Full Stack Development Documentation

Project Title: House Hunt: Finding Your Perfect Rental Home

:

Submitted by
Team Id: **LTVIP2026TMIDS87052**

Table of Contents:

1. Introduction

In today's technology-driven environment, traditional processes like renting a home are being redefined through digital innovation. The process of finding rental properties—once is dominated by manual searches, broker involvement, and word-of-mouth—often proves inefficient, time-consuming, and lacking in transparency. To address these challenges, this project introduces a full-stack web application titled **"House Hunt – Finding Your Perfect Rental Home."**

This application is built using the **MERN stack**—**MongoDB**, **Express.js**, **React.js**, and **Node.js**—and is designed to create a streamlined platform where **tenants**, **property owners**, and **administrators** can interact seamlessly. The system enables users to register based on their roles, browse and list properties, book rental homes, and manage platform activities efficiently.

By incorporating modern web technologies, secure authentication mechanisms, and responsive user interface design, **House Hunt** offers a centralized solution that eliminates the need for intermediaries while enhancing accessibility, security, and ease of use.

1.1 Project Title:

House Hunt – Finding Your Perfect Rental Home

The title "House Hunt – Finding Your Perfect Rental Home" reflects the core objective of the project — to assist users in efficiently locating, listing, and managing rental properties through a centralized digital platform. It represents a real-time solution tailored for tenants, property owners, and administrators to handle the complexities of the rental process without the traditional challenges of manual searching and broker dependency.

The phrase "House Hunt" emphasizes the action-oriented nature of the application — searching, browsing, and shortlisting homes — while "Finding Your Perfect Rental Home" highlights the application's goal of delivering personalized, filter-based property discovery tailored to user preferences.

This title was chosen to be simple, descriptive, and relatable for users and stakeholders, making it easy to identify the purpose and value of the project at first glance.

1.2 Team Members

The development of this project was a collaborative effort by a team of four members, each contributing to specific areas of the MERN stack application development. The division of responsibilities was done to ensure smooth progress across all phases—design, development, testing, and documentation.

Team Composition

Name	Role
Janani	Frontend Developer & UI/UX Designer
Gayathri	Backend Developer
Reshma	Testing & Bug Fixing
Srija	Documentation & Project Coordination

2. Project Overview

2.1 Purpose

The primary purpose of the *House Hunt – Finding Your Perfect Rental Home* application is to streamline the property rental process by creating a centralized, digital platform for users to search, list, and manage rental properties. Traditional methods of house hunting often involve time-consuming processes, lack of updated information, broker dependency, and inefficient communication.

This project aims to solve these challenges by:

- Providing a role-based platform for renters, property owners, and administrators.
- Ensuring secure access and smooth interaction through modern web technologies.
- Enabling renters to filter properties based on location, price, and amenities.
- Empowering property owners to manage listings and approve booking requests.
- Allowing admins to oversee the platform, verify property details, and manage user activities.

Overall, the application is designed to deliver transparency, reliability, and user convenience in the property rental ecosystem.

2.2 Key Features

The application offers the following core features:

Feature	Description
User Registration & Login	Role-based registration for renters, owners, and admin with secure authentication.
Property Listing (Owner)	Owners can list properties with images, descriptions, price, and amenities.
Property Search (Renter)	Renters can browse and filter properties based on city, price, and more.
Booking System	Renters can send booking requests for listed properties.
Booking Approval (Owner/Admin)	Owners/admins can accept or reject booking requests.
Admin Dashboard	Admins can monitor platform activity, approve listings, and manage users.
Responsive UI	Clean and responsive interface compatible with various screen sizes.
JWT Authentication	JSON Web Token-based secure login session management.
MongoDB Database	Data is stored securely in a NoSQL format, enabling fast access and scalability.
Error Handling & Alerts	System alerts and validations for form inputs, errors, and confirmations.

3. Architecture

The technical architecture of the **House Hunt – Finding Your Perfect Rental Home** project is built upon the widely used **MERN stack**, structured around a **client-server model**. The frontend (client) interacts with users, while the backend (server) manages logic, data storage, and communication. This modular and layered architecture allows for ease of development, scalability, and future extensibility.

3.1 Frontend Architecture (React.js)

The frontend of the **House Hunt** application is developed using **React.js**, a powerful JavaScript library that enables the creation of fast, interactive, and dynamic user interfaces. The frontend serves as the client-side of the application and is responsible for rendering UI elements, capturing user input, and handling all visual interactions.

Key Features:

- **Component-Based Architecture**: Reusable React components (e.g., Header, Footer, PropertyCard, SearchFilter) improve modularity and maintainability.
- **Routing**: Implemented using **React Router DOM** to manage multiple views such as Home, Login, Register, Property Listings, and Admin Panel.
- **State Management**: Uses React's useState, useEffect, and optionally Context API to manage component-level and shared state.
- **API Integration**: **Axios** is used for making HTTP requests to backend APIs for user login, property fetching, bookings, etc.
- **Responsive Design**: Layouts adapt seamlessly across desktops, tablets, and mobile devices using **CSS3**, **Bootstrap**, and **Ant Design**.

3.2 Backend Architecture (Node.js & Express.js)

The backend of the application is built using **Node.js** and **Express.js**, which together handle business logic, data operations, authentication, and routing. The backend exposes a set of **RESTful APIs** that interact with the frontend and database.

Core Functionalities:

- **API Routing**: Defined using Express Router (e.g., /api/users, /api/properties, /api/bookings)
- **Authentication**: Implemented using **JWT (JSON Web Token)** for session management and **bcryptjs** for secure password hashing.
- **Middleware**:
    - authMiddleware.js → Protects private routes

- errorHandler.js → Handles server errors gracefully
- **Role-Based Access**: Separate logic for renters, owners, and admins (e.g., owners can add listings, admins can approve them).
- **File Uploads**: Uses **Multer** for handling image uploads (property photos).

