1. Introduction

Project Title: HouseHunt : Smart Search for Smarter Living

Team Members:

Team ID: LTVIP2025TMID55588

Team Leader: Badugu Sandesh

Team member: Attili Nithish (Full Stack Developer)

Team member: B Rupa Sri

Team member: Avanigadda Hemasrichandu

About the Project:

Househunt is a full-stack web application developed using the MERN stack (MongoDB, Express.js, React.js, Node.js). It functions as a digital platform where users can search for rental and sale properties, apply filters like price and location, and directly connect with property owners. The system is designed for both property seekers and property listers, offering a modern, scalable, and user-friendly experience. Through a dynamic and responsive UI built with React, users can register, login, view property listings, and even save their preferences. On the backend, Node.js and Express handle authentication, API routing, and secure data transactions, while MongoDB Atlas provides a scalable cloud-based database solution.

The project was developed collaboratively to simulate a real-world property listing system, reflecting the end-to-end software development lifecycle from frontend design to backend integration. It demonstrates strong understanding of component-based UI architecture, RESTful API development, secure authentication, and MongoDB document storage.

Objective:

The primary objective of this project is to develop a responsive and scalable web-based solution for real estate listing and discovery, offering:

- A simple and effective platform for users to explore available rental/sale properties.
- A secure authentication system to allow verified users to list or view properties.
- A filterable search experience with criteria like budget, location, and property type.
- A system where property listers (owners or admins) can add, edit, or remove listings.

2. Project Overview

Purpose:

The purpose of the Househunt project is to design and implement a fully functional, user-friendly, and responsive property listing and rental platform using the MERN stack (MongoDB, Express.js, React.js, Node.js). In today's fast-moving digital world, people increasingly prefer to find rental or sale properties online due to the convenience and speed it offers. HouseHunt aims to meet this need by offering a centralized web-based solution where users can browse property listings, apply filters based on location or budget, and contact property listers directly.

This platform enables users to search, filter, and view property details, including images, price, address, and availability. Users can register and log in securely to access enhanced features. Simultaneously, property owners/admins can log in to add new property listings, edit existing ones, and manage user requests from a dedicated dashboard.

This project simulates a real-world real estate listing website—similar to platforms like 99acres or MagicBricks—built from scratch with full control over frontend UI/UX, backend APIs, database modeling, secure authentication, and responsive design principles.

Goals:

The key goals of the HouseHunt project are:

- To develop a secure, scalable, and modern real estate web platform using the MERN stack.
- To implement role-based functionality for both general users (buyers/renters) and property listers (admins/owners).
- To demonstrate the ability of the development team to build and maintain a complete full-stack application.
- To provide a responsive and intuitive user interface that adapts seamlessly across various devices using modern CSS libraries like Tailwind CSS.

Features:

Below are the key features and modules implemented in the Househunt platform:

User Authentication & Authorization:

- Registration via email and password.
- Secure login functionality using JWT-based session tokens.
- Role-based access: General User vs. Admin/Lister.
- Secure password encryption using bcrypt.js.

Property Management Module:

- Admin can add, edit, or delete property listings.
- Upload property images through file input and store securely (using local storage or cloud).
- Display list of properties with search and filter options (e.g., location, price range).

Search and Filter System:

- Users can filter properties based on location, price, type (rent/sale).
- Search bar for location-based quick results.
- Real-time filtering and dynamic property cards display.

User Dashboard:

- View all saved property interests or recent views.
- Update profile information (name, email, password).
- View requested property statuses (if implemented).

Admin Panel:

- Access for property listers/admins to:
- Manage property listings.
- Edit property information.
- Remove outdated or inactive listings.
- Track user interactions or interest shown (future scope).

Responsive UI:

- Built with Tailwind CSS for quick and adaptive styling.
- Fully responsive design across mobile, tablet, and desktop.
- Clean, modern layout optimized for real-world usability.

3. Architecture

The Househunt application follows a modern 3-tier architecture using the MERN stack:

- MongoDB (Database)
- Express.js (Backend)
- React.js (Frontend)
- Node.js (Runtime Environment)

This modular structure promotes maintainability, scalability, and a clear separation of concerns across the presentation, logic, and data layers.

Frontend Architecture – React.js:

The frontend is built using React.js, enabling dynamic and responsive user interfaces through reusable, stateful components.

Structure:

- The codebase is organized into reusable React components.
- Pages such as Home, Properties, Login, Register, Dashboard, and Add Property are structured under a clear folder hierarchy within /client.
- Navigation is managed using React Router DOM, allowing seamless transitions and route protection based on user roles (User/Admin).
- A global Layout.jsx component wraps pages to maintain consistent headers, footers, and sidebars.
- Conditional rendering is used to customize views and controls depending on authentication status and role (e.g., hiding "Add Property" from regular users).
- State management utilizes useState, useEffect, and context for handling session data and property interactions.

Styling:

- Styled entirely with Tailwind CSS for consistency and responsiveness.
- Fully mobile-first and adaptive UI that renders cleanly across devices.

Backend Architecture – Node.js & Express.js:

The backend is built using Node.js and Express.js, forming the API layer that handles all client requests, authentication, and database operations.

Key Structure:

- Routes are modularized by feature, such as /routes/propertyRoutes.js, /routes/userRoutes.js, and /routes/authRoutes.js.
- Each route links to its controller file, e.g., createProperty, getProperties, updateUser, etc.
- Middleware (e.g., authMiddleware.js) ensures secure access control and verifies roles (User/Admin).
- Environment configuration (e.g., JWT_SECRET, MONGO_URI) is managed using the dotenv package.
- Multer is used to handle image uploads for property listings.

Security:

- Passwords are securely hashed with bcrypt.js.
- JWT tokens are used for authentication, stored in browser localStorage.
- Admin-only routes are protected using role-based authorization checks.

Database Architecture – MongoDB (Atlas):

The application uses MongoDB Atlas, a cloud-hosted NoSQL database that stores structured documents in separate collections for users, properties, and requests.

Collections:

- users: Stores user credentials, roles, and profile info.
- properties: Contains listing details like title, location, price, image URL, description, and status (rent/sale).
- requests: Captures user property interest or booking inquiries.
- (Optional future collections: favorites, messages, etc.)

Relationships:

- Each property listing references the userId of the creator (owner/admin).
- Booking requests or contact forms can reference both the propertyId and userId involved.

