Full Stack Development with MERN

HouseHunt: Finding Your Perfect Rental Home

Team Members:

Pitta Lavanya – Frontend Development & UI Design

Naga Raju Neelam – Backend Development & API Development

Sheik Mohammad Umar – Database Design & Implementation

V N P L Meghamala Nunna – Testing & System Validation

Introduction

In today's fast-paced digital world, finding suitable housing efficiently has become a necessity. Traditional rental methods often involve manual searching, fragmented communication with owners, and a lack of real-time availability updates. To address these challenges, **HouseHunt: Finding Your Perfect Rental Home** was developed as a full-stack web application using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js).

HouseHunt provides a centralized platform where **Renters** can browse properties, apply filters based on their needs, and send inquiries directly to owners. The system enables **Owners** to manage their property listings (CRUD operations) and allows **Administrators** to verify owner credentials to maintain platform integrity.

Project Overview

Purpose:

The purpose of HouseHunt is to provide a digital solution for managing property rentals efficiently and securely. It eliminates manual inefficiencies by offering an automated, user-friendly, web-based platform.

Key Features:

Renter Features

- Secure registration and login.
- Property browsing with detailed descriptions and images.
- Advanced search filters (Price, Property Type, Location).
- Book appointments by selecting date and time.
- Inquiry submission and booking status tracking (Pending/Approved).

Owner Features

- Account registration and Admin approval request.
- Full CRUD operations (Add, Edit, Delete) for property listings.
- Manage booking inquiries and update property availability.

Admin Features

- Monitor overall platform activities
- Approve or reject account applications
- Maintain system security and user management.

Architecture

The application follows a Client-Server Architecture using the MERN stack.

Frontend Architecture

- Developed using React.js, the frontend utilizes a component-based structure for a responsive and modern UI.
- **Communication: Axios** is used to handle RESTful API calls to the backend.
- **State Management:** React Hooks and Moment.js for date/time formatting.

- **Libraries:** Bootstrap, Material UI, Ant Design (Antd), and MDBootstrap.

Backend Architecture

The backend is built using Node.js and Express.js following the MVC pattern.

- **Authentication:** JWT (JSON Web Tokens) and bcryptjs for secure password hashing.
- **File Handling: Multer** is used for handling property image uploads.
- **Middleware:** Custom authentication middleware to protect private routes.

Database Architecture

MongoDB serves as the NoSQL database with three primary collections:

- **Users:** Stores credentials, roles (Renter/Owner/Admin), and profiles.
- **Property:** Contains property details, images, and owner references (Foreign Key: userID).
- **Booking:** Tracks the relationship between properties, renters, and owners, including status updates.

Setup Instructions

Prerequisites

1.Software Requirements

- **Node.js** (v16 or higher)
- **npm** (comes with Node.js)
- **MongoDB Atlas account** (for cloud database connection)
- **Git** (for cloning the repository)
- **Code Editor** (e.g., VS Code)

2. Optional Tools

- Postman (for API testing)
- MongoDB Compass (for database visualization)

Installation

Step 1: Clone the Repository

Open terminal or command prompt and run:

```
git clone <repository-link>
```

Navigate into the project directory:

```
cd docspot
```

Step 2: Install Backend Dependencies

Navigate to the server folder:

```
cd server
```

Install required dependencies:

```
npm install
```

Step 3: Configure Environment Variables

Inside the server folder, create a `.env` file and add the following:

```
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_secret_key
```

Replace:

- `your_mongodb_connection_string` with your MongoDB Atlas connection string.
- `your_secret_key` with a secure random string.