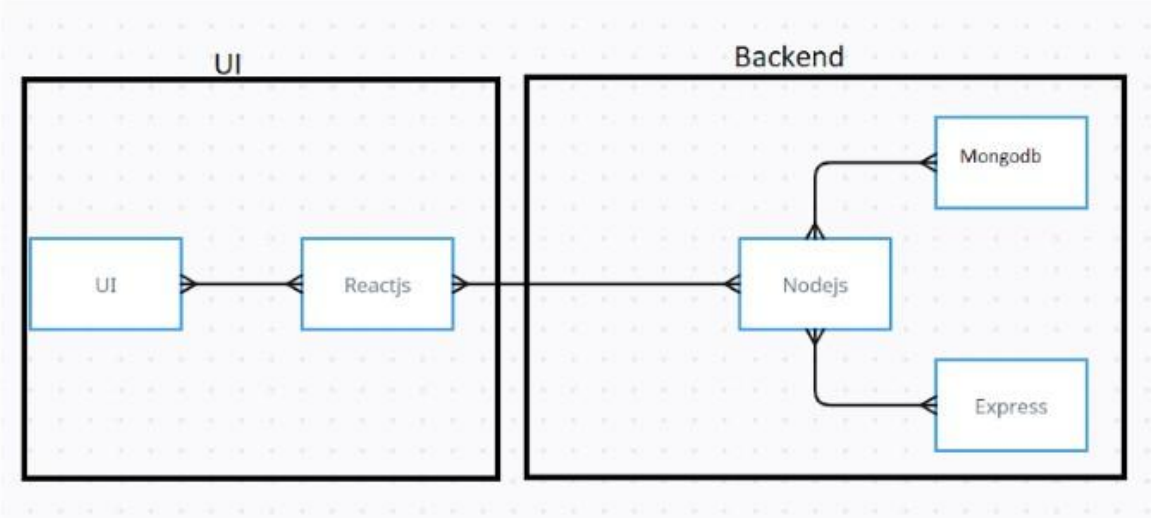
3.3 Database Design (MongoDB with Mongoose)

The database used for this application is **MongoDB**, a flexible NoSQL database well-suited for storing JSON-like data structures. **Mongoose**, an ODM (Object Data Modeling) library, is used to define and interact with schemas.

Collections and Their Roles:

1. **Users Collection**
   - Fields: name, email, password (hashed), role (renter/owner/admin)
   - Stores login credentials and user roles
2. **Properties Collection**
   - Fields: title, description, location, price, ownerId, images[], status
   - Represents rental property listings created by owners
3. **Bookings Collection**
   - Fields: propertyId, renterId, bookingDate, status (pending/approved/rejected)
   - Tracks booking requests and their approval status

TECHNICAL ARCHITECTURE



4. Setup Instructions

The following section provides a complete, step-by-step guide to setting up the House Hunt – Finding Your Perfect Rental Home application on a local development environment. The setup involves configuring both the frontend (React.js) and the backend (Node.js + Express.js), connecting to MongoDB, and preparing the necessary environment variables.

4.1 Prerequisites

Before setting up the House Hunt MERN application, ensure your development environment includes the following software and tools. These are essential for running both frontend and backend components smoothly.

Tool / Software	Purpose	Download / Notes
Node.js (v14 or above)	JavaScript runtime environment for backend & React tooling	https://nodejs.org/en/download
npm	Node package manager to install frontend/backend libraries	Comes bundled with Node.js
MongoDB	NoSQL database for storing all user/property/booking data	https://www.mongodb.com/try/download/community
Git	Version control system to manage project repository	https://git-scm.com/downloads
MongoDB Compass	GUI to view and test your MongoDB database	https://www.mongodb.com/products/compass
Visual Studio Code	Code editor (IDE) for writing and debugging source code	https://code.visualstudio.com/
Postman *(optional)*	API testing tool to verify backend endpoints	https://www.postman.com/downloads/
Nodemon *(optional)*	Auto restarts the backend server on file changes	Install via npm install -g nodemon

4.2 Installation Steps

Follow these steps to install all required dependencies and get the application running locally.

Step 1: Clone the Project Repository

Open your terminal or command prompt and run:

git clone https://github.com/your-name/house-rent.git

cd house-rent

You should now see the following folders:

house-rent/

├── frontend/ → React frontend application

└── backend/ → Node.js backend server with API logic

Step 2: Install Frontend Dependencies

cd frontend

npm install

This will install all React-based packages:

- axios – to make API calls
- react-router-dom – for page navigation
- bootstrap, antd, material-ui – for responsive design
- moment.js – for time/date handling
- mdb-react-ui-kit, react-bootstrap – for extra UI components

Step 3: Install Backend Dependencies

cd ../backend

npm install

This will install backend libraries such as:

- express – server framework
- mongoose – database interaction
- cors – to enable frontend-backend communication
- bcryptjs – password encryption
- jsonwebtoken – user authentication
- dotenv – to manage sensitive config variables
- multer – file/image upload handling
- moment – formatting dates
- nodemon *(optional)* – for auto server reloads during development

4.3 Environment Configuration

In your **backend folder**, create a new file named .env. This file stores environment variables securely and allows different configurations for dev vs production.

- Here's what to include:
- env
- PORT=5000
- MONGO_URI=mongodb://localhost:27017/househunt
- JWT_SECRET=yourSuperSecretKey
- **Note**: Keep this .env file in your .gitignore to avoid exposing it in public repositories.

MongoDB URI:

If you are using **local MongoDB**, the default URI is:

mongodb://localhost:27017/househunt

If using **MongoDB Atlas** (cloud database), replace MONGO_URI with:

mongodb+srv://<username>:<password>@cluster.mongodb.net/househunt?retryWrites=true&w=majority

Optional. env.example Template for GitHub:

Create a. env.example to help collaborators understand what variables they need to add.

env

PORT=5000

MONGO_URI=your_mongodb_connection_uri

JWT_SECRET=your_jwt_secret

By the end of this section, the development environment for the House Hunt – House Rent App is fully prepared for local execution. All necessary software tools, packages, and dependencies have been installed for both the frontend and backend, and environment variables have been configured securely using a .env file.

