Project Documentation format

1. Introduction

- **Project Title:** HouseHunt – House Rent Web Application
- **Team Members:**
  - Kummitha Ganesh Kumar Reddy
  - Marati Pavani Bai
  - Narayanignaneshwar
  - Nayakalu Divyasree

2. Project Overview

- **Purpose:** The purpose of the house rent web application is to provide a user-friendly platform for individuals seeking rental properties and for property owners looking to list their properties for rent. The primary goal is to simplify the rental process by offering a seamless, efficient, and comprehensive solution for both renters and property owners. The application aims to bridge the gap between renters and property owners, making the search, inquiry, and booking processes more straightforward and transparent.
- **Features:**
  - **User Registration and Authentication:** Secure registration and login for renters and property owners.
  - **Dashboard:** Personalized dashboards for renters and property owners, showcasing relevant information and options.
  - **Property Listings:** Detailed property listings with descriptions, photos, rental information, and owner contact details.
  - **Search and Filters:** Advanced search functionality with filters for location, rent range, number of bedrooms, and other criteria.

3. Architecture

- **Frontend:** Using React

The frontend of the house rent web application is built using React, providing a dynamic and responsive user interface.

Login Page:

- **Components:**
  - LoginForm: Handles user input for email and password, form submission.
  - AuthContext: Manages user authentication state and provides authentication functions across the app.
- **State Management:** Use of React hooks (e.g., useState, useContext) to manage form input and authentication state.
- **Validation:** Form validation for email and password fields using custom validation functions or libraries like Formik and Yup.

Property Posting Page:

- **Components:**
    - PropertyForm: Contains input fields for property details (title, description, rent, location, photos).
    - FileUpload: Manages photo uploads and previews.
- **State Management:** Use of hooks (e.g., useState, useReducer) to handle form input and photo uploads.
- **Validation:** Form validation for property details, ensuring all required fields are filled and photos are uploaded.

Main Page:

- **Components:**
    - PropertyList: Displays a list of available rental properties.
    - PropertyCard: Represents individual property items with brief details and a link to full property details.
    - Header: Contains navigation links, search bar, and user menu.
- **State Management:** Use of hooks (e.g., useState, useEffect) to fetch and display property data.
- **Styling:** Use of CSS modules or styled-components for component-specific styling.

Search Bar:

- **Components:**
    - SearchBar: Contains input fields and dropdowns for search criteria (location, rent range, number of bedrooms).
- **State Management:** Use of hooks (e.g., useState) to manage search input and trigger search functionality.
- **Integration:** Fetches filtered property data based on search criteria and updates the PropertyList component.

- **Backend:**. Using Node.js and Express.js

The backend of the house rent web application is built using Node.js and Express.js, providing RESTful APIs for frontend interaction.

Login Endpoint:

- **Routes:**
    - POST /api/login: Authenticates user and returns a JWT token.
- **Middleware:**
    - authMiddleware: Protects routes requiring authentication.
- **Database Interaction:** Fetch user data from MongoDB, validate passwords

Property Posting Endpoint:

- **Routes:**
    - POST /api/propertyForm: Allows authenticated users to post a new property.
- **Controllers:**

- propertyController.createProperty: Handles property data validation, saves new property to the database.

- **Middleware:**
  - authMiddleware: Ensures only authenticated users can post properties.

- **Database Interaction:** Save property details to MongoDB, including handling file uploads for property photos.

- **Database:**

User Registration and Authentication:

- Create: Register a new user with hashed password.
- Read: Find a user by email for login authentication.

Property Posting:

- Create: Save new property details to the database.
- Read: Retrieve properties based on owner or search criteria.

4. Setup Instructions

- **Prerequisites:**

  Before setting up the house rent web application, ensure you have the following software dependencies installed:

- **Node.js**: JavaScript runtime for running the backend server.
- **npm** (Node Package Manager): Comes with Node.js, used to manage project dependencies.
- **MongoDB**: NoSQL database for storing user and property data.
- **Git**: Version control system for cloning the repository.