Step 4: Install Frontend Dependencies

Open a new terminal and navigate to the client folder:

```
cd client
```

Install frontend dependencies:

```
npm install
```

Step 5: Start the Application

Start Backend Server:

```
cd server
npm start
```

Backend runs on:

```
http://localhost:5000
```

Start Frontend Application:

```
cd client
npm start
```

Frontend runs on:

```
http://localhost:3000
```

Folder Structure

Client Folder Structure

```
client/
│
├── public/
│   └── index.html
│
├── src/
│   ├── components/
│   ├── pages/
│   ├── redux/ (or context/)
│   ├── hooks/
│   ├── utils/
│   ├── assets/
│   ├── App.js
│   └── index.js
│
└── package.json
```

Server Folder Structure

```
server/
│
├── config/
├── controllers/
├── models/
├── routes/
├── middleware/
├── utils/
├── uploads/ (if file upload is used)
├── server.js
└── package.json
```

Running the Application

Starting the Backend Server

1. Open a terminal.
2. Navigate to the server directory:

```
cd server
```

3. Start the backend server:

```
npm start
```

If configured correctly, the backend will run on:

`http://localhost:5000`

Starting the Frontend Server

1. Open a new terminal window.
2. Navigate to the client directory:

```
cd client
```

3. Start the React application:

```
npm start
```

The frontend application will run on:

`http://localhost:3000`

API Documentation

Base URL:

http://localhost:5000/api

1. Authentication & User APIs

These endpoints handle the security and identity management for Renters, Owners, and Admins.

User Registration: Create a new account as a Renter or Owner.

User Login: Authenticates credentials and generates a JWT for session management.

Profile Management: Retrieve and update user-specific details such as contact information

2. Property APIs (Owner & General)

These endpoints manage the property listings stored in the **Property** collection.

Add Property: Allows verified Owners to upload new listings (Type, Amount, Images, Address).

Get All Properties: Public endpoint for Renters to browse available listings.

Update/Delete Property: CRUD operations for Owners to manage their own listings.

Filter Properties: Search for homes based on rent range, location, or property type.

3. Booking & Inquiry APIs

These endpoints manage the interaction between Renters and Owners stored in the **Booking** collection.

Send Inquiry/Book: Renters submit their details and interest for a specific property.

Get Bookings: Owners view inquiries for their properties; Renters view their sent requests.

Update Booking Status: Owners approve or reject a "pending" booking request.

4. Admin APIs

Endpoints restricted to the Admin role for platform governance

Manage Users: View all registered Renters and Owners.

Owner Approval: Approve legitimate user requests to upgrade to an "Owner" account to post properties.

System Activity: Monitor overall property listings and booking trends.

All protected routes require Authorization: Bearer <JWT_TOKEN>

Authentication

Authentication is implemented using JWT and bcrypt.

Core Security Features

Password Hashing: Passwords are encrypted with **bcryptjs** before storage in MongoDB to ensure data privacy

JWT Workflow: A secure token is generated upon login, stored on the client side, and attached to request headers for all protected actions.

Token Verification: Custom middleware intercepts requests to verify signatures and enforce user sessions.

Role-Based Control: The system enforces specific permissions for **Renters**, **Owners**, and **Admins** using HTTP 401/403 status codes for unauthorized access.

Protected Operations

Renters: Securely browse and book properties.

Owners: Perform CRUD operations on listings only after verification.

Admins: Approve owners and manage platform-wide integrity.

User Interface

The UI is responsive, modern, and role-based.

Includes:

- Login Page
- Registration Page
- Renter Dashboard
- Property Inquiry Form
- Booking History
- Owner Dashboard
- Admin Panel

Designed using React, Bootstrap, and Material UI.

Testing

Manual and functional testing was performed.

Frontend Testing

- Form validation
- Navigation testing
- Role-based rendering
- Property workflow

Backend Testing

- API response validation
- JWT verification
- CRUD operations
- Error handling

Database Testing

- Data storage validation
- Password hashing verification
- Relationship consistency

