Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title :** HouseHunt: Finding Your Perfect Rental Home
- **Team Leader :** Bandaru Raatna Sai
- **Team member :** Kanthete Naga Durga Yaswanth
- **Team member :** Karanki Naga Mounika
- **Team member :** Bandi Satwika

2. Project Overview

Purpose

The purpose of the HouseHunt project is to design and develop a reliable and user-friendly rental platform that simplifies the process of finding, listing, and booking rental homes by connecting renters, property owners, and administrators through a secure and efficient digital system.

The Goals of the Project are as follows:

• To simplify the rental property search process for renters
• To reduce the time and effort required to find suitable rental homes
• To provide a secure and transparent platform for renters and property owners
• To enable property owners to efficiently manage listings and bookings
• To ensure role-based access and proper platform governance through admin control
• To improve user experience with an intuitive and responsive interface
• To maintain reliable and scalable data management using modern web technologies

Features

• User registration and secure login with role-based access (Renter, Owner, Admin)
• Advanced property search with filters for location, price, and property type
• Detailed property listings with images, descriptions, and owner contact details
• Easy property booking with real-time booking status updates
• Property management for owners (add, update, delete listings)
• Booking request approval and rejection by property owners
• Admin approval and verification of property owners
• Admin control to manage users and remove inappropriate listings
• Responsive and user-friendly interface for web and mobile devices
• Secure data handling and scalable backend architecture

3. Architecture

Frontend

The frontend of HouseHunt is built using React.js, following a component-based architecture. The application is divided into reusable components such as login, registration, property listing, booking, and dashboard modules. React Router is used for client-side navigation, enabling smooth page transitions without reloading. Axios is integrated to handle communication with the backend through RESTful APIs. UI libraries like Bootstrap and Material UI are used to ensure a responsive, interactive, and user-friendly interface across different devices.

Backend

The backend architecture is developed using Node.js with the Express.js framework. Express is used to create RESTful API endpoints that handle user authentication, property management, booking operations, and admin controls. Middleware such as JWT authentication, CORS, and body-parser is implemented to ensure secure request handling and data validation. The backend follows a modular structure with separate routes, controllers, and services, making the system scalable and easy to maintain.

Database

The database layer uses MongoDB, a NoSQL document-oriented database, to store application data. The database schema consists of collections such as Users, Properties, and Bookings. Each collection is designed to store relevant information, including user credentials, property details, and booking status. Mongoose is used as an Object Data Modeling (ODM) library to define schemas, manage relationships, and perform CRUD operations. This setup ensures efficient data storage, fast retrieval, and seamless integration with the backend services.

4. Setup Instructions

Prerequisites

To set up and run the HouseHunt project, the following software and tools are required. Node.js and npm must be installed to run the backend server and manage project dependencies. MongoDB is required for database storage, either as a local installation or through MongoDB Atlas. A code editor such as Visual Studio Code is recommended for development. Basic knowledge of JavaScript, React, and REST APIs is also necessary to understand and modify the application.

Installation

First, clone the project repository to your local system using a version control tool such as Git. After cloning, navigate to the project directory and install the required dependencies separately for the frontend and backend using npm. Create an environment variables file in the backend directory to store sensitive information such as the server port, MongoDB connection string, and JWT secret key. Once dependencies are installed and environment variables are configured, start the backend server and then run the frontend application. The application can be accessed through the local development URL in a web browser.

5. Folder Structure

Client(Frontend)

The client side of the HouseHunt project is developed using React and follows a modular and component-based structure. The frontend folder contains a public directory for static assets and a src directory that holds the core application logic. Inside src, components are organized into modules based on user roles such as admin, owner, renter, and common components. This separation helps in maintaining role-based views and improves code readability. The main application files such as App.js and main.jsx handle routing and application initialization, while CSS files manage global and component-level styling. This structured approach makes the frontend scalable, reusable, and easy to maintain.

Server(Backend)

The server side of the HouseHunt project is built using Node.js and Express.js and follows a well-organized MVC-like architecture. The backend folder includes a config directory for database connection setup, controllers for handling business logic, routes for defining API endpoints, and models for MongoDB schemas such as User, Property, and Booking. Middleware is used for authentication and request validation to ensure secure access. An uploads directory is maintained for storing property images, and the main server file initializes the Express application, connects to the database, and registers all routes. This organized backend structure ensures clean separation of concerns, scalability, and easier maintenance.

