.

While the current version provides essential features such as property listings, user authentication, and basic search functionalities, there is immense potential for growth. Future enhancements like advanced filters, AI-powered recommendations and virtual tours will further elevate the user experience, making the application more efficient and competitive.

14.Result:

The app has been successfully developed using the required software and technologies