

# PROJECT REPORT

## HOUSE RENT APP USING MERN

TEAM MEMBER	ROLE
THASNEEM FATHIMA	Backend & Database
AISHWARYA DEVI	Backend & Database
RAMYA	Frontend User Interface
DARSHINI	Frontend User Interface
DEVI	Frontend User Interface

## Project Overview

### Purpose :

The House Rent App is designed to connect renters and property owners, providing a seamless platform to streamline the rental process. The app facilitates everything from property search to finalizing lease agreements, ensuring transparency and efficiency for all users.

### Features :

- User Registration: Separate accounts for renters and property owners with secure login.
- Messaging System: In-app communication between renters and property owners for inquiries and negotiations.
- Admin Panel: Tools for managing users, approving owners, and ensuring platform governance.
- Property Listings: The app provides a database of available rental properties, complete with detailed descriptions, photos, location, rent amount, and other relevant information.
- Search Filters: Users can apply various filters to narrow down their search results based on criteria such as location, rent range, property type (apartment, house, room, etc.), number of bedrooms, amenities, and more.

## Architecture

The technical architecture of our House rent app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axios library to connect with backend easily by using RESTful Apis.

The frontend utilizes the bootstrap and material UI library to establish real-time and better UI experience for any user whether it is admin, doctor and ordinary user working on it. On the backend side, we employ Express.js frameworks to handle the server-side logic and communication.

### A) Frontend architecture:

The application defines three main roles: **Renter**, **Property Owner**, and **Administrator**. Each role has distinct access levels and workflows tailored to their responsibilities.

## **1. Renter Workflow**

1. **Account Creation:**
  - Register an account and log in.
2. **Property Browsing:**
  - Access the Renter Dashboard and browse properties.
  - Use filters to narrow down options based on preferences.
3. **Booking:**
  - Select a property, provide required details (e.g., contact number), and book.
  - Track the booking status in the dashboard.
4. **Booking Completion:**
  - If approved, proceed with further communication with the property owner.

## **2. Property Owner Workflow**

1. **Account Creation:**
  - Register an account and log in as a property owner.
2. **Property Management:**
  - Access the Owner Dashboard to add properties.
  - Provide property details (type, address, price) and upload images.
3. **Booking Management:**
  - View booking requests from renters.
  - Approve or reject requests based on property availability.
4. **Property Updates:**
  - Edit or remove properties as needed.

## **3. Administrator Workflow**

1. **Authentication:**
  - Log in to the Admin Dashboard.
2. **User Management:**
  - Admin reviews and approves property owners after verifying their details (such as identity and ownership). Only after approval can the owner post and manage properties.
  - Monitor all users (renters and owners) and their activities.
  - Take corrective actions if any user violates policies.
3. **Property Oversight:**
  - Ensure all listed properties adhere to platform standards.

## **B) Backend architecture**

### **1. User Management Service**

**Purpose:** Manages user registration, login, and authentication.

**Responsibilities:**

- **Register:** Store user data (email, password) in the database, ensuring password security through hashing.
- **Login:** Validate the user's credentials and generate a JWT token on success.
- **Password Reset:** Initiates a password reset process by sending an email with a reset link.

**Dependencies:**

- bcryptjs – for password hashing.
- jsonwebtoken – for token creation and validation.
- nodemailer – for sending email notifications for password reset.

## 2. Property Management Service

**Purpose:** Handles operations related to property listings, including adding, retrieving, and managing properties.

**Responsibilities:**

- **Add Property:** Allows users to add new property listings (including details like title, description, price).
- **Get Properties:** Retrieve properties, either all properties or based on search/filter criteria.
- **Update/Delete Property:** Allows property owners to edit or delete their listings.

**Dependencies:**

- mongoose – for interacting with MongoDB to store and manage property data.