6. Running the Application

To run the HouseHunt application locally, both the frontend and backend servers must be started separately.

Frontend

Navigate to the frontend (client) directory and install dependencies if not already installed. Then start the React development server using the following commands:

```
cd frontend

npm install

npm start
```

This will start the frontend application, which can be accessed in a web browser at http://localhost:3000.

Backend

Navigate to the backend (server) directory and install the required dependencies. Then start the Node.js server using the following commands:

```
cd backend

npm install

npm start
```

The backend server will start running on the configured port and will handle API requests from the frontend.

7. API Documentation

The backend of the HouseHunt application exposes RESTful APIs to support user authentication, property management, booking operations, and admin functionalities. Below is a structured documentation of the main endpoints, including request methods, parameters, and example responses

1. User Authentication APIs

- **POST/api/auth/register**
  - Description: Registers a new user.
  - Parameters:
    - username (string, required)
    - email (string, required)
    - password (string, required)
  - Example Response:

```
{
  "message": "User registered successfully" }
```

- **POST /api/auth/login**
  - Description: Authenticates a user and returns a token.
  - Parameters:
    - email (string, required)
    - password (string, required)
  - Example Response:

```
{
  "token": "JWT_TOKEN_HERE"
}
```

- **POST /api/auth/logout**
  - Description: Logs out the user.

    Example Response:

```
{
  "message": "User logged out successfully"
}
```

2. User Management • **GET**

/api/users

- Description: Retrieves all users.

  Example Response:

```
[
 {
  "id": "USER_ID",
  "username": "USERNAME",
  "email": "EMAIL"
 }
]
```

- **PUT /api/users/**
  - Description: Updates user information by ID.
  - Parameters:
    - username (string, optional) ▪ email (string, optional) o Example Response:

```
{
 "message": "User updated successfully"
}
```

- **DELETE /api/users/**
  - Description: Deletes a user by ID.

    Example Response:

```
{
 "message": "User deleted successfully"
}
```

- **POST /api/users/save**

- Description: Saves a post for the user.

  oParameters: postId (string, required)

  Example Response:

```
{
  "message": "Post saved successfully"
}
```

- **GET /api/users/profilePosts**

  - Description: Retrieves posts saved by the user.

  Example Response:

```
[
  {
    "id": "POST_ID",
    "title": "Post Title",    "description":
"Post Description"
  } ]
```

3.Post Management •

GET /api/posts

- Description: Retrieves all posts.

  Example Response:

```
[
  {
      "id": "POST_ID",
     "title": "Post Title",
    "description": "Post Description"
  }
]
```

- **GET /api/posts/**

- Description: Retrieves a post by ID.

  Example Response:

  ```
  {
     "id": "POST_ID",
    "title": "Post Title",  "description":
  "Post Description"
  }
  ```

- **POST /api/posts**

  - Description: Creates a new post. o

    Parameters

    - title (string, required)
    - description (string, required)

  Example Response:

  ```
  {
    "message": "Post created successfully"
  }
  ```

- **PUT /api/posts/**

  - Description: Updates a post by ID.oParameters:

    - title (string, optional)
    - description (string, optional)

  Example Response:

  ```
  {
    "message": "Post updated successfully"
  }
  ```

- **DELETE /api/posts/**

  - Description: Deletes a post by ID.

    Example Response:

  ```
  {
    "message": "Post deleted successfully"
  .}
  ```

8. Authentication

In the HouseHunt project, authentication and authorization are implemented using a secure token-based mechanism to ensure controlled access to the application.