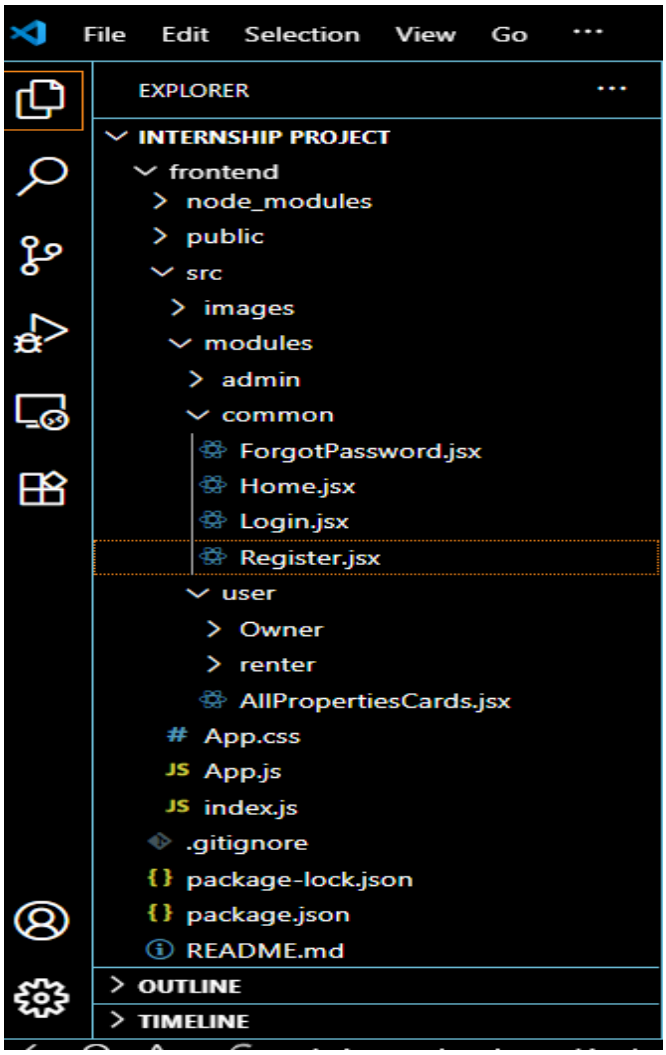
The project structure has been defined, and the MongoDB database is either set up locally or connected via MongoDB Atlas. This foundation ensures that developers can now run, test, and build upon the application with ease**.**

5. Folder Structure

The House Hunt project is organized into two primary directories — **frontend** and **backend**. Each directory follows a modular structure to ensure clear separation of concerns, ease of development, and maintainability.

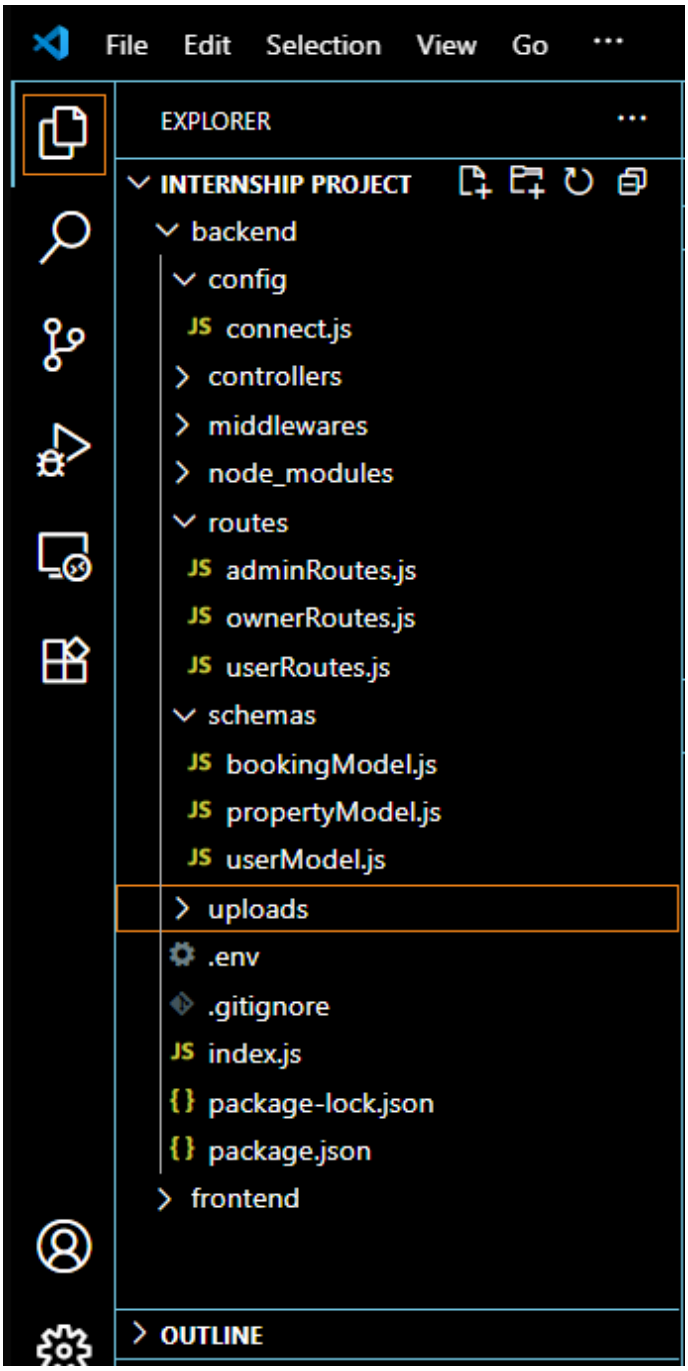
5.1 Client-Side (Frontend)

The frontend is built using **React.js**. It manages the user interface and handles communication with the backend through REST APIs using **Axios**.



5.2 Server-Side (Backend)

The backend is built using **Node.js** with the **Express.js** framework. It provides APIs for authentication, property listings, and bookings, and connects to the **MongoDB** database.



The folder structure ensures:

- A clear **separation of frontend and backend**
- Organized handling of **API logic, middleware, models**, and **UI components**
- Easy onboarding for new developers and easier debugging

This modular project layout enhances **scalability**, **readability**, and **collaboration**.

6. Running the Application

After completing the setup and configuration steps, the application is ready to be executed in a local development environment. This section explains how to run both the **frontend (React.js)** and **backend (Node.js + Express.js)** servers and validate that the system is working properly.

6.1 Frontend Server

The frontend is developed using **React.js** and handles all user interactions, form submissions, and interface rendering. It communicates with the backend using **Axios** to fetch or send data.

Steps to Run the Frontend:

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

cd frontend

3. Install any remaining packages (if not already done):

npm install

4. Start the React development server:

npm start

Output:

- This command will automatically launch the app in your default browser.
- The frontend runs at:
  http://localhost:3000

Frontend Features:

- User login/registration interface
- Property listing and booking interface
- Responsive UI (Bootstrap + Ant Design)
- API integration using Axios

6.2 Backend Server

The backend server is built using **Node.js** with the **Express.js** framework. It is responsible for handling API routes, database communication, user authentication, and business logic.