## 3. Booking Management Service

**Purpose:** Manages the booking process for properties, allowing tenants to book and owners to manage bookings.

**Responsibilities:**

- **Book Property:** Allows tenants to make bookings on available properties.
- **Get Bookings:** Retrieve bookings for tenants and property owners.
- **Manage Booking:** Enables updating or canceling a booking based on the property owner's decisions.

## 4. File Upload Service

**Purpose:** Handles file uploads for property images.

**Responsibilities:**

- **Upload Images:** Enables property owners to upload images with their listings.
- **Validate Files:** Ensures that only valid file types and sizes are uploaded.

**Dependencies:**

- multer – for handling file uploads to the server.

## 5. Utilities Service

**Purpose:** Manages email notifications for user actions (such as password resets and booking confirmations).

**Responsibilities:**

- **Send Password Reset Email:** Sends a reset link to users requesting a password reset.
- **Send Booking Confirmation Email:** Sends a confirmation email after a successful property booking.

**Dependencies:**

- nodemailer – for sending emails to users.

**Database:**

### 1. User Schema

The User schema stores essential information about users within the system. Each user has attributes such as name, email, password, and user type. The user type distinguishes between property owners, tenants, and admins. This schema is primarily used for authentication and authorization.

- **Key Fields:**

- name: Full name of the user.
- email: Email address for login and communication.
- password: User's hashed password.
- type: Type of user (e.g., tenant, owner, admin).

### 2. Property Schema

The Property schema defines the structure for storing property listings on the platform. It includes details such as the property type, address, owner contact information, and additional information about the property.

- **Key Fields:**

- ownerId: References the user who owns the property.
- propertyType: Type of the property (e.g., apartment, house).
- propertyAdType: Type of advertisement for the property (e.g., rent, sale).
- propertyAddress: The address of the property.
- ownerContact: The owner's contact number.
- propertyAmt: The price or rent amount for the property.
- propertyImage: Image or media associated with the property.
- additionalInfo: Any extra details about the property.

### 3. Booking Schema

The Booking schema tracks the bookings made by users for properties. Each booking links to both the user who made the booking and the property being booked. The schema includes details about the user's name, contact information, and booking status.

- **Key Fields:**

- propertyId: References the property being booked.
- ownerId: References the property owner.
- userID: References the user who made the booking.
- userName: The name of the user who made the booking.
- phone: The contact number of the user.
- bookingStatus: The current status of the booking (e.g., confirmed, pending).

### Database Interactions

- **User-Property Relationship:** A user can own multiple properties. The ownerId in the Property schema refers to the user who owns the property.
- **User-Booking Relationship:** A user can make multiple bookings. The Booking schema links a user to the booking they made, referencing the userID.
- **Property-Booking Relationship:** A property can have multiple bookings associated with it. The Booking schema links a booking to a property using the propertyId field.

## Setup Instruction

### Installation :

#### 1. Clone the Repository:

```
git clone [repository URL]

cd house-rent
```

#### 2. Install Dependencies:

##### ○ Frontend:

```
cd frontend

npm install
```

##### ○ Backend:

```
cd ../backend

npm install
```

#### 3. Set Up Environment Variables:

- Create a .env file in the backend directory with the following variables:

```
MONGO_URI=[MongoDB Connection String]
```

```
JWT_SECRET=[Secret Key]
```

```
PORT=5000
```