Indexing & Performance:

- Indexes on location, price, and status improve search and filter performance.
- Queries can be optimized further using aggregation pipelines for advanced search and dashboard analytics.

4. Setup Instructions

This section provides a comprehensive guide to setting up and running the Househunt project locally. It includes required tools, steps to clone the project, install dependencies, configure environment variables, and run both the client and server components.

Prerequisites

Ensure the following software and tools are installed on your system before setting up the project:

Tool	**Purpose**	**Version (Recommended)**
Node.js	JavaScript runtime environment	v18+
npm	Node package manager (comes with Node.js)	v9+
MongoDB Atlas	Cloud-hosted NoSQL database	Free Cluster
Git	Version control for cloning the repository	Any stable version
Code Editor	Recommended: VS Code	Optional
Cloudinary	(Optional) for storing book cover images	Free Tier
Postman	(Optional) for testing backend APIs	Optional
Multer	For image upload handling	Comes with npm install

Installation Guide:

Step 1: Clone the Project Repository

Bash (commands):

- git clone https://github.com/your-username/booknest.git
- cd househunt

The repository typically has two main folders:

- /client – the React frontend
- /server – the Node.js + Express backend

Step 2: Install Dependencies

Install client-side (frontend) packages:

Bash (commands):

- cd client
- npm install

Install server-side (backend) packages:

Bash (commands):

- cd ../server
- npm install

This will install all required packages from package.json including:

- **Backend:** express, mongoose, bcrypt, jsonwebtoken, cors, multer, dotenv
- **Frontend:** react-router-dom, axios, tailwindcss, @heroicons/react

Step 3: Set Up Environment Variables

Inside the /server folder, create a .env file:

Ini (file information):

- MONGO_URI=your_mongodb_atlas_connection_string
- JWT_SECRET=your_secret_key
- PORT=5000

Replace your_mongodb_atlas_connection_string with the URI from MongoDB Atlas.

Ensure the .env file is listed in .gitignore to keep secrets private.

Step 4: Initialize the Database (Optional)

As long as your MongoDB cluster is active, the backend will auto-connect and initialize required collections like users, properties, and requests upon first run.

Step 5: Run the Application

Start the backend server:

Bash (commands):

- cd server
- npm start

Start the frontend React app:

bash

- cd ../client
- npm start

Once started:

- **Frontend runs on:** http://localhost:3000
- **Backend runs on:** http://localhost:5000

You can now:

- Register/Login as a user or admin
- Browse and filter property listings
- Add new properties (admin/lister only)
- Edit/Delete existing listings
- Securely test APIs using Postman or Thunder Client

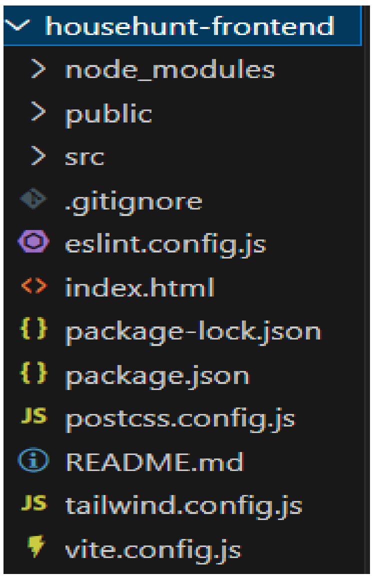
5. Folder Structure

The Househunt code-base is split into two top-level directories—/househunt-frontend and /househunt-backend—to keep the presentation layer fully decoupled from the server logic and database layer. This clear separation makes local development, containerisation, and cloud deployment far easier.

Client Folder Structure (React Frontend)

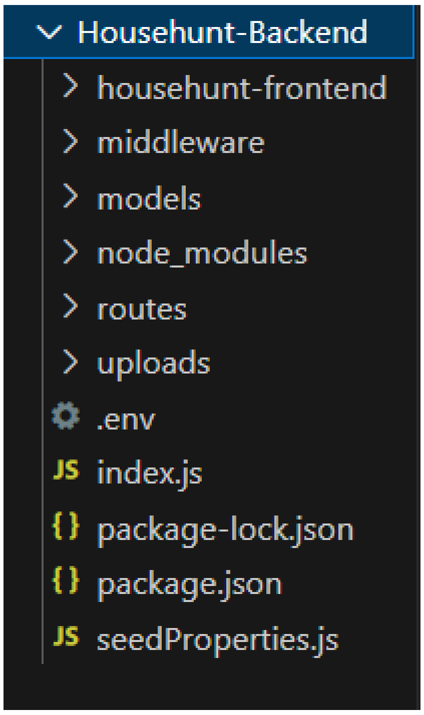
Folder / File	Purpose
househunt-frontend/	Root folder for the React app; contains the vite/webpack build scaffold, index.html, favicon, and other static assets.
public/	Optional place for images, icons, or custom static files you want copied verbatim to the build output.
components/	**Reusable UI blocks** such as PropertyCard, ProtectedRoute, FilterBar, Navbar, Footer, etc. Keeping them generic encourages re-use.
pages/	Top-level screens including Home.jsx, Properties.jsx, Login.jsx, Register.jsx, Dashboard.jsx, AddProperty.jsx, and any future feature pages.
App.jsx	Central router / layout orchestrator; wires up React Router DOM routes and global providers.
Layout.jsx	Wraps every page with shared UI (navbar, footer) and handles conditional rendering by auth role.
index.js (or main.jsx)	React-DOM entry-point that mounts <App /> to the root element.
tailwind.config.js	Tailwind CSS theme and purge configuration.
package.json	Declares React-side dependencies and contains dev, build, and lint scripts.

