House Hunt: Finding your Perfect Rental Home Using MERN Stack

Project Documentation

1. Introduction

Project Title: House Hunt: Finding Your Perfect Rental Home

Team ID: LTVIP2025TMID54354

Team Size: 4

Team Leader: Kondala Somasekhar

Team member: Kudupudi.Divya Sri

Team member: Kanala.Geetha Pranathi

Team member: Jyothirganesh Kanuboyina

2. Project Overview

Purpose: The purpose of **HouseHunt** is to simplify and enhance the home rental process by providing a centralized, user-friendly digital platform built on the MERN stack (MongoDB, Express.js, React, Node.js).

It aims to:

- Connect property owners with renters directly, eliminating the need for third-party brokers.
- Allow users to **search**, **filter**, **and book** rental properties based on location, budget, type, and facilities.
- Empower property owners to **list and manage** their rental properties with images, categories, and availability status.
- Provide **admins** with control over user management, property listings, and platform monitoring.

Ensure a **secure**, **scalable**, **and role-based** architecture for seamless interaction across all user types.

Ultimately, HouseHunt addresses the challenges of traditional property hunting by combining modern web technologies with an intuitive user experience.

Features:

HouseHunt provides a robust set of features to enhance the rental property experience for renters, owners, and admins. The following core functionalities are implemented:

- **1** User Roles & Authentication
 - Role-based access for Renter, Owner, and Admin
 - JWT-based login and registration system
 - Secure user sessions and protected routes

Property Listings

• Owners can add, edit, and delete listings

Properties include:

```
    Type (Apartment, Villa, etc.) ○
    Category (1BHK, 2BHK, etc.) ○
    Facilities (Wi-Fi, Parking, Gym) ○
    Price, Location, and Availability ○
    Image uploads with preview
```

- Advanced Search & Filtering
 - Renters can search properties by: Location Property type/category Price range Facilities
 - Real-time filtered display of matching results
- 31 Booking & Inquiry System
 - Renters can book a property or send inquiries
 - Booking records are stored and managed per user
 - Owners receive requests and manage booking status
- **% Admin Dashboard View and manage: All users All**

 $properties \ \circ$

Booking records

- Approve or remove listings
- Ensure platform quality and safety

Responsive UI

- Built with React and Tailwind CSS for mobile and desktop compatibility
- · Clean navigation, interactive forms, modals, and tooltips

Developer-Friendly

- Modular folder structure for scalability
- RESTful APIs using Express.js
- Mongoose schema for MongoDB

3. Architecture

HouseHunt is built using the **MERN stack** — MongoDB, Express.js, React, and Node.js — and follows a **modular, scalable, and role-based architecture**. The system is structured into four major layers:

1. Frontend Layer (React)

- Developed using **React.js** for component-based UI
- Routing handled via React Router
- Styled with Tailwind CSS
- Provides pages and components for:
 - o Registration &

Login o Property

Listings o Booking

Forms o Admin

Dashboard

• Communicates with backend via Axios over REST APIs

2. Backend Layer (Node.js + Express.js)

Express handles all HTTP requests

- □ RESTful APIs for:
 - User Authentication o

Property CRUD

operations o Booking requests

- · Middleware for role-based authorization
- Error handling and validation included

3. Database Layer (MongoDB with Mongoose) Three primary collections:

users – stores user info, role, contact o
 properties – details of rental listings o
 bookings – rental inquiries and
 statuses

Relationships are modeled using references (e.g., userID in properties)