Steps to Run the Backend:

1. Open a **separate terminal window**.
2. Navigate to the backend directory:

```
cd backend
```

3. Install required backend dependencies (if not already done):

```
npm install
```

4. Start the server using:

```
nodemon index.js
```

If nodemon is not installed globally:

```
npm install -g nodemon
```

Output:

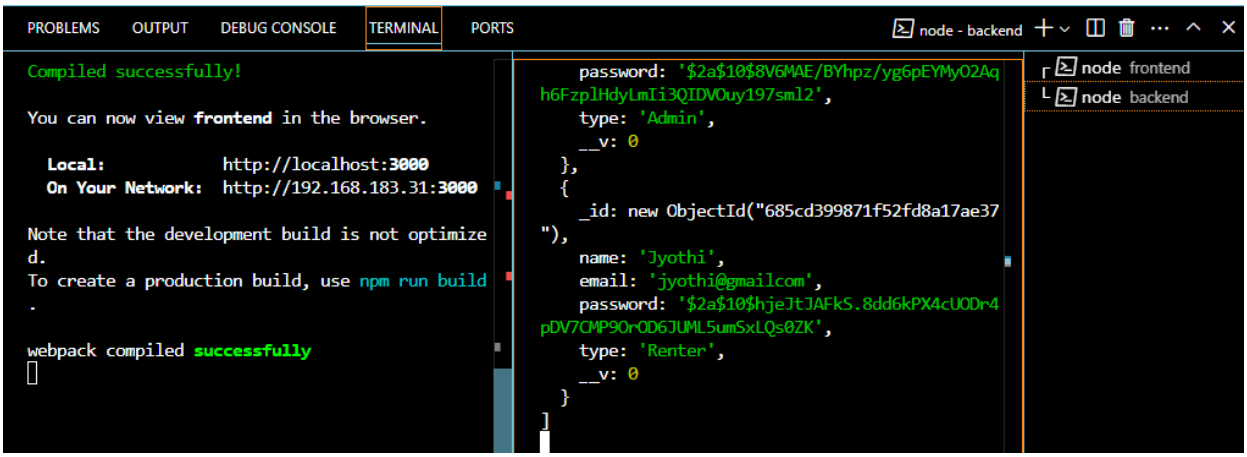
- The backend server runs at:
  http://localhost:5000
- It connects to the MongoDB database and exposes secure APIs.

Backend Features:

- RESTful API for authentication, bookings, properties, admin controls
- Token-based security using JWT
- Image/file uploads using Multer
- Role-based access (renter, owner, admin)

Confirmation of Successful Execution

Component	Expected URL	Output
Frontend	http://localhost:3000/	Loads the House Hunt UI
Backend	http://localhost:5000/	Responds to API requests (Postman/browser)



7. API Documentation

The backend of the House Hunt application exposes a set of **RESTful API endpoints** that handle authentication, property management, booking requests, and admin operations. These APIs are accessed by the frontend using **Axios** and are secured using **JWT-based authentication**.

7.1 Endpoint Descriptions

Here's a table listing the core endpoints used in the project:

Endpoint	Method	Access	Purpose
/api/users/register	POST	Public	Register as a renter or owner
/api/users/login	POST	Public	Authenticate user and return JWT token
/api/properties	GET	Public	Get all property listings
/api/properties	POST	Owner Only	Add a new property listing
/api/properties/:id	PUT	Owner Only	Update property details
/api/properties/:id	DELETE	Owner Only	Delete a property listing
/api/bookings	POST	Renter Only	Submit a booking request
/api/bookings/user/:id	GET	Renter Only	View all bookings by a renter
/api/bookings/owner/:id	GET	Owner Only	View bookings received on owner's properties
/api/bookings/:id	PUT	Owner/Admin	Approve or reject a booking
/api/admin/users	GET	Admin Only	View all registered users
/api/admin/approve-owner/:id	PUT	Admin Only	Approve user as a verified property owner

All protected routes require a valid **JWT token** in the request header:

Authorization: Bearer <token>

7.2 Request and Response Examples

Let's walk through a few common API requests and expected responses:

Registering a New User

Endpoint: POST /api/users/register
Request Body:

json

```
{
  "name": "Janani",
  "email": "janani@example.com",
  "password": "password123",
  "role": "renter"
}
```

Response:

json

```
{
  "message": "User registered successfully",
  "user": {
    "_id": "60b123...",
    "name": "Janani",
    "email": "janani@example.com",
    "role": "renter"
  }
}
```

Login

Endpoint: POST /api/users/login
Request Body:

json

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

Response:

json

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "_id": "60b123...",
    "name": "Janani",
    "role": "renter"
  }
}
```

Add Property (Owner Only)

Endpoint: POST /api/properties
Headers:

makefile

```
Authorization: Bearer <token>
```

Request Body:

json

```
{
  "title": "2BHK Apartment in Vijayawada",
  "price": 10000,
  "location": "Labbipet, Vijayawada",
  "description": "Spacious flat near schools and shops",
  "amenities": ["Wi-Fi", "Parking", "Water Supply"]
}
```

Response:

json

```
{
  "message": "Property listed successfully",
  "property": {
    "_id": "60c456...",
    "title": "2BHK Apartment in Vijayawada",
    "status": "pending"
  }
}
```

Send Booking Request (Renter Only)

Endpoint: POST /api/bookings

Headers:

makefile

Authorization: Bearer <token>

Request Body:

json

```
{
  "propertyId": "60c456...",
  "renterId": "60b123...",
  "message": "Interested in visiting the property this weekend"
}
```

Response:

json

```
{
  "message": "Booking request submitted",
  "bookingStatus": "pending"
}
```

Approve Booking (Owner/Admin Only)

Endpoint: PUT /api/bookings/:id

Request Body:

json