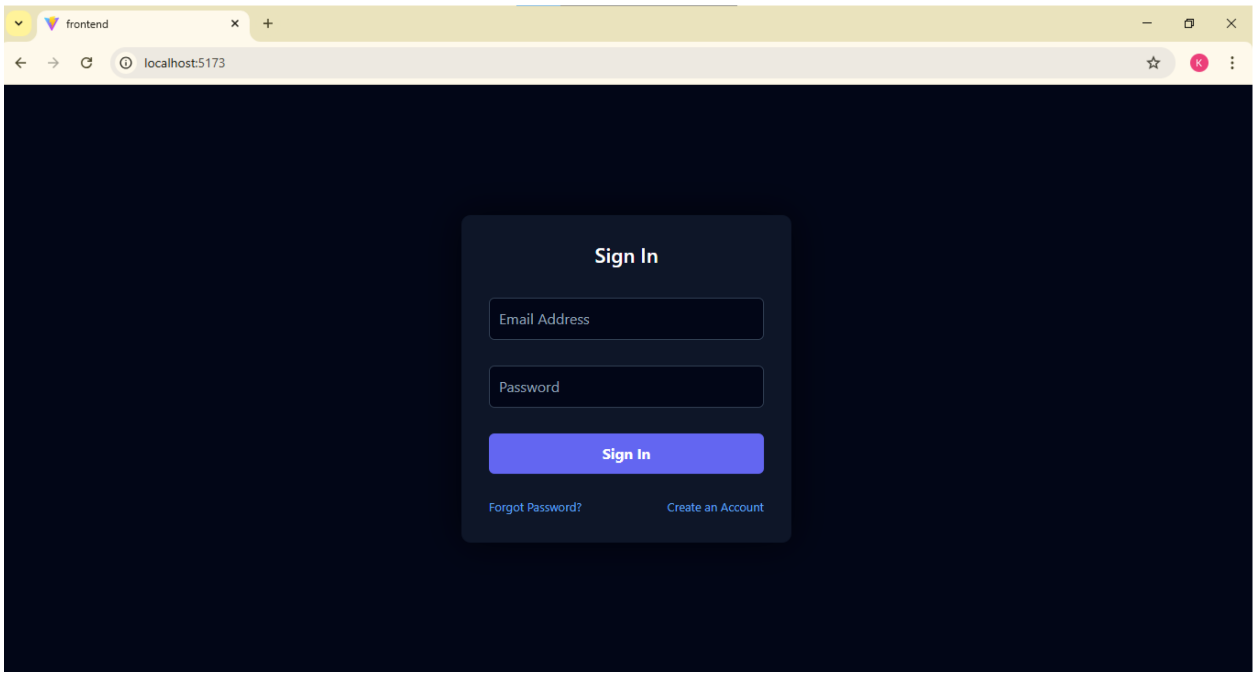
User authentication is handled through JSON Web Tokens (JWT). When a user registers or logs in with valid credentials, the backend verifies the information and generates a JWT. This token is sent to the frontend and stored securely (for example, in local storage). The token is included in the Authorization header for all subsequent API requests that require authentication.

Authorization is enforced using middleware in the backend. The authentication middleware verifies the JWT on each protected request and extracts the user's identity and role (renter, owner, or admin). Based on this role, access to specific routes is granted or restricted. For example, only owners can add or manage properties, renters can book properties, and administrators can manage users and listings.