#### 4. Start the Development Server:

##### ○ Frontend:

```
cd frontend

npm start
```

##### ○ Backend:

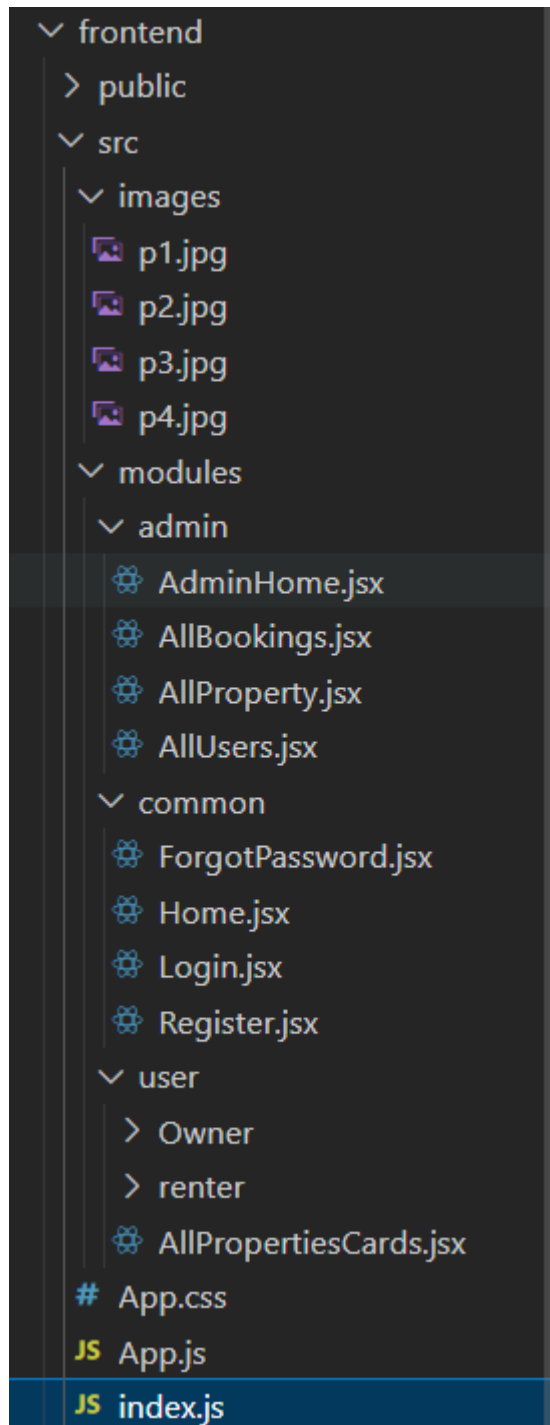
```
cd ../backend

npm start
```

The house rent app will be accessible at <http://localhost:3000>

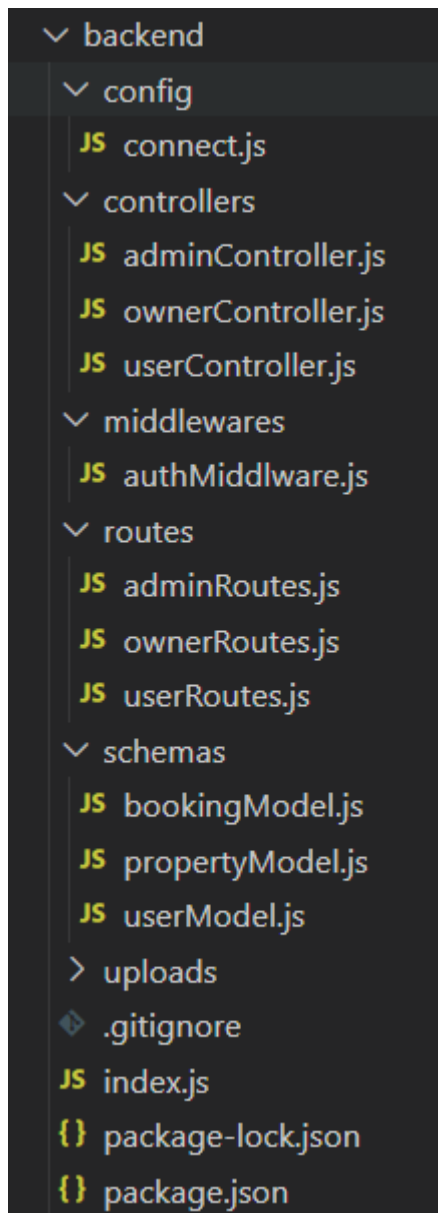
## Folder Structure

### 1. CLIENT (FRONTEND)



### 2. SERVER (BACKEND)





## Running the application

### Frontend:

```
cd frontend
```

```
npm start
```

### Backend:

```
cd ../backend
```

```
npm start
```

## API Documentation

### ADMIN:

1. Endpoint: GET /api/getallusers

Authentication: *Enabled*

Description: Fetch all users from the database.