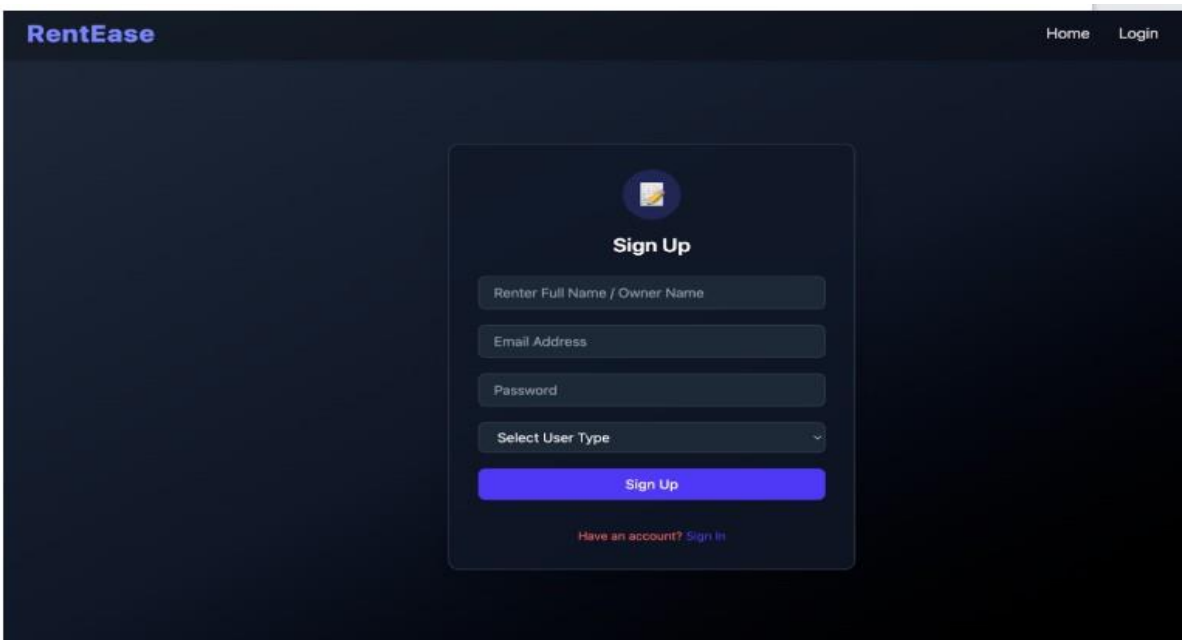
Integration Testing

- Axios API communication
- Real-time UI updates
- Data consistency

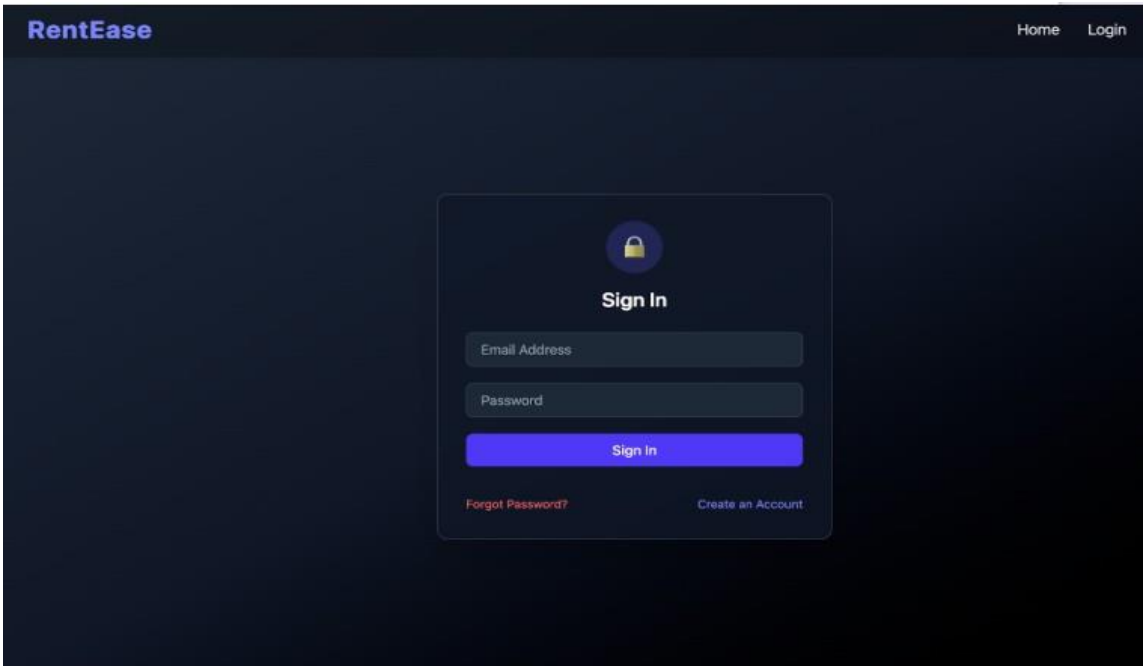
All core functionalities were validated successfully.