Tip: If you use Vite, you'll also see vite.config.js for faster local HMR.


```
househunt-frontend
  node_modules
  public
  src
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  postcss.config.js
  README.md
  tailwind.config.js
  vite.config.js
```

Server Folder Structure (Node.js + Express Backend)

Folder / File	Purpose
controllers/	Business-logic handlers, e.g. propertyController.js, userController.js, authController.js. Each function performs validation, interacts with models, and returns JSON.
middleware/	Cross-cutting concerns: authMiddleware.js (JWT verify), roleCheck.js (admin/lister guard), errorHandler.js, multerConfig.js for image uploads, etc.
models/	Mongoose schemas such as User.js, Property.js, and (future) Request.js, defining validation rules and static/helper methods.
routes/	Express routers grouped by feature: propertyRoutes.js, userRoutes.js, authRoutes.js. Keeps the URL map tidy.
uploads/	Local fallback directory for property images if you choose not to use a cloud bucket (S3, Cloudinary).
config/	Optional helpers for establishing DB or third-party connections (e.g., db.js, cloudinary.js).
.env	Environment variables—MONGO_URI, JWT_SECRET, PORT, etc. **Never commit this file.**
server.js	Application entry point: loads env vars, connects MongoDB, sets up Express, applies middleware, mounts routes, and starts the HTTP server.
package.json	Declares backend dependencies (express, mongoose, bcrypt, jsonwebtoken, cors, multer, dotenv) plus start / dev scripts.


```
Househunt-Backend
  > househunt-frontend
  > middleware
  > models
  > node_modules
  > routes
  > uploads
  .env
  index.js
  package-lock.json
  package.json
  seedProperties.js
```

6. Running The Application

Once all dependencies are installed and environment variables are configured, you’re ready to run the Househunt application locally. This section outlines the commands to launch both the React frontend and the Node.js backend, along with expected output and runtime behavior.

Starting the Frontend (React – househunt-frontend)

Navigate to the client folder:

Bash (commands):

- cd client

Start the development server:

Bash (commands):

- npm start

Output:

Upon successful start, you’ll see something like:

```
Compiled successfully!

You can now view BookNest in the browser.

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

- This launches the React development server at http://localhost:3000.
- The app supports hot reloading—any component changes are reflected in real time.
- React Router handles navigation across pages like /login, /register, /properties, /dashboard, /add-property, etc.

Starting the Backend (Node + Express Server):

Navigate to the server folder:

Bash (commands):

- cd ../server

Start the Express server:

Bash (commands):

- npm start

(Or use nodemon for development if installed: npx nodemon server.js)

Output:

If the .env file is correctly set up, you'll see:

```
Connected to MongoDB...

Server running on http://localhost:5000
```

The Express server runs on http://localhost:5000.

It exposes REST API endpoints such as:

- /api/users
- /api/properties
- /api/auth
- (optional: /api/requests, /api/admin)

Workflow:

- The React frontend interacts with backend APIs via **Axios** at **http://localhost:5000/api/...**
- Backend APIs securely connect to **MongoDB Atlas** to manage user accounts, property listings, and request data.
- **JWT-based middleware** is used to protect routes and restrict access based on roles (User or Admin).

Summary of Commands:

Task	Command	Directory
Install Client	npm install	/booknest-frontend
Run Frontend	npm start	/booknest-frontend
Install Server	npm install	/BookNest-Backend
Run Backend	npm start	/ BookNest-Backend

After both servers are running, you can open your browser and access:

- Frontend UI: http://localhost:3000
- Backend API: http://localhost:5000/api

Your full-stack real estate listing platform is now fully functional and ready for use, testing, or further development.

7. API Documentation

The Houseunht backend is built with Express.js and exposes a REST-style API for all core operations. Endpoints are grouped into six logical modules:

- Authentication – user sign-up / sign-in
- Properties – CRUD for rental / sale listings
- Requests – when a user expresses interest in a property
- Favorites – quick “save for later” list (optional)
- Profile – read / update logged-in user info
- Admin – platform-wide management endpoints

All routes return JSON and follow standard HTTP verbs (GET, POST, PUT, DELETE).

Authentication Routes:

Method	Endpoint	Description	Payload / Params	Sample Response
POST	/api/register	Register new user	{ "name", "email", "password" }	{ "token", "user": { … } }
POST	/api/login	User login	{ "email", "password" }	{ "token", "user": { … } }

Property Routes:

Method	Endpoint	Description	Payload / Params	Sample Response
GET	/api/properties	Fetch all properties (public)	Optional query: location, minPrice, maxPrice	[{ "_id", "title", "price", … }]
GET	/api/properties/:id	Fetch single property	—	{ "_id", "title", "description", … }
POST	/api/properties	**Add new listing** (Admin / Lister only)	multipart/form-data ⇒ images + fields	{ "message": "Property created" }
PUT	/api/properties/:id	**Update listing** (Admin / Lister only)	JSON with updated fields	{ "message": "Property updated" }
DELETE	/api/properties/:id	**Delete listing** (Admin / Lister only)	—	{ "message": "Property removed" }

Request (Enquiry) Routes:

Method	**Endpoint**	**Description**	**Payload / Params**	**Sample Response**
POST	/api/requests	Create a new enquiry for a property	{ "propertyId", "message", "contactNumber" }	{ "message": "Request sent" }
GET	/api/requests	Get all enquiries made **by the user**	JWT token in header	[{ "propertyId", "status", … }]
PUT	/api/requests/:id	**Lister/Admin** update request status	`{ "status": "Accepted"	"Rejected" }`

Favorites Routes:

Method	**Endpoint**	**Description**	**Payload / Params**	**Sample Response**
POST	/api/favorites/:id	Toggle property in favorites list	propertyId in URL	{ "message": "Added to favorites" }
GET	/api/favorites	Retrieve user’s favorite properties	JWT token	[{ "propertyId", "title", … }]

User Profile Routes:

Method	**Endpoint**	**Description**	**Payload/Params**	**Response Example**
GET	/api/users/profile	Get current user profile	JWT token	{ name, email, role }
PUT	/api/users/profile	Update user profile	{ name, email, password }	{ message: "Profile updated" }

Admin Routes:

Method	**Endpoint**	**Description**	**Access**
GET	/api/admin/users	View all users	Admin Only
GET	/api/admin/requests	View all property enquiries	Admin Only
DELETE	/api/admin/users/:id	Delete a user	Admin Only

Authentication using Thunder Client:

All protected routes require a JWT token to be sent via headers:

- Authorization: Bearer <token>

Sample Response Format:

```
{
  "message": "Request sent successfully",
  "request": {
    "_id": "666aa12fa3d9b1c8ed7d5420",
    "property": "665bb31c2d1a9f209c4e91e2",
    "user": "665bb15fca0cbb1f25793c5e",
    "status": "Pending",
    "contactNumber": "+919876543210"
  }
}
```

8. Authentication

Secure authentication and role-based authorization are integral to HouseHunt. A stateless, JWT-driven approach safeguards user sessions while ensuring that only privileged users (Admins/Listers) can perform sensitive actions such as creating, editing, or deleting property listings.

Authentication Flow

Step	Endpoint	What Happens
1. Register	POST /api/register	• User submits **name, email, password**.• Password is **hashed with bcrypt.js** and stored in MongoDB.• Server returns a signed **JWT** plus the user object.
2. Login	POST /api/login	• Server locates user by email.• Uses bcrypt.compare() to validate the password.• On success, issues a **JWT** signed with JWT_SECRET from .env.• Frontend stores the token (e.g. localStorage).
3. Authenticated Requests	Any protected route	• Frontend sends header Authorization: Bearer <token>.• Middleware verifies the token and attaches decoded data to req.user.

JWT Token Details

Field	Purpose
userId	MongoDB ID of the logged-in user
email	Email address
role	"user" or "admin" (or "lister")
iat / exp	Issued-at and expiry timestamps

- **Library:** jsonwebtoken
- **Storage (client):** localStorage (or secure cookies if preferred)
- **Typical expiry:** 24 h (configurable)

Protected Routes & Middleware

The backend uses middleware to secure routes:

/middleware/authMiddleware.js:

```
const jwt = require("jsonwebtoken");

const protect = (req, res, next) => {

  const token = req.headers.authorization?.split(" ")[1];

  if (!token) return res.status(401).json({ message: "Unauthorized" });

  try {

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    req.user = decoded;           // { userId, email, role }

    next();

  } catch {

    return res.status(403).json({ message: "Invalid Token" });

  }

};

module.exports = protect;
```

This middleware:

- Applied to endpoints like POST /api/properties, PUT /api/properties/:id, GET /api/users/profile, etc.Authorization by Role

Some routes are **restricted to admins only** using an additional check:

```
const adminOnly = (req, res, next) => {

  if (req.user.role !== "admin") {

    return res.status(403).json({ message: "Access denied" });