Request Parameters: None

### Response:

```
{  
  "success": true,  
  "message": "All users",  
  "data": [/* Array of user objects */]  
}
```

2. Endpoint: POST /api/handlestatus

Authentication: *Enabled*

Description: Update a user's "granted" status in the database.

Request Parameters: {

"userid": "string",

"status": "string"

}

### Response:

```
{  
  "success": true,  
  "message": "User has been <status>"  
}
```

3. Endpoint: GET /api/getallproperties

Authentication: *Enabled*

Description: Fetch all properties from the database.

Request Parameters: None

**Response:**

```
{  
  "success": true,  
  "message": "All properties",  
  "data": [/* Array of property objects */]  
}
```

**4. Endpoint: GET /api/getallbookings**

Authentication: *Enabled*

Description: Fetch all bookings from the database.

Request Parameters: None

**Response:**

```
{  
  "success": true,  
  "data": [/* Array of booking objects */]  
}
```

**OWNER****1. Endpoint: POST /api/postproperty**

Authentication: *Enabled*

Description: Fetch all bookings from the database.

Request Parameters: {

```
  "userId": "string",  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "New Property has been stored"  
}
```

**2. Endpoint: GET /api/getallproperties**

Authentication: *Enabled*

Description: Fetch all properties owned by a specific user.

Request Parameters: {

"userId": "string"

}

**Response:**

{

"success": true,

"data": [/\* Array of properties owned by the user \*/]

}

3. Endpoint: GET /api/getallbookings

Authentication: *Enabled*

Description: Fetch all bookings associated with a specific owner.

Request Parameters: {

"userId": "string"

}

**Response:**

{

"success": true,

"data": [/\* Array of bookings associated with the owner \*/]

}

4. Endpoint: POST /api/handlebookingstatus

Authentication: *Enabled*

Description: Update the status of a booking and the availability of the associated property.

Request Parameters: {

"bookingId": "string",

"propertyId": "string",

"status": "string"

}

**Response:**

```
{  
  "success": true,  
  "message": "changed the status of property to <status>"  
}
```

5. Endpoint: DELETE /api/deleteproperty/:propertyid

Authentication: *Enabled*

Description: Delete a property from the database by its ID.

Request Parameters: {

```
  "propertyid": "string"  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "The property is deleted"  
}
```

6. Endpoint: PATCH /api/updateproperty/:propertyid

Authentication: *Enabled*

Description: Update the details of a property by its ID.

Request Parameters: {

```
  "userId": "string"  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Property updated successfully."  
}
```

**RENTER(USER):**

1. Endpoint: POST /api/register

Authentication: *None*

Description: Register a new user, check if the user already exists, and handle user creation with password hashing.

Request Parameters: {

```
"email": "string",  
"password": "string",  
"type": "string",  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Register Success"  
}
```

2. Endpoint: POST /api/login

Authentication: *None*

Description: Log in a user by validating their email and password, generating a JWT token upon successful authentication.

Request Parameters: {

```
"email": "string",  
"password": "string",  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Login success",  
  "token": "string",  
  "user": {  
    "_id": "string",  
    "email": "string",  
    "name": "string",  
  }  
}
```

3. Endpoint: POST /api/forgotpassword

Authentication: *None*

Description: Allow a user to reset their password by providing their email and new password. The password is hashed before being updated in the database.

Request Parameters: {

```
"email": "string",  
"password": "string",  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Password changed successfully"  
}
```

4. Endpoint: GET /api/getAllProperties

Authentication: *None*

Description: Fetch all properties from the database.

Request Parameters: None

**Response:**

```
{  
  "success": true,  
  "data": [/* Array of all properties */]  
}
```

5. Endpoint: POST /api/getuserdata

Authentication: *Enabled*

Description: Fetch a user by their userId and return the user data if found.