```
{
  "status": "approved"
}
```

Response:

json

{

"message": "Booking has been approved"

}

8. Authentication

Authentication is a key part of the House Hunt application. It ensures that only **verified users** can access protected features such as booking a property, listing a property, or managing users (in the case of admins).

The app implements **role-based authentication** using **JSON Web Tokens (JWT)**, enabling secure and stateless login sessions.

8.1 User Authentication Workflow

The House Hunt app supports three types of users: **Renter**, **Owner**, and **Admin**. The authentication process is handled as follows:

Step-by-Step Login Flow:

1. **User Registration**
    - A new user (renter or owner) signs up via /api/users/register.
    - Details are stored in MongoDB with role information.
2. **User Login**
    - The user enters email and password via the login form.
    - The credentials are sent to /api/users/login.
3. **Validation & Token Generation**
    - Backend validates credentials.
    - If valid, a **JWT token** is generated and sent back.

4. **Frontend Stores Token**
    - Token is stored in **localStorage** (or cookies) on the client-side.
    - It is then used for all subsequent authenticated requests.
5. **Protected Route Access**
    - When accessing secure APIs (like /api/bookings or /api/properties), the frontend attaches the token to the Authorization header.
6. **Backend Token Verification**
    - Backend middleware verifies the token using a secret key from .env.
    - If valid, the request proceeds.
    - If invalid or expired, the request is denied.

8.2 Token Handling (JWT)

The application uses **JWT (JSON Web Tokens)** to manage authenticated sessions.

How JWT Works:

- After login, the server creates a token with encoded user data (ID, role, expiry).
- This token is signed using a **JWT_SECRET** (defined in .env).
- The token is returned to the client and stored for future use.

Structure of a JWT:

A JWT typically has 3 parts:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

- **Header**: Algorithm & Token type
- **Payload**: Encoded user data (like user ID and role)
- **Signature**: Secret key signature for verification

Token Security Measures:

Feature	Description
Expiry Time	Tokens have a limited lifespan (e.g., 1 hour)
Middleware Protection	Backend checks token on every request to secure routes
Role Authorization	Certain actions (like admin approvals) require specific roles
No Sensitive Info	Passwords and personal data are never stored in tokens

Sample Authorization Header:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...

Refresh Strategy (Optional for future scope):

- Implement token **refresh logic** if you want longer sessions.
- Store a **refresh token** securely in HTTP-only cookies.

Conclusion of Section 8

The House Hunt application uses a secure, modern, and scalable authentication system powered by JWT. With role-based access and encrypted sessions, it ensures that only authorized users can access sensitive features. This protects user data, enables multi-role functionality, and sets the foundation for future enhancements like refresh tokens or two-factor authentication.

9. User Interface

The **User Interface (UI)** of the House Hunt application is designed to be **intuitive, responsive, and user-role specific**. It ensures smooth and clear navigation experience for three types of users — **Renters, Property Owners, and Admins**. The frontend is developed using **React.js**, with styling and layout powered by **Bootstrap**, **Ant Design**, and **Material UI**.

The application follows component**-based architecture** to ensure maintainability, reusability, and scalability.

9.1 UI Components Overview

The UI is carefully divided into reusable sections, providing a clean structure for code management and user experience.

Core Components and Pages:

Component/Page	Description
Navbar	Persistent header with navigation options (Home, Login, Register, Dashboard)
Home Page	Landing page introducing the platform
Login/Register Forms	Secure login and sign-up pages with form validation and error handling
User Dashboard	Different for Renters and Owners; shows properties or booking activity
Admin Panel	Admin can view and approve users and listings
Property Listings	Cards showing property details with images, location, and pricing
Property Filters	Enables renters to search by location, price, amenities, etc.