  }

  next();};
```

Added to admin-exclusive routes such as:

- GET /api/admin/users,
- DELETE /api/admin/users/:id,
- PUT /api/requests/:id (update enquiry status), etc.

Session Management

- Househunt is a stateless app (no server-side sessions)
- All user identity and session info is maintained via JWT tokens
- Tokens are validated on every request using the middleware, ensuring security

Aspect	**Implementation**
Auth Type	Token-based (JWT)
Storage (Frontend)	localStorage
Password Protection	bcrypt.js
Route Protection	Express Middleware
Admin Control	Role-based access with middleware
Expiry Handling	Configurable in JWT options

9. User Interface

The Househunt user interface is designed with a focus on clarity, responsiveness, and accessibility, ensuring an intuitive experience for both property seekers and listers. Built using React.js for component-driven interactivity and Tailwind CSS for utility-first styling, the platform ensures seamless navigation and consistent performance across devices.

Frontend Technology Stack

- **React.js:** Component-based architecture allows for reusable and manageable UI blocks (e.g., PropertyCard, Navbar, FilterBar, etc.).
- **Tailwind CSS:** Enables fast and responsive UI design through utility classes.
- **React Router DOM:** Provides smooth navigation between routes (e.g., /properties, /login, /dashboard, /add-property, etc.).

UI Features Overview

1. Landing Page (Home)

- A welcoming hero section with a brief description of the platform.
- Call-to-action buttons for "Explore Properties" or "List a Property."
- Clean navigation bar with links to login/register, property listings, and dashboard (post-authentication).

2. Property Listings Page

- Grid-based display of available houses/flats with:
- Title, image, location, price, and a brief description.
- Filter options (location, price range, type).
- Search functionality for quick access to relevant listings.
- Clickable cards that redirect to detailed property views.

3. Property Detail Page

In-depth view of a selected property:

- Image carousel
- Full description, amenities, location map, lister details
- Enquiry form for interested users to contact the lister.

4. Authentication Pages

- Login/Register pages with minimalist form design.
- Password protection, form validation, and error feedback included.

5. Dashboard

Customized based on user role:

- User Dashboard: View enquiries made, saved favorites, and profile management.
- Lister/Admin Dashboard: Manage property listings, view requests, and respond to users.

6. Add/Edit Property Form

- Image upload (with preview), field validation, and location inputs.
- Conditional rendering based on whether the form is being used to create or edit a property.

10. Testing

Thorough manual testing was conducted for both the frontend and backend components of the Househunt platform to ensure functionality, stability, and security across all user workflows. While automated test frameworks were not implemented, rigorous step-by-step verification was performed to validate each module and interaction.