Request Parameters: {

```
"userId": "string"  
}
```

**Response:**

```
{  
  "success": true,  
  "data": {  
    "_id": "string",
```

```
"email": "string",  
"name": "string",  
}  
}
```

6. Endpoint: POST /api/bookinghandle/:propertyid

Authentication: *Enabled*

Description: Handle booking creation and status updates for a property.

Params: propertyid

Request Parameters: {

```
"userDetails": {  
  "fullName": "string",  
  "phone": "string"  
},  
"status": "string",  
"userId": "string",  
"ownerId": "string"  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Booking status updated"  
}
```

7. Endpoint: GET /api/getallbookings

Authentication: *Enabled*

Description: Handle booking creation and status updates for a property.

Params: propertyid

Request Parameters: {

```
"userDetails": {  
  "fullName": "string",  
  "phone": "string"
```



```
    },  
    "status": "string",  
    "userId": "string",  
    "ownerId": "string"  
  }  
}
```

**Response:**

```
{  
  "success": true,  
  "message": "Booking status updated"  
}
```

## Authentication and Authorization

### 1. Authentication

Authentication is the process of verifying a user's identity to allow access to the platform. The system uses **JWT (JSON Web Token)** for stateless authentication. The steps involved in the authentication Process includes,

a) **User Registration:**

- New users provide their details (name, email, password, type) during registration.
- Passwords are hashed using a secure hashing algorithm (e.g., bcrypt) before being stored in the database.

b) **User Login:**

- The user provides their email and password.
- The backend validates the email and password against the database.
- Upon successful validation, a JWT is generated and sent to the client.

c) **Token-Based Authentication:**

- The JWT contains the user's unique identifier (userId) and their role (type) encoded in its payload.
- The token is signed using a secret key to ensure its authenticity.
- The client stores the token (e.g., in local storage or a secure HTTP-only cookie) and includes it in subsequent requests for validation.

### 2. Authorization

Authorization ensures that users can only access resources and perform actions that align with their role.

**a) Role-Based Access Control (RBAC):**

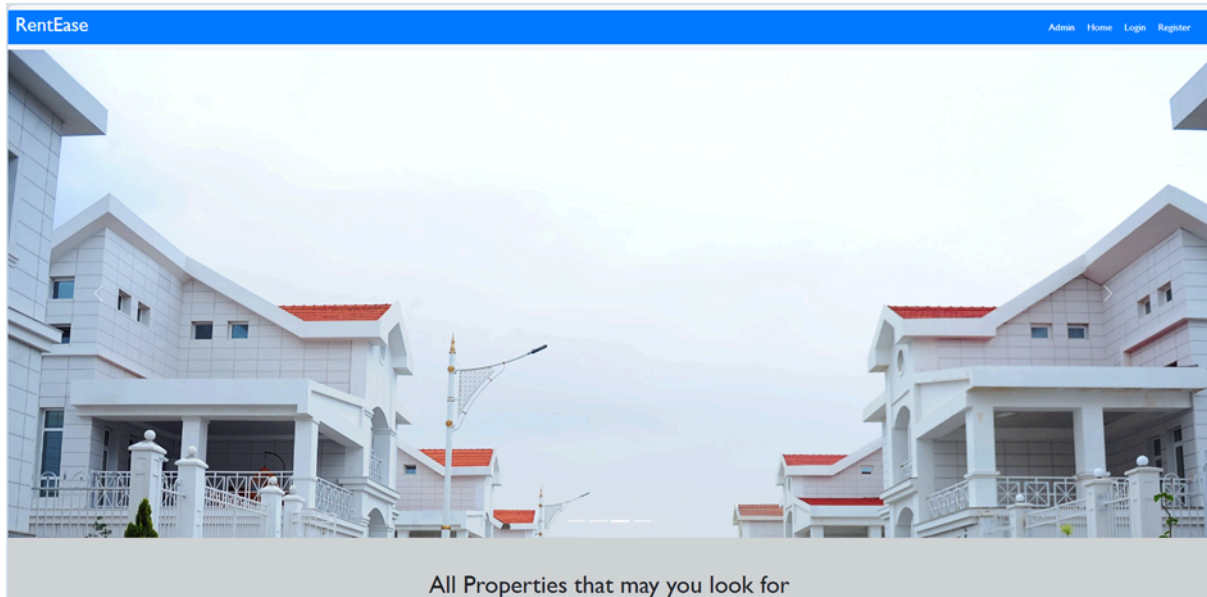
- Users have roles (e.g., tenant, property owner, admin), which determine their permissions.
- The type field in the JWT payload specifies the user's role.
- Middleware checks the user's role and grants or restricts access to specific routes or operations.