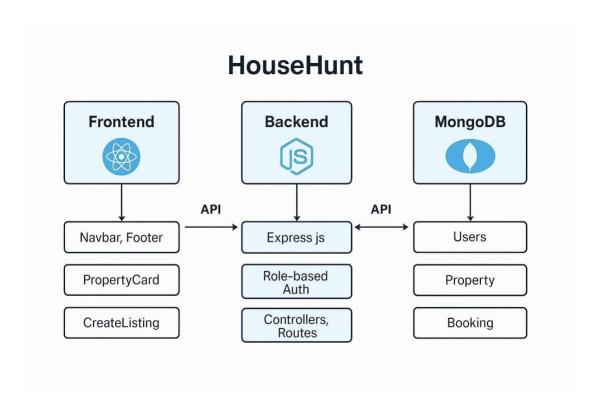
4. Authentication & Authorization

- **JWT tokens** issued during login
- Middleware checks user roles for access (Renter, Owner, Admin)
- Secure route protection for CRUD and booking actions

Data Flow Overview

- 1. User logs in \rightarrow JWT token issued
- 2. Renter searches listings \rightarrow API filters listings from DB
- 3. Owner adds property → Data sent to backend & stored
- 4. **Renter sends booking request** → Booking stored in DB
- 5. Admin reviews listings/bookings via dashboard

Architectural Diagram:



4. Setup Instructions

Prerequisites

- Node.js (v18+), npm
- · MongoDB Atlas account
- Git (for version control)
- VS Code or similar editor

1. Backend Setup

Navigate to the backend/ folder and run:

npm install

Create a .env file with:

MONGO_URI=your_mongodb_uri

JWT_SECRET=your_jwt_secret

PORT=5000

Then start the server:

npm run dev

2. Frontend Setup

Navigate to frontend/ and run

npm install npm run dev

The frontend will launch at:

http://localhost:5173

3. MongoDB Atlas

- · Create a free cluster

 Add a database user
- · Whitelist your IP
- Use the connection string in the backend .env

4. Configuration Tips

- Enable CORS in backend using:
- app.use(cors());
- Use tools like Postman to test API endpoints

5. Folder Structure

The HouseHunt project follows a clear and modular structure, separating concerns between frontend and backend:

bash CopyEdit

```
# Utility functions (e.g., validations)
      — utils/
                    # Context Providers for auth, state, etc.
     · context/
      —— App.js
                          # Root component with routes
      index.js
                          # React entry point
                        # Node.js + Express Backend
    - server/
                          # Mongoose schemas (User, Property,
     — models/
Booking)
 routes/
                         # Route definitions (userRoutes,
propertyRoutes)
                          # Logic for each API endpoint
      — controllers/
       – middleware/
                            # JWT auth, error handling
      — config/
                          # DB connection and environment setup
       – server.js
                         # Entry point for backend
                         # Environment variables
       - .env
    - .gitignore
    - README.md
    - package.json
                           # Project dependencies
```

▶ Running the Application

To run the HouseHunt application locally

✓ Prerequisites

Make sure the following are installed:

- Node.js (v16 or higher)
- npm or yarn
- MongoDB (Local or Atlas cloud instance)
- Git (for cloning the project)

Backend Setup (Node.js + Express)

1. Navigate to the server directory:

cd server

2. Install dependencies:

npm install

3. Set environment variables:

MONGO_URI=your_mongodb_connection_string

JWT SECRET=your jwt secret

PORT=5000

3. Start the server

npm run dev

The backend will run on: http://localhost:5000

- **③** Frontend Setup (React)
 - 1. Navigate to the client directory:

cd client

2. Install dependencies:

npm install

3. Start the development server:

npm start

The frontend will run on: http://localhost:3000

5. API Documentation

The backend of HouseHunt uses RESTful APIs built with Node.js and Express.js. It includes endpoints for user authentication, property management, and booking operations. All data is stored and retrieved from MongoDB using Mongoose models.

Authentication APIs

Method	Endpoint	Description	Access
POST	/api/register	Register a new user	Public
POST	/api/login	Login and return JWT token	Public
GET	/api/profile	Get logged-in user profile	Authenticated

Property APIs

Method	Endpoint	Description	Access
POST	/api/properties	Create a new property	Owner Only
		listing	
GET	/api/properties	Get all property listings	Public
GET	/api/properties/:id	Get property by ID	Public
PUT	/api/properties/:id	Update property by ID	Owner Only
DELETE	/api/properties/:id	Delete property	Owner/Admin

31 Booking APIs

Method	Endpoint	Description	Access
POST	/api/bookings	Create a new	Renter Only
		booking request	
GET	/api/bookings/user/:id	Get bookings by	Renter Only
		user ID	
GET	/api/bookings/property/:id	Get bookings for	Owner/Admin
		a specific	
		property	
PUT	/api/bookings/:id	Update booking	Owner/Admin
		status	
		(approve/deny)	

Notes

• Authorization is handled using JWT in the Authorization header.