Testing Strategy

The testing strategy for Househunt included multiple levels:

Layer	Focus
Unit Testing	Verified individual components such as forms, cards, buttons, modals.
Integration Testing	Checked API connectivity and data flow between frontend and backend.
End-to-End (E2E)	Simulated full user journeys such as register → login → request property.
Role-Based Testing	Verified access restrictions between users, admins, and listers.

Frontend Testing (Manual)

- **Form Validations:** Checked login, register, add property, and enquiry forms for empty field handling, invalid data, and error message display.
- **Component Rendering:** All UI components like Navbar, PropertyCard, Dashboard, Filters tested for proper rendering.
- **Protected Routes:** Verified redirection to login when accessing /dashboard, /add-property, /admin without authentication.
- **Responsive Design:** Used browser dev tools to ensure layouts adapt to mobile, tablet, and desktop screen sizes.

Tools used:

- Google Chrome DevTools
- React Developer Tools Extension
- Firefox and Edge browser testing

Backend API Testing (Manual using Postman)

All backend API endpoints were tested using Postman to verify:

- Correct response structure
- Status codes (200, 201, 400, 401, 403, 404)
- Protected route behaviour with and without JWT
- Admin-only route access

Sample Tests Conducted:

API Endpoint	Scenario	Expected Result
POST /api/register	Submit invalid email or weak password	400 Bad Request with error msg
POST /api/login	Correct email but wrong password	401 Unauthorized
GET /api/properties	Public access without token	Returns all listings
POST /api/properties	Non-admin tries to add listing	403 Forbidden
GET /api/requests	Fetch enquiries with valid JWT	Returns array of user requests
PUT /api/requests/:id	Admin changes status of request	Status updated successfully

Role-Based Testing:

Scenario	Expected Outcome
Regular user accesses admin panel	403 Forbidden error
Admin deletes a user	User removed successfully
Non-lister tries to add property	Access denied
JWT expired or tampered	Returns Invalid Token gracefully

Cross-Browser Testing

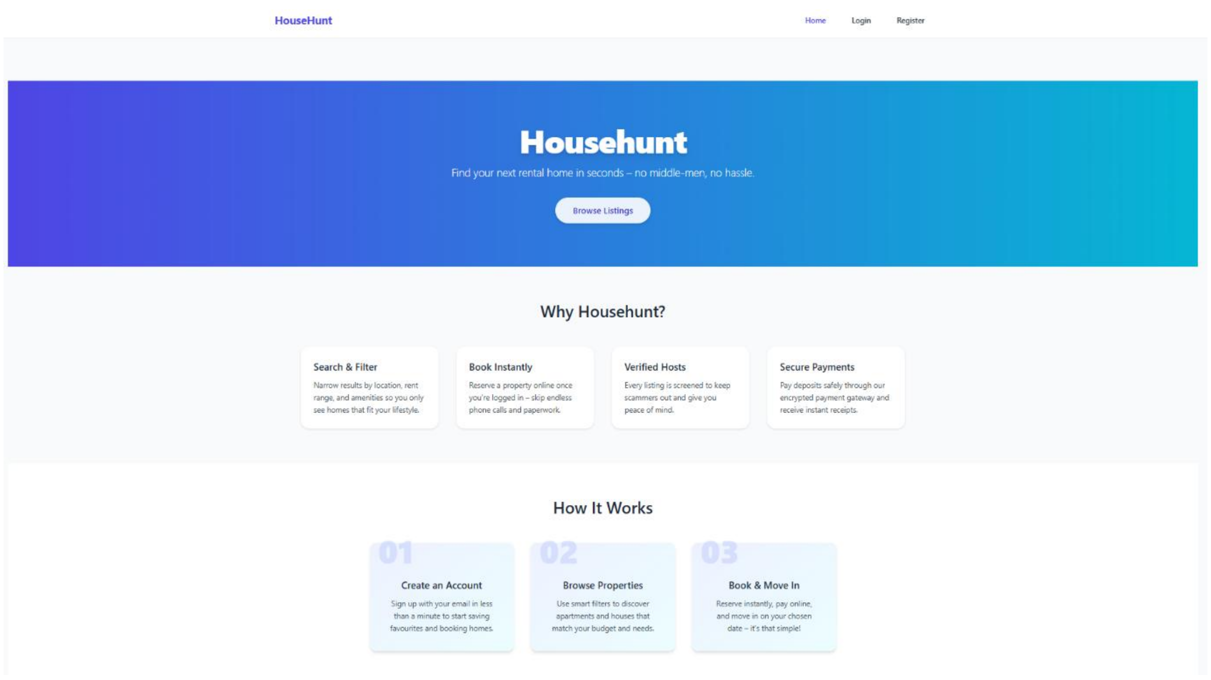
The app was tested on:

- Google Chrome
- Microsoft Edge
- Firefox

All displayed consistent behaviour and layout rendering.

11. Screenshots / Demo

Home Page:


HouseHunt

Home Login Register

Househunt

Find your next rental home in seconds – no middle-men, no hassle.

Browse Listings

Why Househunt?

Search & Filter
Narrow results by location, rent range, and amenities so you only see homes that fit your lifestyle.

Book Instantly
Reserve a property online once you're logged in – skip endless phone calls and paperwork.

Verified Hosts
Every listing is screened to keep scammers out and give you peace of mind.

Secure Payments
Pay deposits safely through our encrypted payment gateway and receive instant receipts.