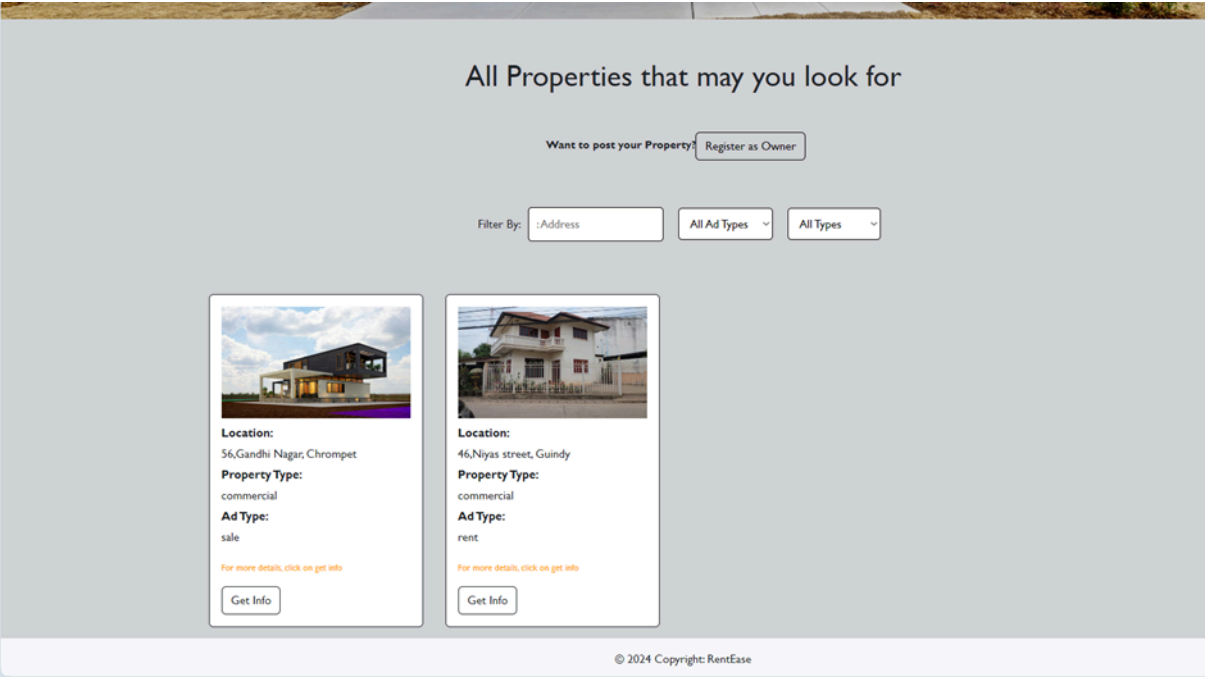
**b) Token Validation:**

- Every incoming request to protected routes includes the JWT in the Authorization header.
- Middleware decodes and verifies the token using the secret key:
  - If the token is valid, the user is allowed access, and their role is further checked for permissions.
  - If the token is invalid or missing, the user is denied access with an appropriate error response.