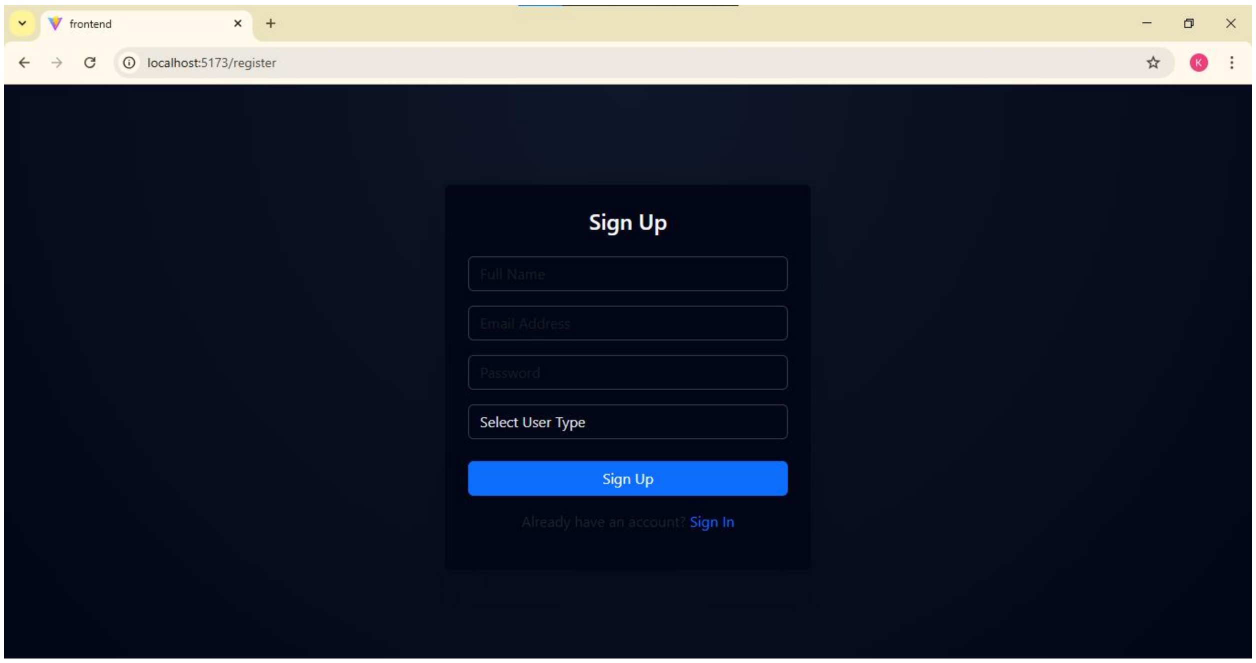
Passwords are securely stored using hashing techniques (bcryptjs), ensuring that plain-text passwords are never saved in the database. Environment variables are used to store sensitive information such as JWT secret keys and database credentials. This approach ensures secure session handling, role-based access control, and protection of user data throughout the application.

9. User Interface

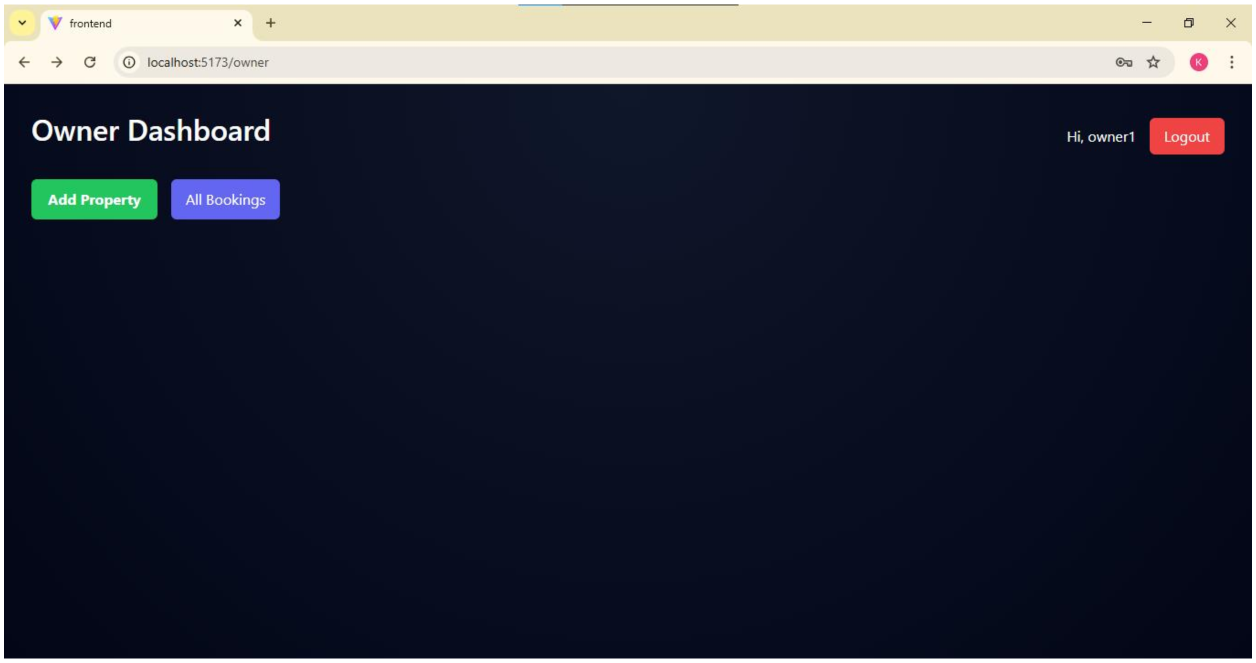
Combined Login Page for Renter and Owner