Screenshots

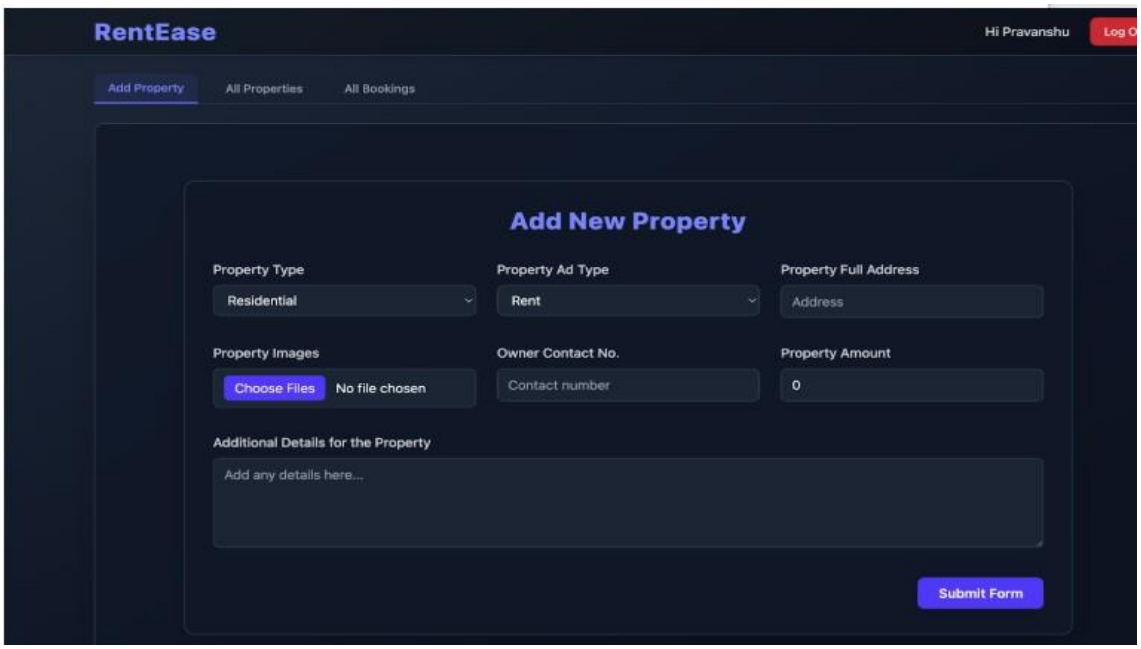
Login and register page



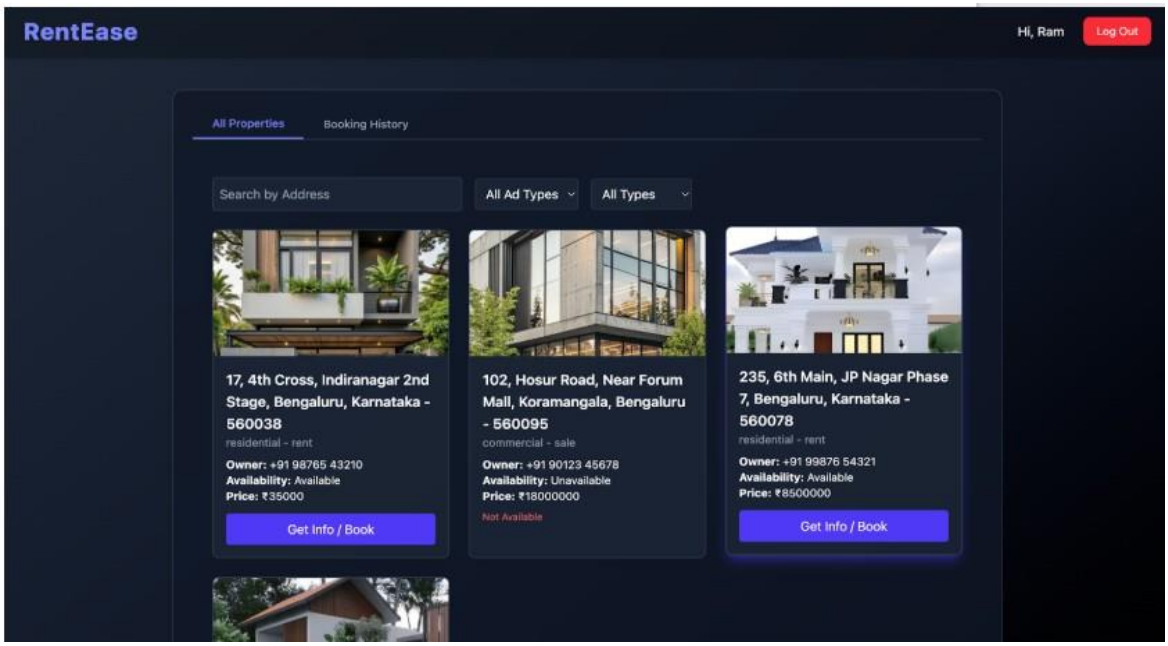
Admin Panel



Owner panel



Tenant Panel



Known Issues

No Real-Time Push Notifications:

The system does not support real-time push notifications; users must manually check their dashboard for status updates like booking confirmations or cancellations as WebSockets or push services are not yet implemented.

No Email or SMS Alerts:

Notifications are limited to in-app updates; appointment confirmations and status reminders are not sent via external communication channels like email or SMS.

No Online Payment Integration:

The platform lacks a payment gateway; rental deposits or consultation fees cannot be processed online and must be handled offline.

Basic Error Messages:

Some system and backend validation errors return generic messages, which could be made more descriptive to improve the debugging process and user experience.

No Load Balancing or Caching:

Advanced performance optimization techniques like caching or load balancing are not implemented, which may lead to performance degradation under heavy user traffic.

Manual Testing Only:

The project relies entirely on manual testing; the absence of automated frameworks like Jest or Mocha may limit long-term maintainability.

No Token Refresh Mechanism:

JWT authentication is implemented without refresh tokens; users must log in again manually once a token expires, which can interrupt the user experience.

Limited File Validation:

File validation for document or image uploads is basic; strict file type checking, size limits, and secure cloud storage integration are not fully realized.

Minor Mobile UI Adjustments:

While the application is responsive, some layout and spacing elements require further optimization for a more polished experience on smaller mobile screens.