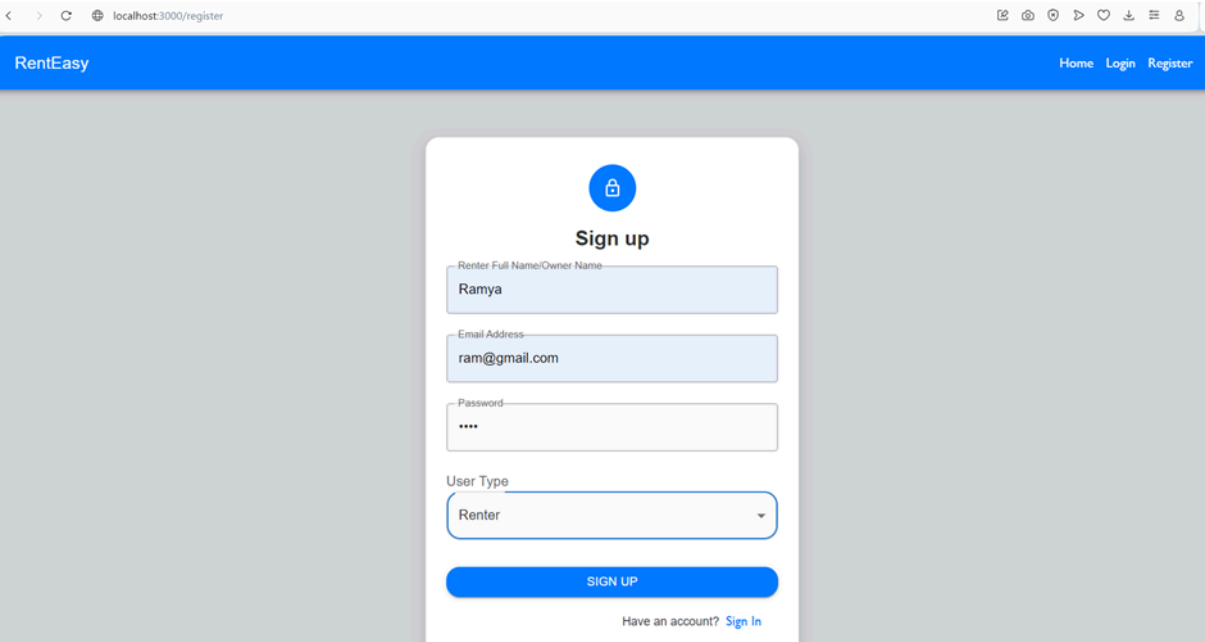
## User Interface

### HOME PAGE





## REGISTER AND LOGIN



localhost:3000/login

RentEasy Home Login Register

### Sign in

Username

ram@gmail.com

Password

\*\*\*\*

SIGN IN

[Forgot password?](#) [Don't have an account? Register](#)

localhost:3000/forgotpassword

Would you like the password manager to save the password for "localhost:3000"? Save

### Forgot Password?

Email Address

ram@gmail.com

New Password

\*\*\*\*

Confirm Password

\*\*\*\*

CHANGE PASSWORD

[Remembered your password? Login](#) [Don't have an account? Register](#)

# OWNER

- Add property

localhost:3000/ownerhome

Would you like the password manager to save the password for "localhost:3000"?

SaveNever

RentEase

Hi RamyaLog Out

ADD PROPERTY

ALL PROPERTIES

ALL BOOKINGS

Property type

Residential

Property Ad type

Rent

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

- View property

RentEase

Hi RamyaLog Out

ADD PROPERTY

ALL PROPERTIES

ALL BOOKINGS

Property ID	Property Type	Property Ad Type	Property Address	Owner Contact	Property Amt	Property Availability	Action
67375755119cf63106ee9711	commercial	sale	56,Gandhi Nagar, Chrompet	9876543221	2500000	Unavailable	<div>EditDelete</div>
673767d4119cf63106ee975c	commercial	rent	46,Niyas street, Guindy	9876543221	50000	Available	<div>EditDelete</div>