- **Installation:**

  Clone the repository:

  git clone https://github.com/yourusername/househunt.git

  cd househunt

  Install backend dependencies:

  cd househunt-backend

  npm install

  Set up environment variables:

  Create a .env file in the househunt-backend directory and add the following:

  MONGODB_URI=your_mongodb_connection_string

  PORT=5000

  Install frontend dependencies:

  cd ../househunt-frontend

  npm install

  Running the Application

  Start the backend server:

  cd househunt-backend

npm run dev

Start the frontend server:

cd ../househunt-frontend

npm start

The application will be available at http://localhost:3000.

5. Folder Structure

- **Client:** Describe the structure of the React frontend.
- **Server:** Explain the organization of the Node.js backend.

6. Running the Application

- To run the application locally, follow these steps:

  - **Frontend**

1. Navigate to the client directory:
   cd client
2. Install the necessary dependencies:
   npm install
3. Start the frontend server:
   npm start

  - **Backend**

1. Navigate to the server directory:
   cd server
2. Install the necessary dependencies:
   npm install
3. Start the backend server:
   npm start

7. API Documentation

- **User Authentication**

1. **Register a new user**
   a. **Endpoint:** POST /api/user/register
   b. **Request Body:** { "username": "exampleUser", "password": "examplePass", "email": "user@example.com" }
   c. **Response:** { "message": "User registered successfully", "user": { "id": "userId", "username": "exampleUser", "email": "user@example.com" } }
2. **Login a user**
   a. **Endpoint:** POST /api/user/login

b. **Request Body:** { "username": "exampleUser", "password": "examplePass" }

c. **Response:** { "token": "jwtToken", "user": { "id": "userId", "username": "exampleUser", "email": "user@example.com" } }

- **User Management**

1. **Get user information by ID**

   a. **Endpoint:** GET /api/user/:id

   b. **Response:** { "id": "userId", "username": "exampleUser", "email": "user@example.com" }

2. **Update user information by ID**

   a. **Endpoint:** PUT /api/user/:id

   b. **Request Body:** { "username": "updatedUser", "email": "updated@example.com" }

   c. **Response:** { "message": "User updated successfully", "user": { "id": "userId", "username": "updatedUser", "email": "updated@example.com" } }

- **Workout Plans**

1. **Get all workout plans**

   a. **Endpoint:** GET /api/workoutplans

   b. **Response:** [{ "id": "workoutPlanId1", "name": "Workout Plan 1", "details": "Details of Workout Plan 1" }, { "id": "workoutPlanId2", "name": "Workout Plan 2", "details": "Details of Workout Plan 2" }]

2. **Create a new workout plan**

   a. **Endpoint:** POST /api/workoutplans

   b. **Request Body:** { "name": "New Workout Plan", "details": "Details of the new workout plan" }

   c. **Response:** { "message": "Workout plan created successfully", "workoutPlan": { "id": "newWorkoutPlanId", "name": "New Workout Plan", "details": "Details of the new workout plan" } }

- **Equipment**

1. **Get all equipment**

   a. **Endpoint:** GET /api/equipment

   b. **Response:** [{ "id": "equipmentId1", "name": "Equipment 1", "type": "Type 1" }, { "id": "equipmentId2", "name": "Equipment 2", "type": "Type 2" }]

2. **Add new equipment**

   a. **Endpoint:** POST /api/equipment

   b. **Request Body:** { "name": "New Equipment", "type": "Type of the new equipment" }

   c. **Response:** { "message": "Equipment added successfully", "equipment": { "id": "newEquipmentId", "name": "New Equipment", "type": "Type of the new equipment" } }

- **Monthly Plans**

1. **Get all monthly plans**

   a. **Endpoint:** GET /api/monthlyplans

b. **Response:** [{ "id": "monthlyPlanId1", "name": "Monthly Plan 1", "details": "Details of Monthly Plan 1" }, { "id": "monthlyPlanId2", "name": "Monthly Plan 2", "details": "Details of Monthly Plan 2" }]