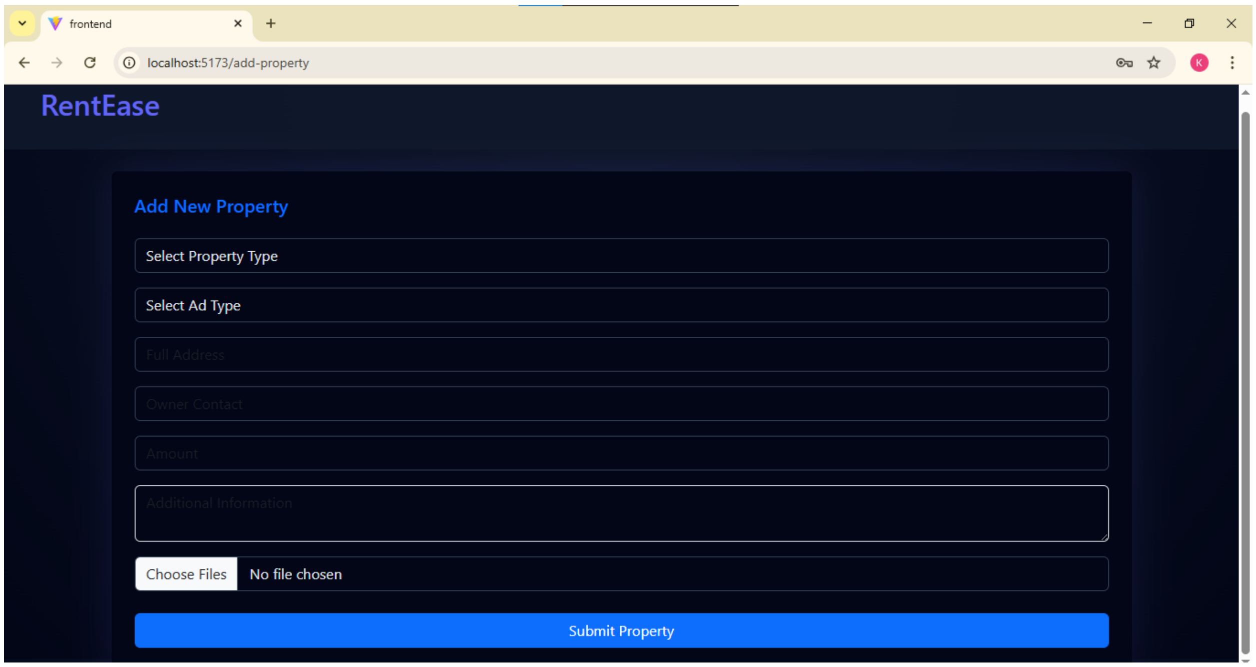
SignUp Page



Owner Dashboard



Owner Dashboard-Add property



Add New Property

Select Property Type

Select Ad Type

Full Address

Owner Contact

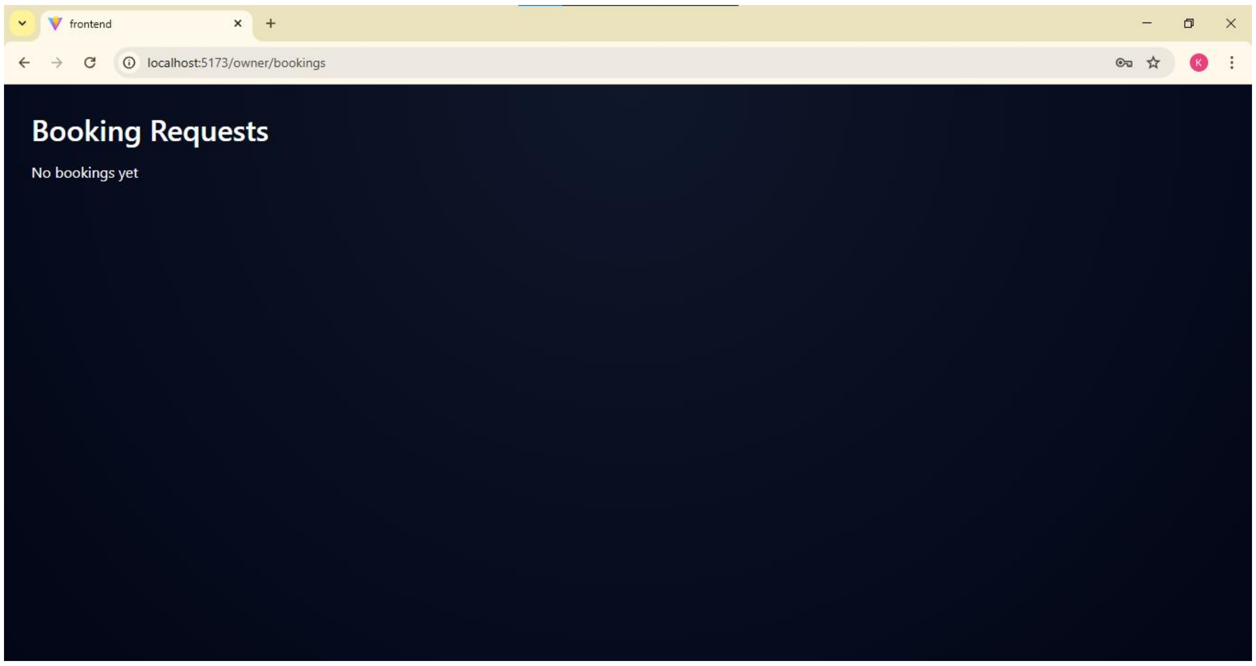
Amount

Additional Information