Property Upload Form	Used by owners to upload new properties with photo support (via Multer)
Booking Form	Allows renters to submit a booking request for a selected property
Booking Status View	Renters can view current status: Pending, Approved, or Rejected
Footer	Contains site links, policies, and contact info

10. Testing

Testing is a crucial phase of the development lifecycle to ensure that the application works as expected, is free of major bugs, and performs reliably under different scenarios. The House Hunt application underwent both **manual functional testing** and **API testing** using modern tools and techniques.

This section covers the testing approach, tools used, and a summary of outcomes to validate the correctness and performance of the system.

10.1 Testing Tools and Strategy

To ensure the reliability, usability, and correctness of the House Hunt application, a structured testing methodology was followed. This involved testing both the **client-side (frontend)** and **server-side (backend)** components using a mix of **manual testing techniques** and **automated testing tools**. Each module was tested in isolation, followed by **end-to-end role-based testing**.

The testing phase aimed to identify bugs, validate business logic, verify API interactions, check responsiveness, and confirm role-based restrictions.

Tools Used

Tool	Purpose
Postman	Used to test RESTful API endpoints with different HTTP methods (GET, POST, etc.)
Chrome DevTools	Used to inspect UI rendering, console errors, responsive layouts, and network activity
Lighthouse	Automated tool integrated with Chrome DevTools to evaluate performance, accessibility, and SEO
Visual Studio Code	Debugging backend logic, monitoring server responses, and checking logs
MongoDB Compass	GUI tool to visualize and verify database operations (data insertion, updates, and relationships)

Testing Strategy

The testing process for House Hunt followed a **module-wise and role-specific approach**, broken down as follows:

1. Functional Testing (Frontend + Backend)

- Verified that all **user-facing features** (login, registration, booking, property upload) worked as expected.
- Covered various user roles: **Renter**, **Owner**, and **Admin**.
- Checked whether **navigation** between pages functioned properly.
- Tested **form validation** and **user input handling**.
- Confirmed **correct response codes** and UI feedback.

2. API Testing (Backend)

- Used **Postman** to test all API endpoints.
- Checked for:
    - Correct HTTP status codes (200, 201, 400, 401, 403)
    - Proper request/response structures
    - Authentication and token handling (JWT)
    - Role-based access (Owner vs Admin)
- Ensured **data consistency** between frontend actions and MongoDB collections.

3. UI/UX Testing (Frontend)

- Performed **manual UI walkthroughs** to evaluate:
    - Button behavior
    - Form inputs and alerts
    - Field validations (empty input, incorrect format)

    - Real-time feedback (e.g., success messages, error prompts)
- Used **responsive mode** in Chrome to simulate various screen sizes:
    - Desktop
    - Tablet
    - Mobile

4. Performance Testing

- Ran **Lighthouse audits** to evaluate:
    - **First Contentful Paint (FCP)**
    - **Load Time**
    - **Accessibility**
    - **Best Practices**
- Ensured lightweight frontend components to avoid heavy load times.

5. Security & Access Testing

- Verified that **protected routes** (like adding a property or booking approval) required a valid **JWT token**.
- Attempted unauthorized access using invalid/expired tokens to confirm route protection.
- Ensured **admin-only** pages are inaccessible to normal users.

6. Database Testing (MongoDB)

- Inspected MongoDB via Compass:
    - Confirmed correct insertion of user, property, and booking documents
    - Checked relational mapping using ObjectId references
    - Validated update and delete operations through backend routes

Test Coverage Summary

Module Tested	Scope	Status
Authentication	Register/Login, JWT Token Storage	Passed
Dashboard Functionality	Owner, Renter, and Admin Role Views	Passed
Property Management	CRUD Operations, Image Upload	Passed
Booking System	Request/Approval Flow	Passed
UI Layout	Cross-device Rendering and Usability	Passed
Backend API	Secured Endpoints, Status Codes, Tokens	Passed

10.2 Manual and API Testing