2. **Create a new monthly plan**

   a. **Endpoint:** POST /api/monthlyplans

   b. **Request Body:** { "name": "New Monthly Plan", "details": "Details of the new monthly plan" }

   c. **Response:** { "message": "Monthly plan created successfully", "monthlyPlan": { "id": "newMonthlyPlanId", "name": "New Monthly Plan", "details": "Details of the new monthly plan" } }

- **Street Map Integration**

1. **Get map data for specified locations**

   a. **Endpoint:** GET /api/streetmap

   b. **Request Params:** { "latitude": "latitudeValue", "longitude": "longitudeValue" }

   c. **Response:** { "location": { "latitude": "latitudeValue", "longitude": "longitudeValue" }, "mapData": "Map data details" }

2. **Fetch and store new map data**

   a. **Endpoint:** POST /api/streetmap

   b. **Request Body:** { "latitude": "latitudeValue", "longitude": "longitudeValue" }

   c. **Response:** { "message": "Map data fetched and stored successfully", "mapData": { "location": { "latitude": "latitudeValue", "longitude": "longitudeValue" }, "mapDetails": "Details of the map data" } }

8. Authentication

In the HouseHunt project, authentication and authorization are handled using JSON Web Tokens (JWT). Here's a brief overview of the process:

1. **User Registration**

- **Endpoint:** POST /api/user/register
- User information (username, password, email) is stored in MongoDB after hashing the password using bcrypt.

2. **User Login**

- **Endpoint:** POST /api/user/login
- Users provide their credentials. If valid, a JWT token is generated and sent back to the client.

3. **JWT Token**

- Tokens are generated using a secret key and include user ID and other payload information.
- Example token format:

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

4. **Protected Routes**

- Middleware checks for a valid JWT token in the Authorization header for protected routes.
- Example middleware:

```
const jwt = require('jsonwebtoken');
const config = require('../config');
function authenticateToken(req, res, next) {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).send('Access Denied');

  try {
    const verified = jwt.verify(token, config.jwtSecret);
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).send('Invalid Token');
  }
}
```