Choose Files No file chosen

Submit Property

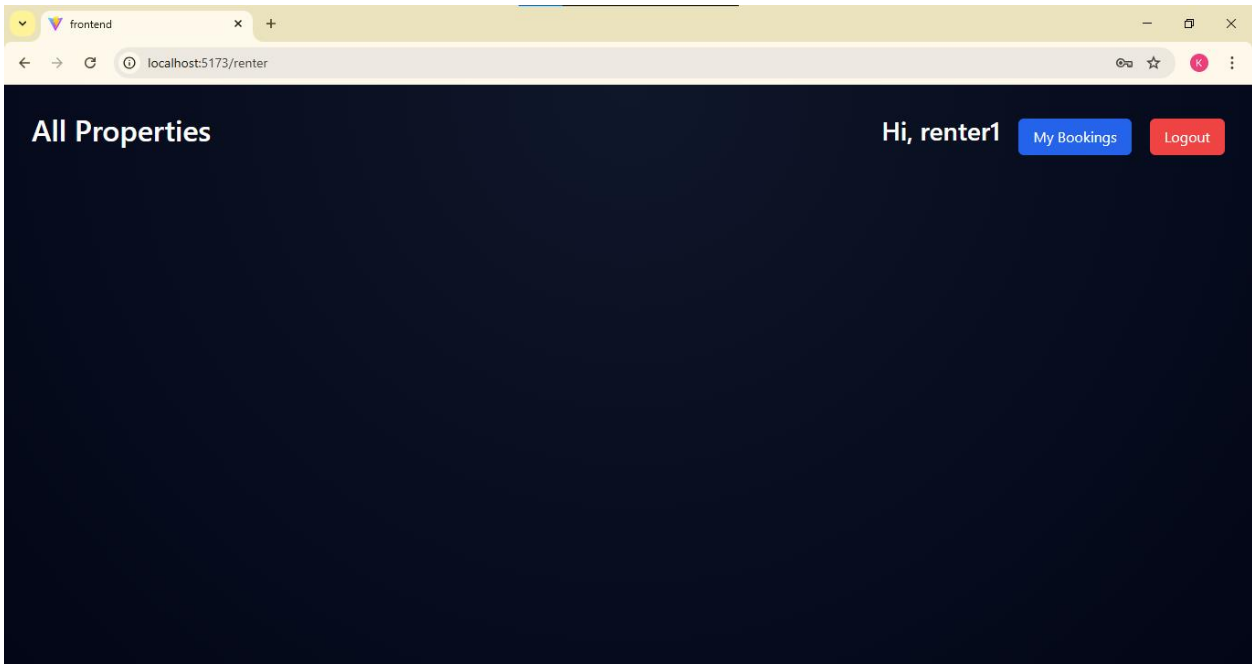
Owner Dashboard - All Bookings Page



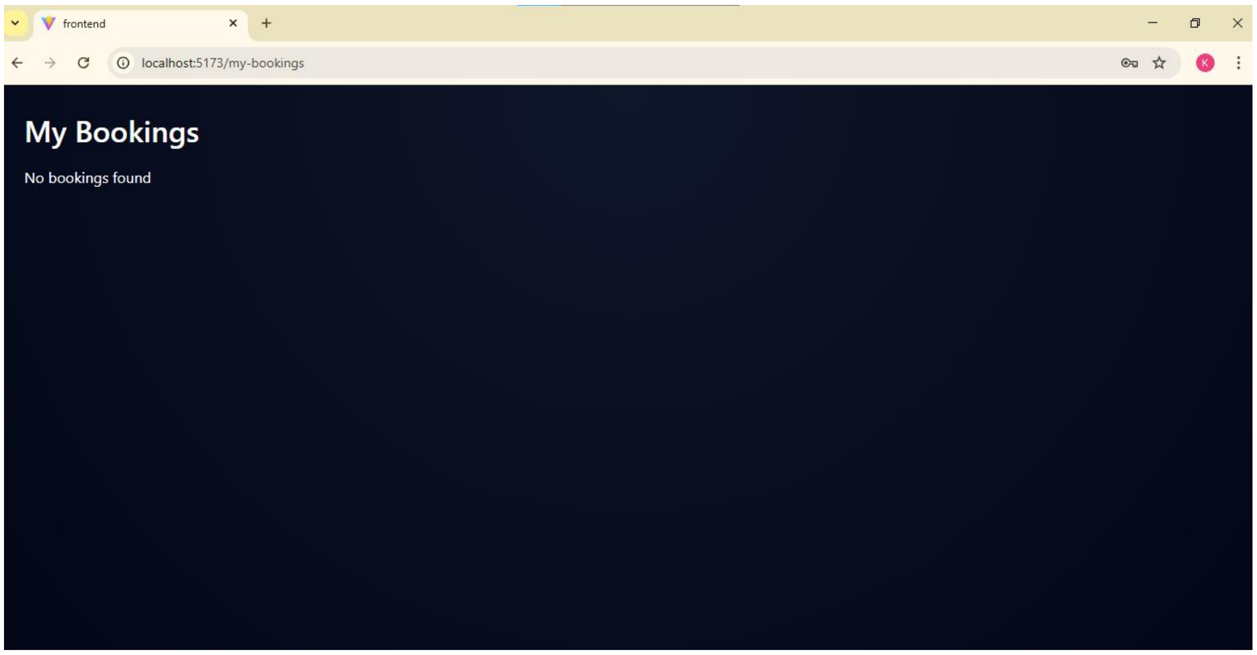
Booking Requests

No bookings yet

Renter Dashboard



Renter Dashboard – All bookings Page



No bookings found

10. Testing

The testing approach for the HouseHunt application focuses on ensuring reliability, correctness, and smooth user experience across both frontend and backend components. A combination of manual testing and API-level testing was used due to the full-stack nature of the project.