To ensure all features of the House Hunt application work seamlessly and securely, both **manual testing** (via the user interface) and **API testing** (via Postman) were conducted. These testing methods helped validate the system's behavior under real-world usage and identified any hidden issues.

Manual Testing

Manual testing was performed by interacting directly with the frontend application using a web browser. The goal was to simulate real user actions across all roles — **Renter**, **Owner**, and **Admin** — and ensure the interface responded correctly.

Key Functionalities Tested Manually:

Test Scenario	Expected Behavior	Result
Register as a new user (renter/owner)	Account should be created and redirected to login page	✅ Passed
Login with valid and invalid details	Successful login returns token; invalid details show error message	✅ Passed

Add new property (owner)	Form should validate inputs and upload data, then display success alert	✅ Passed
View property listings (renter)	All approved properties should be displayed with filters (location, price, etc.)	✅ Passed
Submit a booking request	After filling the form, the request should be stored and shown as “Pending” in dashboard	✅ Passed
View booking history (renter)	Renter should see all submitted bookings with current status	✅ Passed
Admin approval of listings	Admin should approve new owners and properties; status should update	✅ Passed
Responsive layout	UI should render well on mobile, tablet, and desktop views	✅ Passed

Browsers Used:

- Google Chrome (primary)
- Firefox (for cross-browser consistency)

API Testing Using Postman

To validate backend functionality independently from the frontend, API endpoints were tested using **Postman**. This allowed precise control over requests, payloads, and headers to confirm the backend logic, data validation, and authentication system.

Token-Based Authentication Testing:

- Sent requests to protected routes with valid and invalid JWT tokens
- Verified that **unauthorized users** received 401 Unauthorized responses
- Ensured only **admins and owners** could access their respective endpoints

Sample API Tests:

API Endpoint	Method	Purpose	Status
/api/users/register	POST	Create a new user	✅ Passed
/api/users/login	POST	Log in and receive JWT	✅ Passed
/api/properties	POST	Add a new property (owner only)	✅ Passed
/api/properties	GET	View all approved properties	✅ Passed
/api/bookings	POST	Submit a booking request (renter only)	✅ Passed
/api/bookings/user/:id	GET	View bookings made by a renter	✅ Passed
/api/bookings/owner/:id	GET	View bookings received by an owner	✅ Passed
/api/bookings/:id (status update)	PUT	Approve or reject a booking (owner/admin)	✅ Passed
/api/admin/approve-owner/:id	PUT	Admin approval of owner registration	✅ Passed

Headers Example:

pgsql

Authorization: Bearer <JWT_TOKEN>

Content-Type: application/json

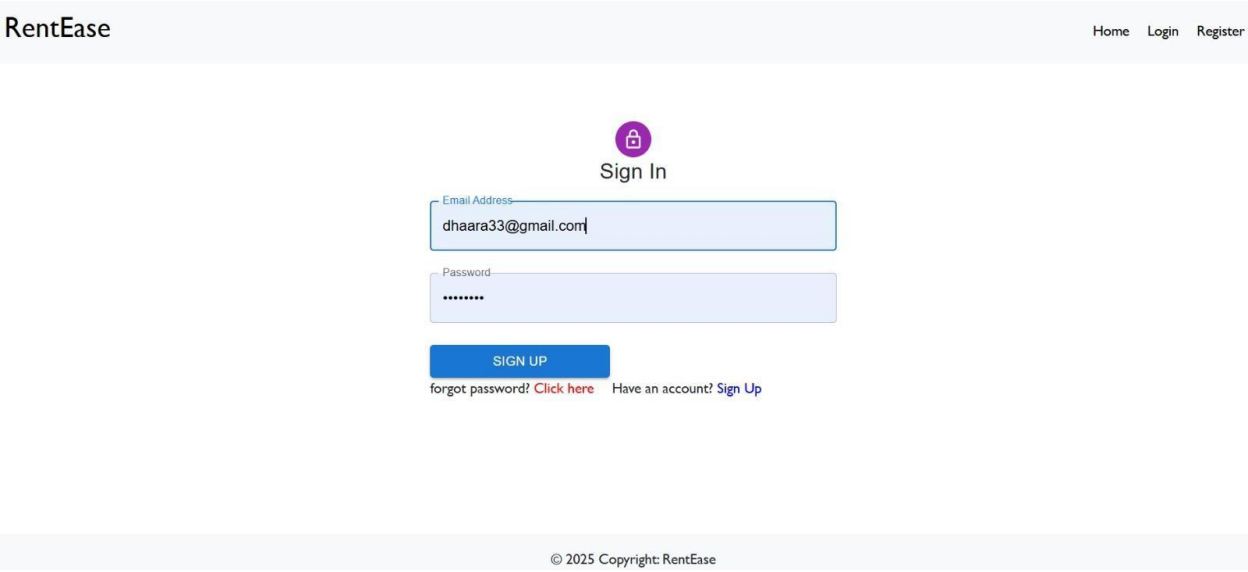
11. Demo and Output

The final output of the House Hunt application showcases a smooth, user-friendly rental property platform with key features implemented successfully. This section includes real-time **screenshots** and a **demo video** to illustrate functionality across user roles.

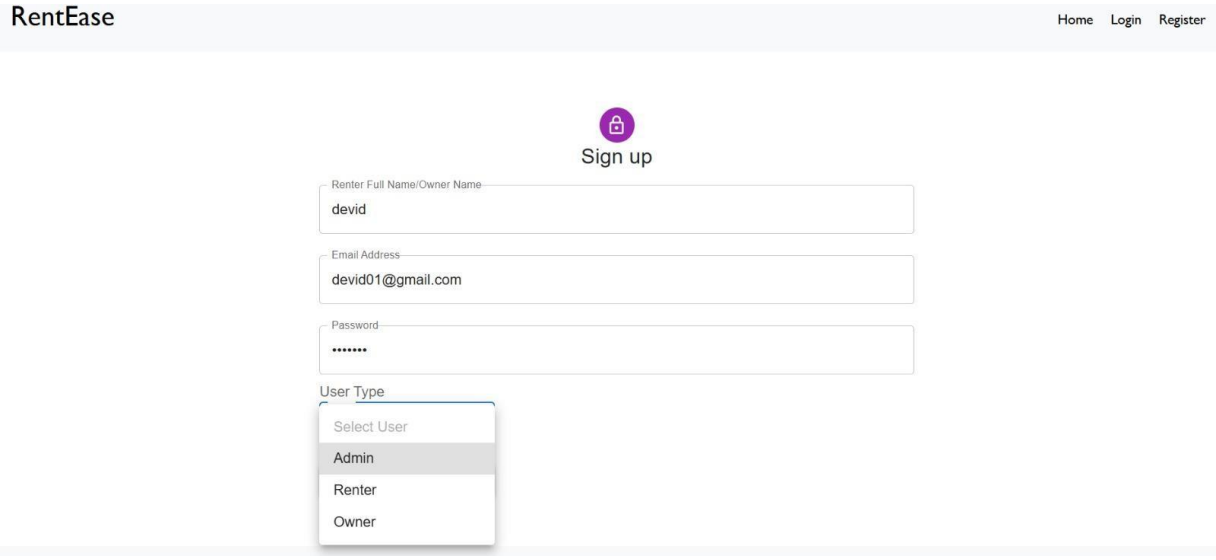
11.1 Screenshots

Include the following screenshots in your documentation (with figure numbers and brief captions):

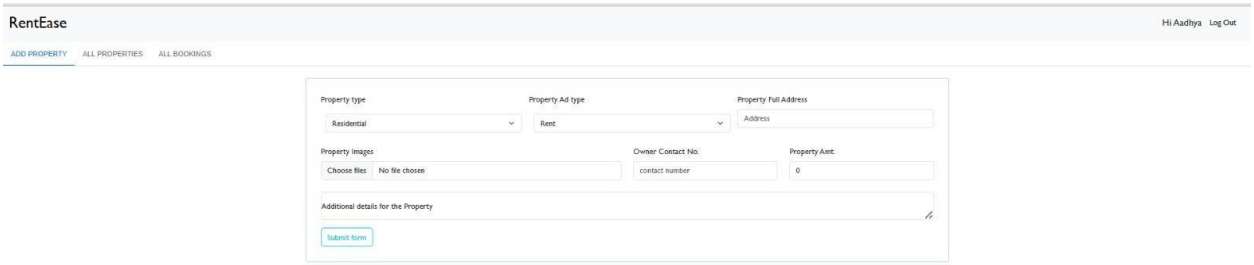
Login Page:



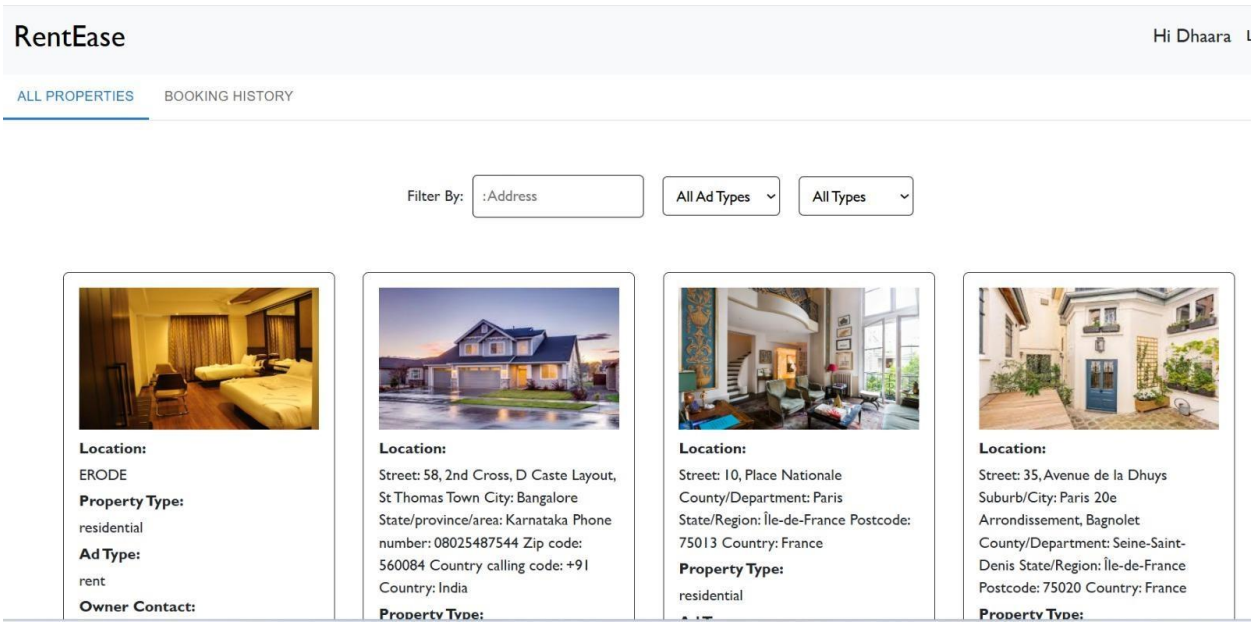
Register Page



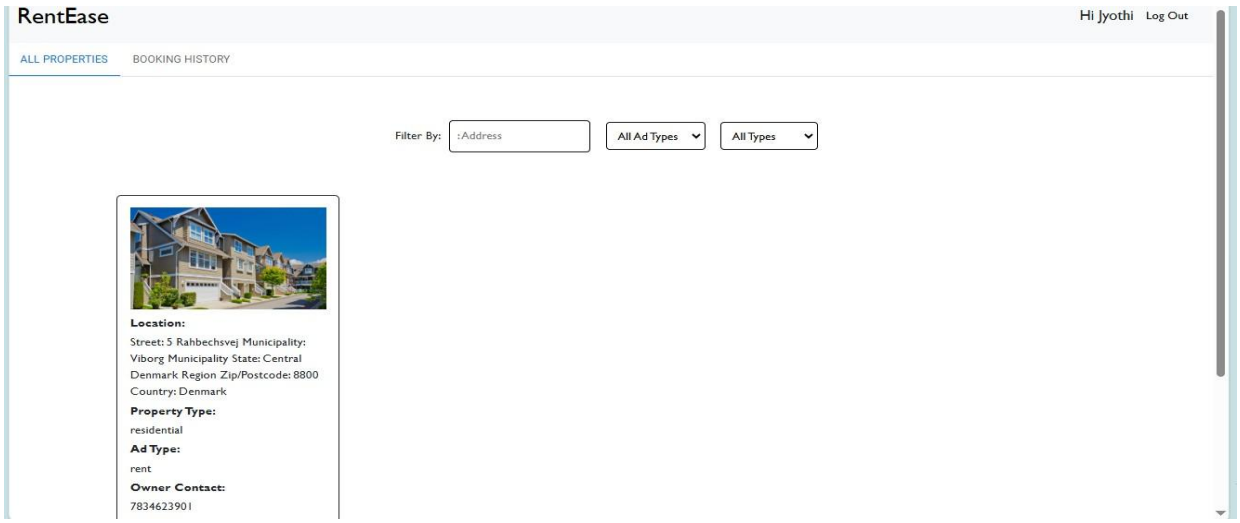
Owner Dashboard



Property Upload Page

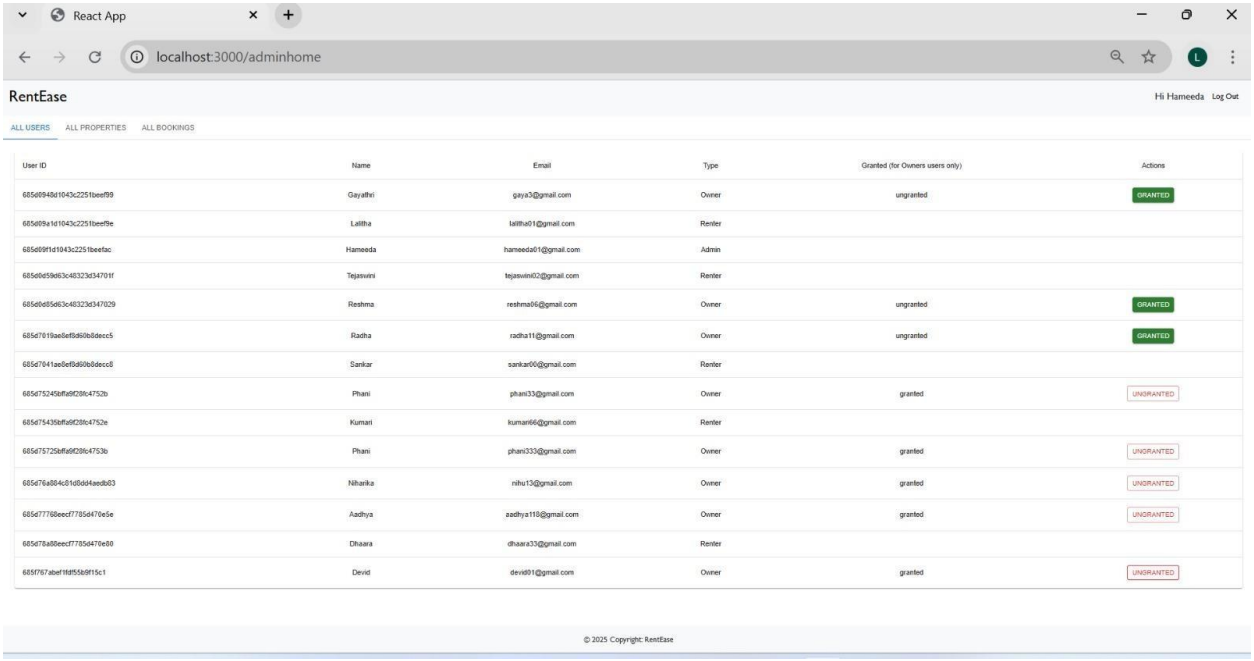


Renter View – Property Listings



Booking History

Admin Panel



11.2 Demo Video Link

https://github.com/devavaka/HouseHunt/tree/main/Demo%20video

12. Known Issues

Despite successful implementation and testing, a few limitations and minor bugs were observed during the development and demonstration of the **House Hunt – Finding Your Perfect Rental Home** application. These issues are not critical but may affect usability or scalability in the long term. Identifying them helps plan future improvements.

12.1 Technical Limitations

These are features that were either simplified due to time constraints or are planned for future updates:

Limitation	Description
+ **No Notification System**	Currently, the application does not support email/SMS notifications for bookings or approvals.
+ **No Payment Integration**	There is no option for renters to pay security deposits or rent through online gateways.