5. **Session Management**

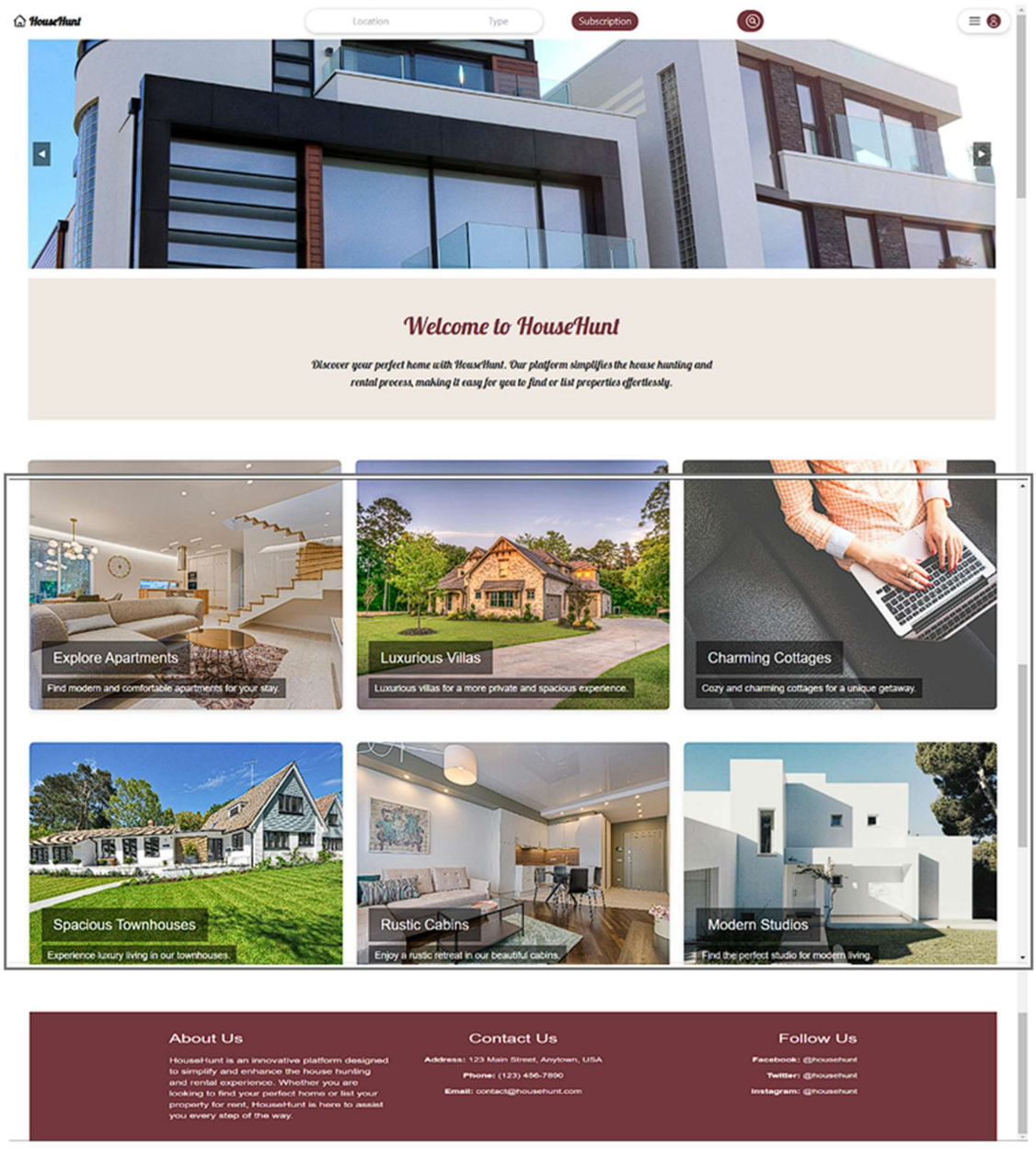
- Tokens are stored client-side (e.g., localStorage) and included in the Authorization header of each request.