- Endpoints marked "Owner Only" or "Admin" require appropriate role-based middleware.
- Responses are returned in JSON format.
- All sensitive operations are protected via middleware and token verification.

6. User Interface

The **frontend** of HouseHunt is developed using **React.js** and styled with **Tailwind CSS**, offering a responsive and clean interface for three main user roles: **Renter**, **Owner**, and **Admin**.

Main Application Layout

- **Navigation Bar**: Displays brand, login/register (or profile) links, and role-specific menu items.
- Footer: Contains contact info, quick links, and copyright.
- Sidebar (Admin only): Links to user/property/booking management.

Page Structure

Page	Description
Home	Introduction to platform, highlights key features, and call-to-action
Register / Login	Forms with role selection (Renter / Owner) and input validation
Property Listings	Displays cards for available properties with filters and search bar
Property Details	Full view with images, location, price, facilities, and booking form
Create/Edit Listing	Form for owners to add/edit properties with image upload and dropdowns
My Listings	Owner dashboard for managing their posted properties
My Bookings	Renter dashboard showing booking requests and status

Admin	View and manage users, properties, and bookings in
Dashboard	tabular form

Features and Technologies React Components: • Property Card, Navbar, Footer, Sidebar, Booking Form State

Management: • Context API for auth and user session Form

Handling: • Controlled components with real-time validation
Notifications: • Success/error toasts using react-toastify

Image Upload: • Preview before upload, optionally linked to Cloudinary Routing:

Handled via React Router for SPA navigation

Responsiveness

- Tailwind CSS ensures fully responsive design for mobile, tablet, and desktop.
- Uses CSS grid and flexbox for layout adaptability.

7. Testing

A combination of unit testing, integration testing, and manual user acceptance testing (UAT) was applied to ensure that HouseHunt is functional, secure, and user-friendly.

✓ Unit Testing

- ☐ Tools Used: Jest, Supertest (for backend) ☐ Tested Components:
 - API routes and controllers (register, login, property CRUD)
 Utility functions (e.g., JWT generation, input validation)
 - o Mongoose models for schema validation

Example:

```
describe('User Registration', () => { it('should register a
user with valid input', async () => { const res = await
request(app).post('/api/register').send({ name: 'Test
User', email: 'test@example.com', password:
'123456',
    role: 'renter'
    });
    expect(res.statusCode).toBe(201);
});
```

☐ Integration Testing □ APIs

tested using Postman Derified end-to-end flows:

User registration → Login → JWT issuance ○ Owner
 creating listings → Renter viewing → Booking ○ Admin
 accessing protected dashboard routes

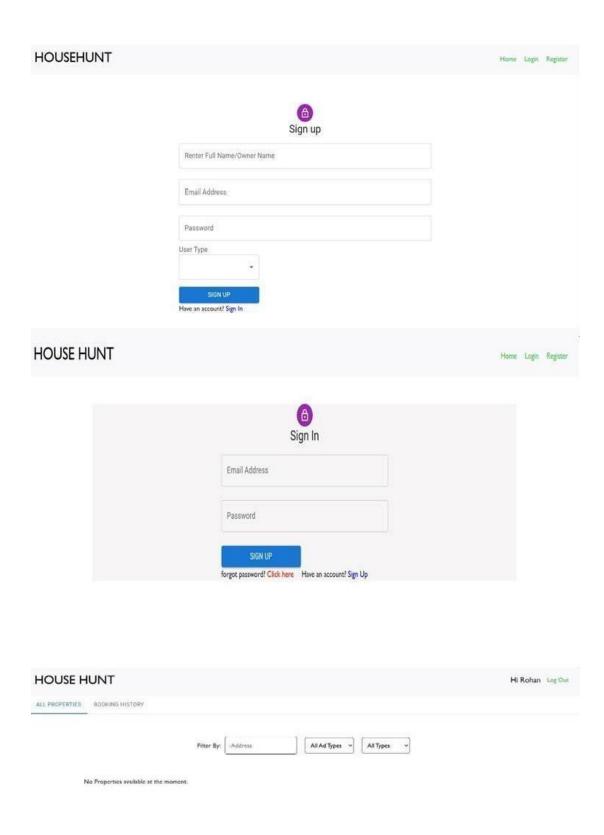
User Acceptance Testing (UAT)