Future Enhancements

- **Online Payment Integration**: Integrate secure gateways (credit/debit, UPI) to allow renters to pay security deposits or booking fees online.

  Reduces vacancy rates by securing commitments.

  Provides transparent, secure transaction record

  Improves overall user convenience.

- **Email and SMS Notifications**: Implement automated alerts for booking confirmations, reminders, and status updates via external APIs.

- **Virtual Property Tours**: Add telemedicine-inspired support for remote viewings through video calling integration and secure online meeting rooms.

- **Rating and Review System**: Allow renters to rate properties and owners to improve transparency and platform trust.

- **Advanced Filtering and Search**: Enhance property discovery with location-based filtering (GPS), interactive availability calendars, and specific amenity ranges.

- **Admin Analytics Dashboard**: Develop a detailed panel showing active user counts, booking statistics, and monthly rental trends.

- **Mobile Application**: Develop a cross-platform app using **React Native** or **Flutter** to improve accessibility and engagement.

- **Cloud File Storage Integration**: Utilize cloud services for property image and document uploads to improve scalability and security.

- **Automated Testing Implementation**: Introduce **Jest** for frontend and **Mocha/Chai** for backend testing to improve code reliability.

- **Token Refresh Mechanism**: Implement refresh tokens to improve session management and prevent frequent login interruptions.

These enhancements will evolve the platform into a more scalable, production-ready system capable of handling large user bases and real-world real estate requirements.