6. **Example Usage**

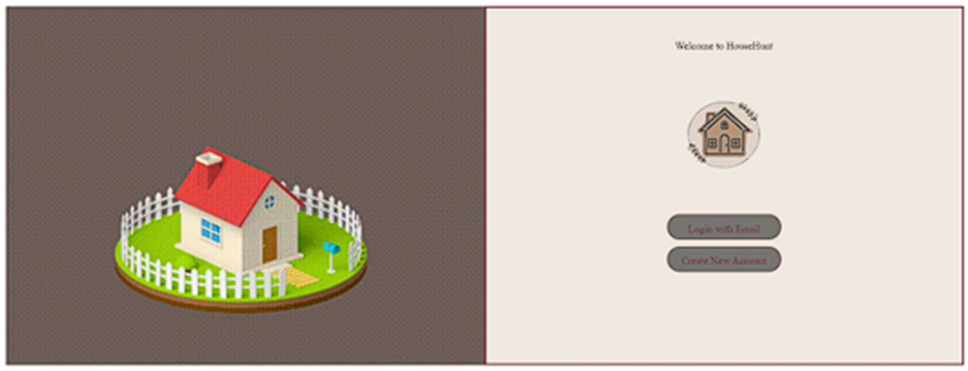
- After login, the JWT token is included in the Authorization header:

```
GET /api/protectedEndpoint
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

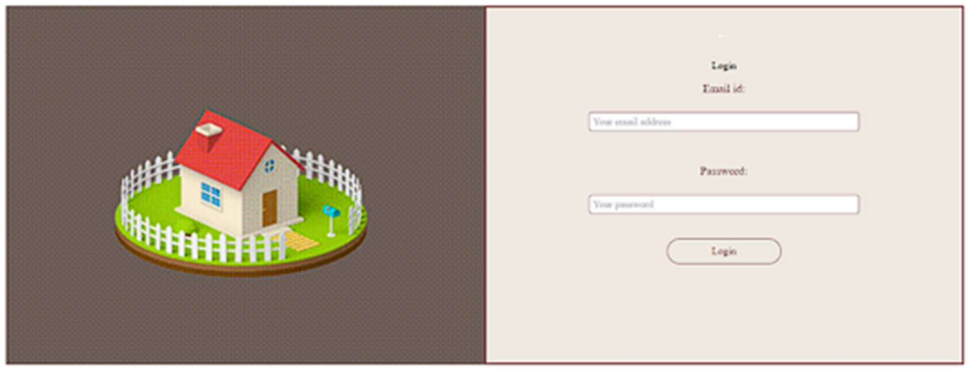
9. User Interface


HouseHunt
Location
Type
Subscription
Welcome to HouseHunt
Discover your perfect home with HouseHunt. Our platform simplifies the house hunting and rental process, making it easy for you to find or list properties effortlessly.
Explore Apartments
Find modern and comfortable apartments for your stay.
Luxurious Villas
Luxurious villas for a more private and spacious experience.
Charming Cottages
Cozy and charming cottages for a unique getaway.
Spacious Townhouses
Experience luxury living in our townhouses.
Rustic Cabins
Enjoy a rustic retreat in our beautiful cabins.
Modern Studios
Find the perfect studio for modern living.
About Us
HouseHunt is an innovative platform designed to simplify and enhance the house hunting and rental experience. Whether you are looking to find your perfect home or list your property for rent, HouseHunt is here to assist you every step of the way.
Contact Us
Address: 123 Main Street, Anytown, USA
Phone: (123) 456-7890
Email: contact@househunt.com
Follow Us
Facebook: @househunt
Twitter: @househunt
Instagram: @househunt

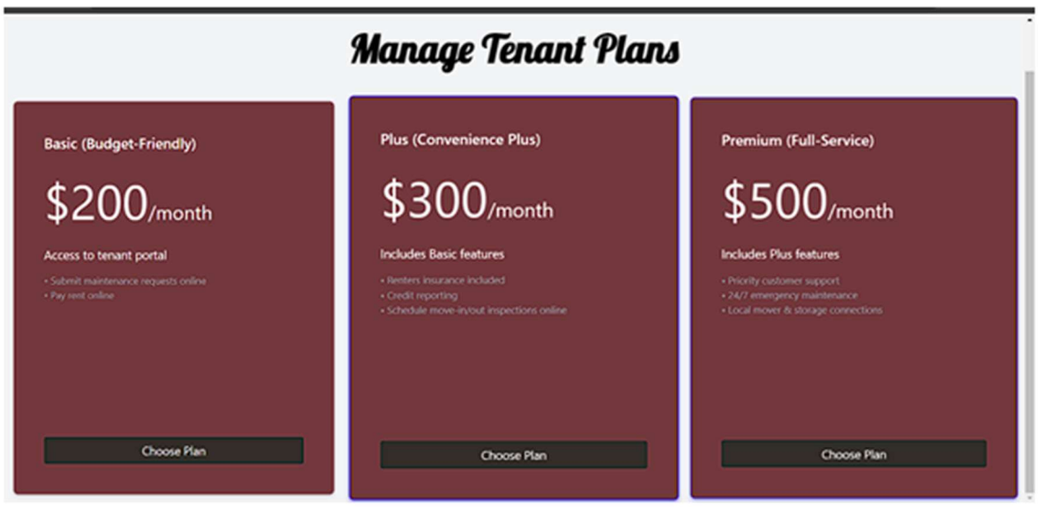
HouseHunt Location Type Subscription

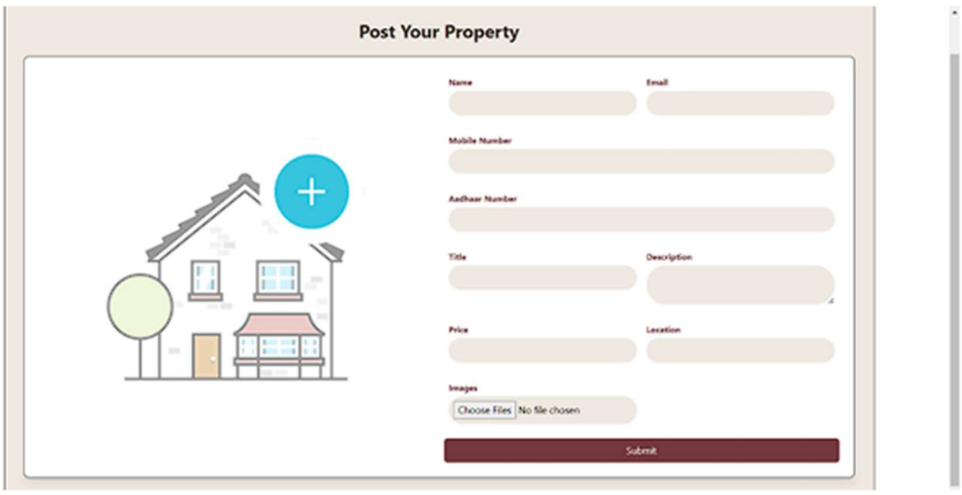


HouseHunt Location Type Subscription



HouseHunt Location Type Subscription





10. Testing

The testing strategy for the house rent web application aims to ensure the application is robust, reliable, and free of critical bugs. The strategy encompasses different levels of testing to cover various aspects of the application:

1. **Unit Testing:**
    a. **Purpose:** Verify individual components, functions, and modules work correctly in isolation.
    b. **Scope:** Test individual React components, utility functions, and backend routes/controllers.
2. **Integration Testing:**
    a. **Purpose:** Ensure different parts of the application work together as expected.
    b. **Scope:** Test interactions between frontend components and backend APIs, database operations, and overall data flow.
3. **End-to-End (E2E) Testing:**
    a. **Purpose:** Validate the application works as a whole, simulating real user scenarios.

b. **Scope:** Test complete workflows such as user registration, login, property posting, searching, and booking.

4. **Manual Testing:**

   a. **Purpose:** Identify issues that automated tests might miss, focusing on usability and user experience.

   b. **Scope:** Conduct exploratory testing and perform ad-hoc testing on the application.

Tools Used

1. **Unit Testing Tools:**

   a. **Jest:** A JavaScript testing framework for unit testing React components and utility functions.

   b. **React Testing Library:** A library for testing React components, focusing on user interactions and ensuring components render correctly.

2. **Integration Testing Tools:**

   a. **Jest:** Also used for integration tests to validate interactions between components and services.