The testing strategy was divided into the following phases:

1. Unit Testing (Backend Logic Validation)
    - Individual backend controllers such as authentication, property management, and booking handling were tested by triggering API endpoints with valid and invalid inputs.
    - Edge cases like missing fields, unauthorized access, and incorrect data formats were verified to ensure proper error handling.
2. Integration Testing
    - End-to-end workflows were tested, such as:
        - User registration → login → role-based dashboard redirection
        - Owner adding a property → renter viewing the property → renter booking → owner approving/rejecting
    - Database interactions using MongoDB were verified to ensure data consistency and correct relationships between users, properties, and bookings.
3. Frontend Functional Testing
    - Manual testing was performed on all major UI flows:
        - Login and registration forms
        - Owner dashboard actions (Add Property, View Bookings, Approve/Reject)
        - Renter dashboard actions (View Properties, Image Carousel, Book Property)
    - UI responsiveness and conditional rendering (based on user roles) were verified.
4. Error Handling and Security Testing
    - Authentication-protected routes were tested to confirm that unauthorized users cannot access restricted pages.
    - Token-based authorization was validated for all booking and property-related actions.
    - Invalid requests were tested to ensure meaningful error messages are returned.

Testing Tools Used

- Browser Developer Tools
    - Console logs and network tab were used to debug API calls, inspect request headers, and track frontend errors.
    - Helped identify issues related to missing tokens, incorrect routes, and rendering errors.
- MongoDB Compass
    - Used to verify database entries after operations such as property creation and booking requests.
    - Helped confirm correct schema relationships and data persistence.
- Manual UI Testing
    - Performed by interacting with the application as both Owner and Renter roles.
    - Ensured real-world usability, navigation flow, and UI consistency.

11. Screenshots or Demo

https://drive.google.com/file/d/1P7Vx_EgTiWRb071xeYuZdxU8ANqgbtMc/view?usp=sharing

12. Known Issues

Despite implementing all core functionalities successfully, the following known issues and limitations exist in the current version of the HouseHunt application:

1. Limited Automated Testing
   - The project primarily relies on manual testing and API testing.
   - Automated test cases using frameworks like Jest or Cypress are not implemented in the current version.
2. Image Handling Limitations
   - Uploaded property images are stored locally on the server.
   - If the server restarts or deployment environment changes, images may need to be re-uploaded.
   - Cloud-based storage solutions (e.g., AWS S3 or Cloudinary) are not yet integrated.
3. Basic Error Messages
   - Some error messages displayed to users are generic and do not provide detailed explanations.
   - Improved user-friendly error handling can be added in future versions.
4. No Email Notifications
   - Users do not receive email notifications for booking approvals or rejections.
   - Booking status updates are only visible inside the dashboard.
5. Role-Based UI Dependency
   - Navigation and dashboard access depend entirely on user role data stored in local storage.
   - If local storage is cleared manually, the user must log in again to restore access.
6. Limited UI Customization
   - The UI is functional but minimal.
   - Advanced UI features such as animations, theme customization, and accessibility enhancements are not fully implemented.
7. Scalability Constraints
   - The current backend architecture is designed for small-scale usage.
   - Performance under heavy concurrent user load has not been stress-tested.

13. Future Enhancements

The HouseHunt application can be further improved by implementing the following enhancements in future versions:

1. Cloud-Based Image Storage
    - Integrate cloud storage services such as AWS S3 or Cloudinary to securely store and manage property images.
    - This will improve scalability and prevent data loss during server restarts or deployments.
2. Advanced Search and Filters
    - Add filters based on price range, location, property type, and ad type.
    - Implement sorting options such as lowest price, highest price, and newest listings.
3. Email and Notification System
    - Send email or in-app notifications for booking requests, approvals, and rejections.
    - Notify owners and renters about important updates in real time.
4. Online Payment Integration
    - Integrate secure payment gateways (e.g., Razorpay, Stripe) for booking confirmation or advance payments.
    - This will enhance trust and automate the rental process.
5. Improved Security Measures
    - Implement refresh tokens and role-based access control (RBAC).
    - Add rate limiting and request validation to prevent unauthorized access.
6. Automated Testing
    - Introduce automated unit and integration testing using tools like Jest and Cypress.
    - This will ensure better reliability and easier maintenance of the application.
7. Enhanced User Experience
    - Improve UI/UX with animations, responsive layouts, and accessibility features.
    - Add dark/light mode support.
8. Admin Dashboard
    - Create an admin panel to manage users, properties, and bookings.
    - Enable monitoring and moderation features.
9. Mobile Application
    - Develop a mobile version of the application using React Native or Flutter.
    - This will allow users to access the platform on the go.