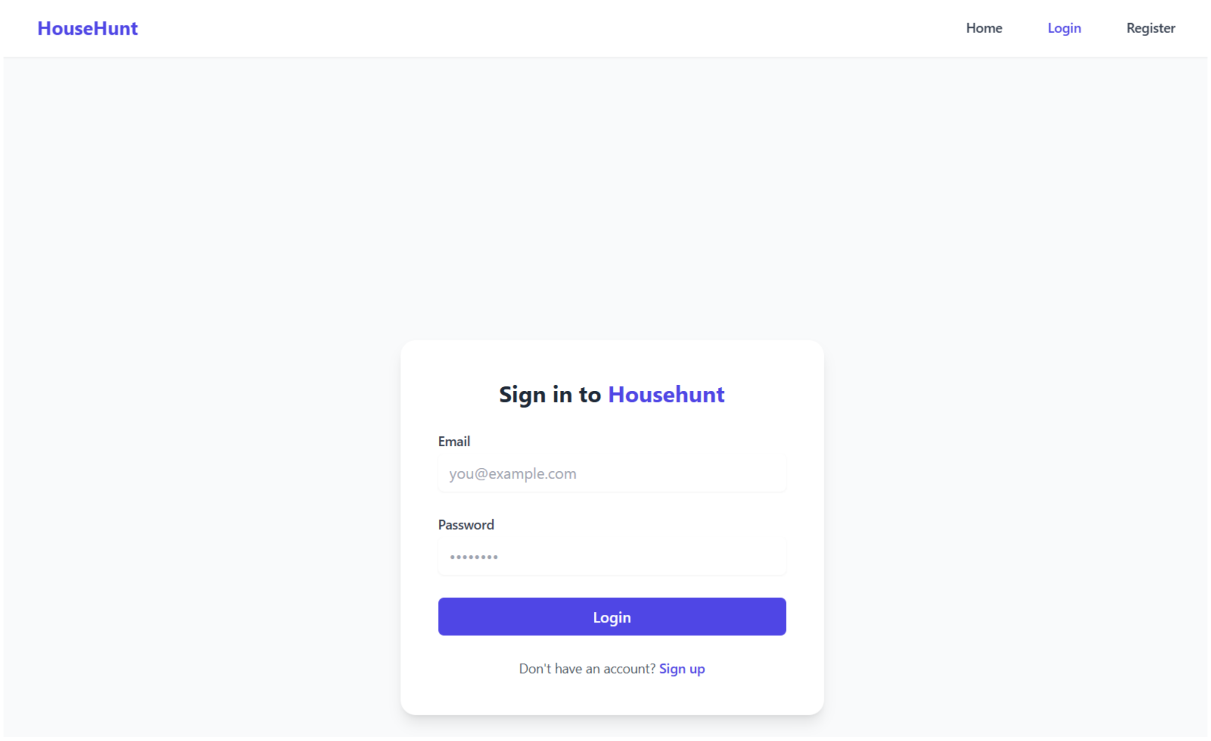
How It Works

01
Create an Account
Sign up with your email in less than a minute to start saving favourites and booking homes.

02
Browse Properties
Use smart filters to discover apartments and houses that match your budget and needs.

03
Book & Move In
Reserve instantly, pay online, and move in on your chosen date – it's that simple!

Login Page:


HouseHunt

Home Login Register

Sign in to Househunt

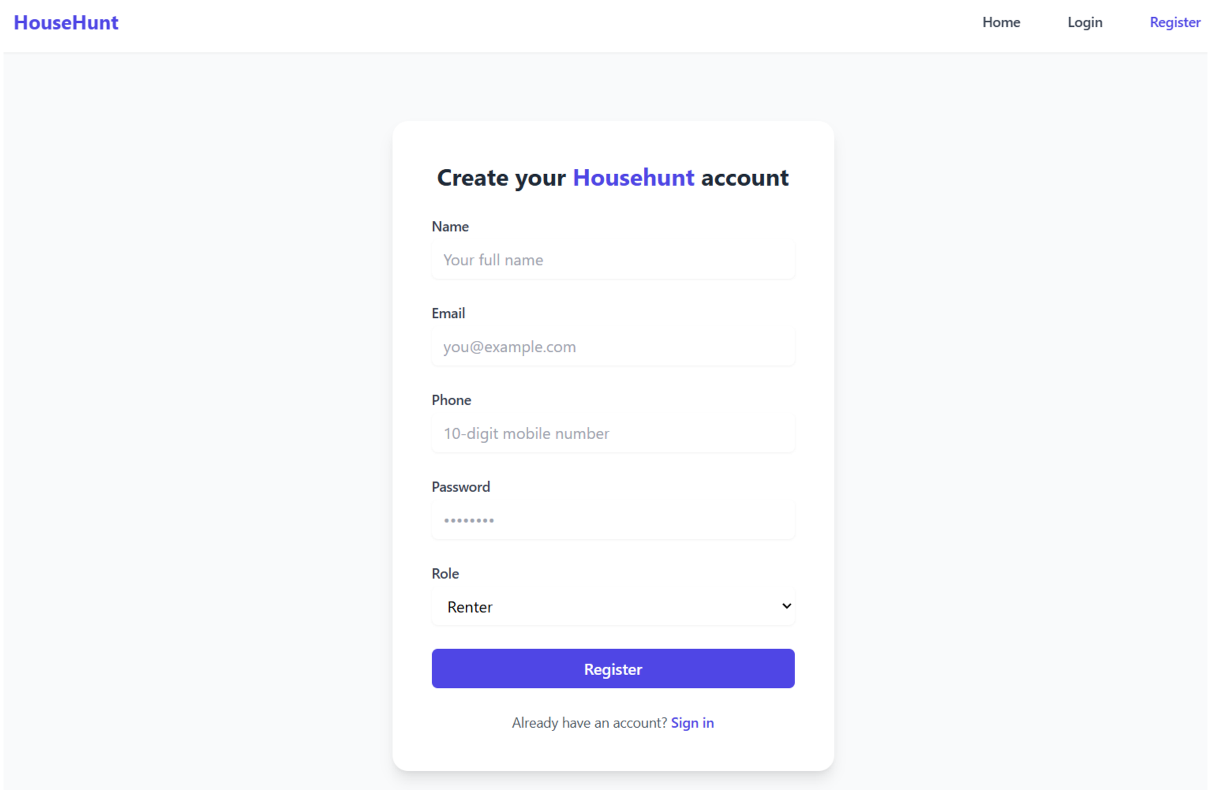
Email
you@example.com

Password
••••••••

Login

Don't have an account? Sign up

Registration Page:



HouseHunt

Create your Househunt account

Name

Your full name

Email

you@example.com

Phone

10-digit mobile number

Password

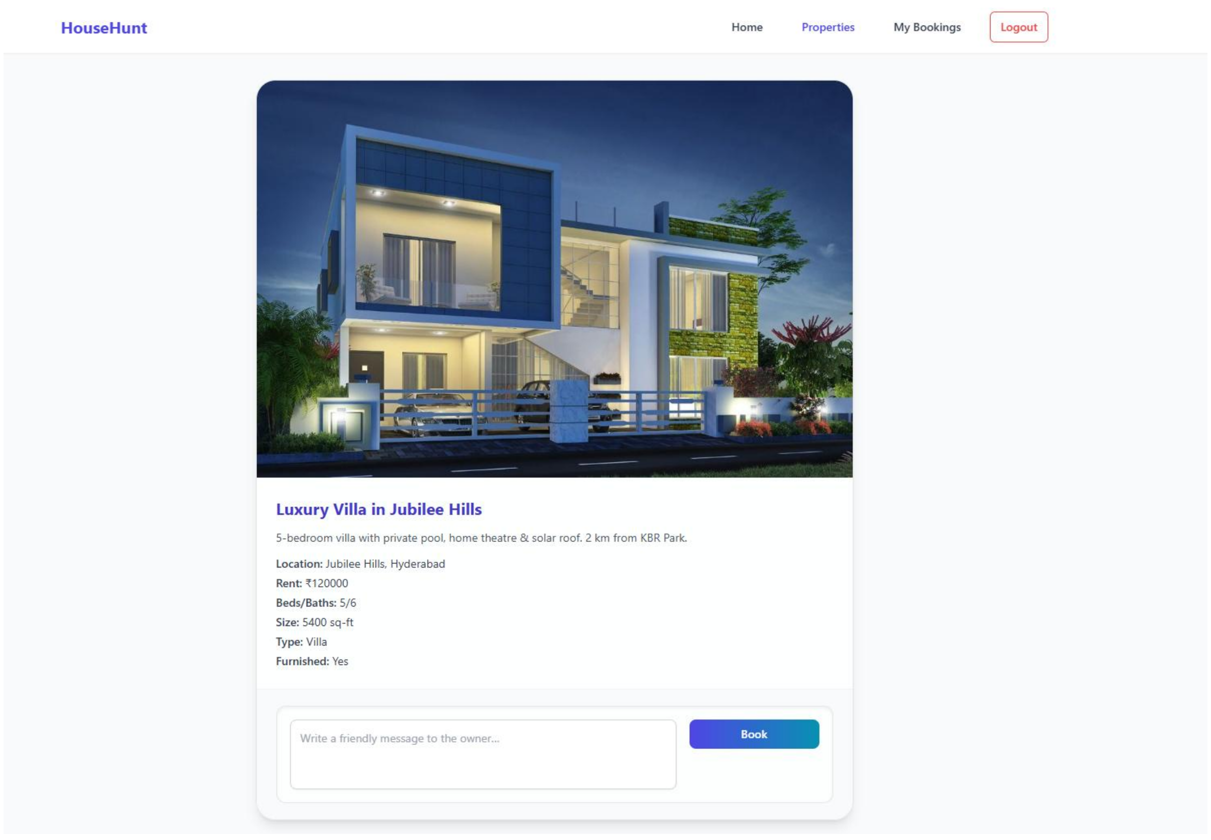
••••••••

Role

Renter

Already have an account? Sign in

Properties Page:



HouseHunt

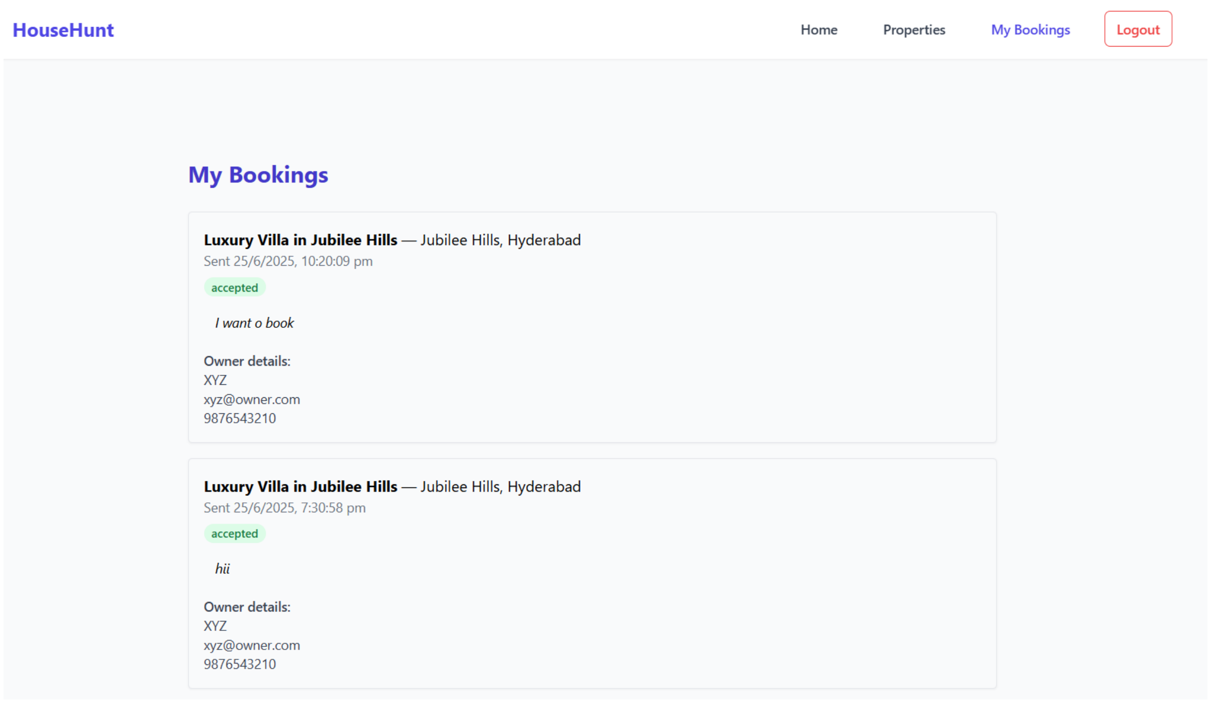
Luxury Villa in Jubilee Hills

5-bedroom villa with private pool, home theatre & solar roof. 2 km from KBR Park.

Location: Jubilee Hills, Hyderabad
Rent: ₹120000
Beds/Baths: 5/6
Size: 5400 sq-ft
Type: Villa
Furnished: Yes

Write a friendly message to the owner...

My Bookings Page of a User:



My Bookings

Luxury Villa in Jubilee Hills — Jubilee Hills, Hyderabad
Sent 25/6/2025, 10:20:09 pm
accepted

I want o book

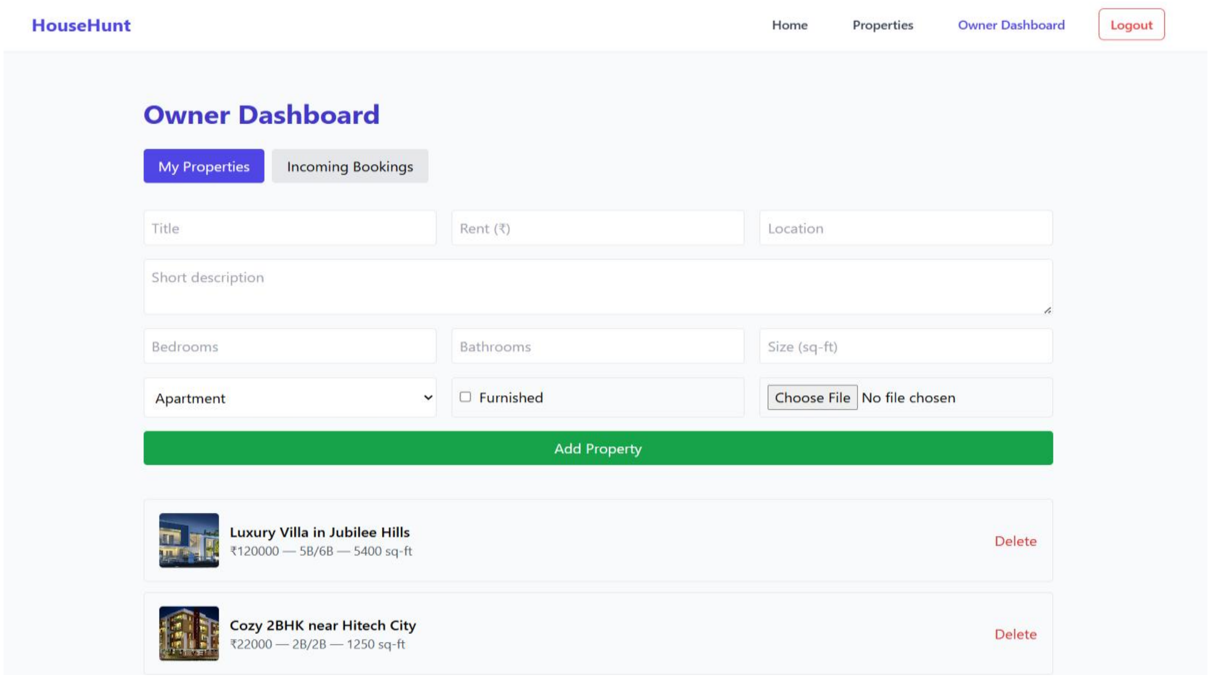
Owner details:
XYZ
xyz@owner.com
9876543210

Luxury Villa in Jubilee Hills — Jubilee Hills, Hyderabad
Sent 25/6/2025, 7:30:58 pm
accepted

hii

Owner details:
XYZ
xyz@owner.com
9876543210

Dashboard of an owner:



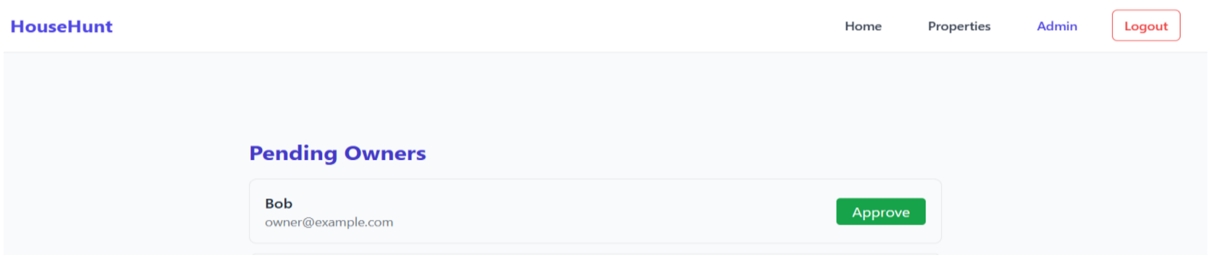
Owner Dashboard

No file chosen

Luxury Villa in Jubilee Hills
₹120000 — 5B/6B — 5400 sq-ft

Cozy 2BHK near Hitech City
₹22000 — 2B/2B — 1250 sq-ft

Admin Dashboard:



Pending Owners

Bob
owner@example.com

12. Known Issues

While the **Househunt** platform is fully functional and has been tested across essential use cases, a few known issues and limitations remain. These are either minor or are being considered for improvement in future iterations. The current issues do not significantly affect core operations but are documented here for transparency and future development planning.

1.No Map Integration for Property Locations

- **Issue:** Property listings only display location text (city/locality) without a visual map.
- **Impact:** Users cannot easily assess a property's exact location.
- **Planned Fix:** Integrate Google Maps or Leaflet.js for interactive map previews during listing and search.

2. No Email Verification During Registration

- **Issue:** Users can sign up without verifying their email address.
- **Impact:** Increases the risk of fake/spam accounts and reduces data integrity.
- **Planned Fix**: Implement email verification via Nodemailer and token-based account activation.

3. No Advanced Search Filters

- **Issue:** Property search is limited to basic criteria like city and type.
- **Impact**: Users cannot filter by price range, number of bedrooms, or amenities.