   b. **Supertest:** A library for testing HTTP requests, used for testing backend APIs.

3. **End-to-End (E2E) Testing Tools:**

   a. **Cypress:** A robust E2E testing framework that simulates user interactions and validates the application's overall functionality.

4. **Manual Testing Tools:**

   a. **Browser Developer Tools:** Used for manual testing to inspect elements, debug issues, and monitor network requests.

   b. **JIRA/Trello:** Project management tools for tracking bugs and organizing manual testing efforts.

11. Screenshots or Demo:

👉(https://drive.google.com/drive/folders/1Jgow8_KkyyQPgJiSz1_IruzSbc7h7W8v?usp=drive_link)

12. Known Issues

1. Registration Form Email Validation

- **Description:** The registration form sometimes fails to accept valid email addresses, showing an "invalid email format" error.
- **Impact:** Users may be unable to register, leading to frustration and potential loss of new users.
- **Workaround:** Users can try using a different email address or refreshing the page before attempting registration again.
- **Status:** Open, under investigation.

2. Filter for Number of Bedrooms Not Working

- **Description:** The filter for selecting the number of bedrooms in the search bar is not filtering properties correctly.
- **Impact:** Users may see properties that do not match their specified criteria, leading to a poor search experience.

- **Workaround:** Users can manually browse through the listings or use other filters to narrow down the search results.

- **Status:** In progress, expected to be fixed in the next release.

3. Inquiry Form Submission Fails Intermittently

- **Description:** The inquiry form submission sometimes fails with a server error message.

- **Impact:** Users may be unable to contact property owners, leading to missed rental opportunities.

- **Workaround:** Users can try resubmitting the form after some time or contacting the owner directly via the provided contact information.

- **Status:** Open, root cause analysis in progress.

4. Booking Confirmation Notification Not Received

- **Description:** Users do not receive notification emails after their booking request has been confirmed by the property owner.

- **Impact:** Users may be unaware of the status of their booking request, causing confusion and potential booking overlaps.

- **Workaround:** Users should manually check their dashboard for booking status updates.

- **Status:** Open, notification system review underway.

5. Property Photo Upload Issues

- **Description:** Property photo uploads occasionally fail or take an extended time to process.

- **Impact:** Property owners may face difficulties in uploading images, resulting in incomplete property listings.

- **Workaround:** Owners can try uploading photos one at a time or use smaller file sizes.

- **Status:** Open, performance improvements and bug fixes planned.

6. Responsive Design Glitches

- **Description:** Some UI elements do not render correctly on mobile devices or small screen sizes.

- **Impact:** Users on mobile devices may experience a suboptimal browsing experience.

- **Workaround:** Users can switch to a desktop or larger screen device for better functionality.