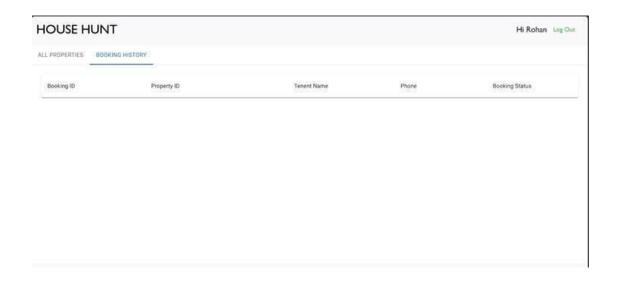
- Conducted across all three roles:
 - Renter: Book property, view status o Owner:
 Add/Edit/Delete property, view bookings o Admin:
 Manage users, view analytics
- Feedback loop with test users to refine UI and functionality

⚠ Error Handling

- All backend routes use try-catch blocks and centralized error middleware
- Frontend catches and displays network/API errors via toast notifications
- Includes 401/403 status handling for unauthorized or forbidden access

8. Screenshots





9. Known Issues

1. Image Upload is Local (No Cloud Storage)

Property images are stored locally, which may not scale well for production environments.

Cloud storage services like Cloudinary or AWS S3 are recommended for better file handling and URL-based retrieval.

2. No User Password Recovery

Forgot password functionality is not yet implemented.

Users currently have no way to reset their credentials if lost.

3. Payment Gateway Not Integrated No

rent payment or deposit system exists yet.

Future versions may integrate Stripe or Razorpay.

4. No Real-Time Booking Confirmation

Booking requests are stored in the database but don't trigger notifications or real-time updates.

Notifications via email/SMS are recommended for future iterations.

D 5. Admin Dashboard Is Minimal

Admin functionalities are limited to viewing data.

There's no analytics or graphical dashboard for insights like most rented areas or monthly activity.

6. Limited Mobile Optimization

While the app is responsive, some components (like modals and tables) need better optimization for small-screen devices.

10. Future Enhancements

1. Payment Gateway Integration

Integrate Stripe or Razorpay to allow renters to pay deposits or monthly rent securely online.

(1) 2. Email & SMS Notifications

- Notify owners and renters about new bookings, approvals, or cancellations.
- Reminders for upcoming rent dues or property expirations.

3. Mobile App (React Native)

 Develop a cross-platform mobile app for Android/iOS to allow users to manage their listings and bookings on-the-go.

4. Recommendation System

- Use machine learning to suggest properties based on:
 - User behavior
 Preferred locations
 Price range and amenities

5. In-App Chat System

• Implement real-time messaging between renters and property owners to enhance communication.

D 6. Advanced Admin Dashboard

• Add analytics and reporting tools for:
o Monthly activity
o Active listings by region
o Booking trends and user engagement

7. Map Integration

• Integrate **Google Maps API** for visual location selection during listing creation and for property previews.

8. Role-Based Access Control (RBAC)

• Improve access layers with fine-grained permissions for different user roles, including sub-admins or agents.

2 9. Multi-Listing Support

 Allow owners to add multiple units under the same building/property.