- View all bookings

ADD PROPERTY

ALL PROPERTIES

ALL BOOKINGS

Booking ID	Property ID	Tenant Name	Tenant Phone	Booking Status	Actions
67375f02119cf63106ee9740	67375755119cf63106ee9711	Ram	5678904567	booked	<div>Change</div>


RENTER

- Property listing:

Filter By:

All Ad Types

All Types



**Location:**  
56,Gandhi Nagar, Chrompet

**Property Type:**  
commercial


**Ad Type:**  
sale

**Owner Contact:**  
9876543221

**Availability:**  
Unavailable

**Property Amount:**  
Rs.2500000

Not Available



**Location:**  
46,Niyas street, Guindy

**Property Type:**  
commercial

**Ad Type:**  
rent

**Owner Contact:**  
9876543221

**Availability:**  
Available

**Property Amount:**  
Rs.50000

[Get More Info of the Property](#) [Get Info](#)

- Booking history:

localhost:3000/renterhome

Hi Ram Log Out

ALL PROPERTIES

BOOKING HISTORY

Booking ID	Property ID	Tenant Name	Phone	Booking Status
67375f02119cf63106ee9740	67375755119cf63106ee9711	Ram	5678904567	booked

ADMIN

- View all bookings

localhost:3000/adminhome

Hi Ram Log Out

ALL USERS

ALL PROPERTIES

ALL BOOKINGS

Booking ID	Owner ID	Property ID	Tenant ID	Tenant Name	Tenant Contact	Booking Status
67375f02119cf63106ee9740	67374d6e74c41556613840a9	67375755119cf63106ee9711	67375ee0119cf63106ee9739	Ram	5678904567	booked

- View all properties

Property ID	Owner ID	Property Type	Property Ad Type	Property Address	Owner Contact	Property Amt
67375755119cf63106ee9711	67374d6e74c41556613840a9	commercial	commercial	56,Gandhi Nagar, Chrompet	9876543221	2500000
673767d4119cf63106ee975c	67374d6e74c41556613840a9	commercial	commercial	46,Niyas street, Guindy	9876543221	50000

- View all users

User ID	Name	Email	Type	Granted (for Owners users only)	Actions
67374d6e74c41556613840a9	Ramya	ram@gmail.com	Owner	granted	UNGRANTED
67375ee0119cf63106ee9739	Ram	ram12@gmail.com	Renter		

## Testing Strategy

### 1. Performance Testing:

**Goal:** Assess app speed and scalability.

**Tools:** JMeter, Lighthouse.

**Example:** Test server response with multiple concurrent users.

### 2. Functional Testing:

**Goal:** Test user workflows like registration and bookings.

**Tools:** Cypress, Selenium.

**Example:** Confirm renters can book properties successfully.

### 3. Integration Testing:

**Goal:** Validate interaction between frontend, backend, and database.

**Tools:** Postman, Supertest.

**Example:** Verify renter inquiries are saved in the database.

## Demo

Demo Link: [Insert hosted app URL]

## Known issues

### 1. Limited Search Filters:

- Current search functionality doesn't include advanced filters such as pet-friendliness or nearby amenities.
- **Solution:** Expand the filtering options to include more criteria.

### 2. Slow Image Uploads:

- Property images upload slowly due to unoptimized file handling.
- **Solution:** Implement file compression or use a CDN for faster uploads and retrieval.

## Future Enhancements

### 1. Integrated Payment Gateway:

- Add functionality to process rental payments securely through integrated payment gateways like Stripe or PayPal.

### 2. Advanced Analytics Dashboard:

- Provide insights for property owners and admins, including user activity, rental trends, and property engagement statistics.

### 3. AI-Powered Recommendations:

- Use AI to recommend properties to renters based on their preferences and previous interactions.

### 4. Enhanced Messaging System:

- Include real-time chat with file-sharing capabilities between renters and owners.



## 5. **Push Notifications:**

- Notify users about new property listings, status updates, or expiring leases.