- **Planned Fix:** Add advanced filters such as budget range, BHK count, and furnishing status for better search control.

4. No Role-Based Admin Panel

- **Issue:** Admins cannot manage listings or users from the UI; roles must be edited manually in the database.
- **Impact:** Limits non-developer team members from moderating content or managing users.
- **Planned Fix:** Build an admin dashboard with role management and listing approval features.

5. No Pagination or Infinite Scroll

- **Issue:** All property listings load at once on the homepage.
- **Impact:** As listings grow, this will affect performance and increase load time.
- **Planned Fix:** Implement server-side pagination or infinite scrolling with lazy loading.

6. No Automated Testing Framework

- **Issue:** Testing is currently done manually.
- **Impact:** Risk of regressions during updates and no CI pipeline support.
- **Planned Fix:** Add Jest for backend unit testing, and React Testing Library or Cypress for frontend E2E tests.

13. Future Enhancements

To improve the functionality, scalability, and user experience of **Househunt**, the following enhancements are planned for future versions:

1. Advanced Search & Filter Options

- **Why:** Current search functionality is limited to location and property type.
- **Enhancement:** Add filters for price range, number of bedrooms (BHK), property age, amenities (e.g., parking, balcony), and furnishing type.
- **Impact:** Helps users narrow down listings and find properties that meet their exact needs.

2. Interactive Map Integration

- **Why:** Users cannot currently visualize the location of properties.
- **Enhancement:** Integrate Google Maps API or Leaflet.js to show property pins on a map view with interactive features.
- **Impact:** Improves transparency and decision-making by allowing users to assess neighbourhood and locality visually.

3. Admin Dashboard for Role and Listing Management

- **Why:** Admin tasks such as user management or property approval must be done manually in the database.
- **Enhancement:** Build a dedicated admin panel for approving/rejecting listings, managing users, and viewing site activity.
- **Impact:** Empowers non-technical team members to moderate and manage content from the UI securely.

4. Image Optimization and Compression

- **Why:** Users can currently upload large images, which affects site load times.
- **Enhancement:** Implement image compression on the client side or backend using tools like sharp or compression libraries.
- **Impact:** Improves performance, faster image loading, and better experience on slower connections.

5. Deployment to Cloud Platforms

- **Why:** Application is currently limited to local development environments.
- **Enhancement:**

  Deploy the React frontend on Vercel or Netlify

  Deploy the Node.js backend on Render, Railway, or Heroku

  Use MongoDB Atlas for cloud database
- **Impact:** Makes Househunt publicly accessible for demo, testing, or production use.