- **Status:** Open, responsive design enhancements in progress.

7. Slow Performance on Property Listings Page

- **Description:** The property listings page loads slowly when there are a large number of properties.

- **Impact:** Users may experience delays and sluggish performance, affecting the overall user experience.

- **Workaround:** Users can apply filters to reduce the number of listings displayed.

- **Status:** Open, performance optimization strategies being implemented.

13. Future Enhancements

1. Enhanced Search Functionality

- **Advanced Filters:** Implement more granular filters for search results, such as neighborhood amenities, proximity to schools, and public transportation.

- **Voice Search:** Integrate voice search capabilities for a more intuitive user experience.

- **Saved Searches and Alerts:** Allow users to save their search criteria and receive notifications when new listings match their preferences.

2. User Experience Improvements

- **Interactive Maps:** Incorporate interactive maps with property overlays, neighborhood statistics, and nearby points of interest.
- **Virtual Tours:** Add virtual tour functionality or 360-degree views of properties to give users a better sense of the space.
- **Enhanced Property Details:** Include more comprehensive property details such as energy efficiency ratings, property history, and neighborhood crime statistics.

3. Personalization Features

- **Recommendation Engine:** Develop a recommendation system based on user preferences and behavior to suggest properties that fit their criteria.
- **User Profiles:** Allow users to create profiles where they can manage their preferences, saved properties, and communication with agents.

4. Integration with Third-Party Services

- **Mortgage Calculators:** Integrate mortgage calculators to help users estimate monthly payments and affordability.
- **Insurance Quotes:** Provide options to get insurance quotes for the properties they are interested in.

5. Mobile App Development

- **Mobile Application:** Create a mobile app version of the platform to enhance accessibility and provide a seamless experience on-the-go.

6. Data Analytics and Reporting

- **Analytics Dashboard:** Develop a dashboard for users and agents to analyze market trends, property values, and user engagement.
- **User Feedback Collection:** Implement mechanisms for collecting user feedback to continuously improve the platform based on user suggestions.

7. Security and Performance Enhancements

- **Data Encryption:** Ensure all user data is encrypted and secure.
- **Performance Optimization:** Regularly optimize the application for speed and responsiveness.

8. Internationalization and Localization

- **Multi-Language Support:** Add support for multiple languages to reach a broader audience.
- **Localized Listings:** Allow users to search for properties in different regions or countries.

9. AI and Machine Learning Integration

- **Predictive Analytics:** Use machine learning to predict market trends and property values.
- **Chatbots:** Implement AI-driven chatbots to provide instant support and answer user queries.