+ **Limited Search Filters**	Filtering properties is limited to basic fields like location and price. No map-based or advanced filter features.
+ **Only One Image per Property**	Each property supports a single image upload. Multi-image gallery support is not implemented.
+ **Lack of In-App Messaging**	Renters and owners cannot chat or communicate directly within the app.
+ **Static Admin Dashboard**	Changes made by admin (like approvals) sometimes need a manual refresh to appear immediately.

These limitations were acknowledged during development planning and can be resolved in the next version or future sprint.

12.2 Bugs Identified

These are unintended behaviors observed during manual or API testing. While they don't break the app, they may affect the user experience or require minor fixes.

Bug	Observed Behavior	Impact Level
No File Size Restriction	Large image files can be uploaded, which slows down the form or causes timeout.	Medium
Missing Field-Level Validation	Some forms lack proper error messages for missing or invalid input.	Low
Owner Role Delay After Approval	Owners may need to re-login to access their dashboard after admin approval.	Low
Form Submission Lacks Feedback	Booking or property forms do not show a loader/spinner, making users think it's stuck.	Medium
Navbar Overlap on Mobile View	On smaller screens, the navigation dropdown may cover page content.	UI/UX Bug

Developer's Note

These issues were discovered during testing on local environments using standard devices and browsers. None of these bugs prevent core functionality, but they may lead to reduced user satisfaction. A log of these bugs has been maintained for scheduled refactoring and polishing post-deployment.

The listed bugs and limitations are considered minor and do not impact the primary workflows of the House Hunt application. However, documenting them shows a professional approach to software development and helps plan future releases efficiently. As with any modern web application, addressing these areas will elevate the platform's overall quality, user experience, and maintainability.

13. Future Enhancements

13.1 Suggested Improvements

- **UI/UX Enhancements:** Improve responsiveness and accessibility for better cross-device compatibility and user experience.
- **Real-Time Features:** Integrate WebSocket or similar technologies for real-time updates and notifications.
- **Role-Based Access Control (RBAC):** Implement user roles (admin, editor, viewer) to manage permissions and access to specific functionalities.
- **Analytics Dashboard:** Add an admin dashboard to monitor usage, performance metrics, and user engagement statistics.
- **Offline Functionality:** Enable offline access to critical features using service workers and local storage.
- **Mobile App Version:** Develop a mobile counterpart using React Native or Flutter for broader accessibility.

13.2 Feature Roadmap

Feature	Description	Priority	Estimated Timeline
Role-Based Access	Admin/user roles and permissions	High	2 Weeks
Notification System	Push and in-app notifications	Medium	3 Weeks
Dark Mode	UI toggle for light/dark themes	Low	1 Week
Analytics Dashboard	Visual metrics for usage data	Medium	2-3 Weeks
Mobile App	Cross-platform mobile version	High	1-2 Months

14. APPENDIX

14.1 GitHub link for the video demo and documentation and source code:

https://github.com/Mullaarif12/HouseHunt

Conclusion:

The House Hunt project successfully demonstrates a functional and user-friendly platform for discovering and exploring residential properties. By integrating modern web technologies, intuitive UI, and efficient backend services, the system provides a smooth experience for users searching for homes based on various criteria like location, budget, and property type.

This project not only highlights core concepts in full-stack web development but also showcases practical implementation of features like property listing, dynamic search filters, and responsive design. While the current version serves as a strong foundation, there remains ample scope for future enhancements such as AI-based recommendations, real-time notifications, and integration with third-party APIs for live listings.

Overall, House Hunt proves to be a valuable solution for simplifying the property search process and can be expanded into a full-fledged application with real-world utility.