

**INSTITUTE FOR ADVANCED COMPUTING AND**

**SOFTWARE DEVELOPMENT, AKURDI, PUNE**

**StayEn.Casa**

**PG-DAC February 2025**

Submitted By:

Group No: 50

**Roll No. Name of Student**

252001 Aadarsh Yadav

252107 Utkarsh Maurya

**Mr. Vaibhav Verulkar Mr. Prashant Deshpande**

Project Guide Centre Coordinator

# **ABSTRACT**

The real estate sector plays a crucial role in urban development, yet individuals often face challenges in renting or purchasing properties due to scattered listings, lack of transparency, and inefficient platforms. This project presents a Property Renting and Selling Platform, developed using Java Spring Boot Microservices, MongoDB & MySQL databases, and a React frontend, with JSON Web Token based authentication to ensure secure access.  
  
The system provides property owners with the ability to list properties, upload images via Supabase integration, and manage bookings. Users can search for properties using multiple filters such as location, price range, furnishing status, and category (Flat, Villa, Apartment). Bookings and payments are managed seamlessly through dedicated microservices.  
  
The architecture adopts a microservices approach, ensuring modularity, scalability, and ease of deployment. Performance evaluation demonstrates efficient API response times, ensuring smooth user interaction. The system not only addresses the existing gaps in property management platforms but also lays a foundation for future enhancements such as AI-powered property recommendations and payment gateway integrations.

**ACKNOWLEDGEMENT**

I take this occasion to thank God, almighty for blessing us with his grace and taking our endeavour to a successful culmination. I extend my heartfelt thanks to our esteemed guide, Mr. Vaibhav Verulkar for providing me with the right guidance and advice at the crucial juncture and showing me the right way. I sincerely thank our respected Centre Co-Ordinator, Mr. Prashant Deshpande, for allowing us to use the available facilities. I would also like to thank the other faculty members at this occasion. Last but not least, I would like to thank my friends and family for the support and encouragement they have given me during our work.

Aadarsh Yadav (250241220001)

Utkarsh Maurya (250241220211)

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Page No.** |
| 1 | [Introduction](#_INTRODUCTION) | 4 |
| 2 | [Project Architecture](#_PROJECT_ARCHITECTURE) | 7 |
| 3 | [Software Requirement Specification](#_SOFTWARE_REQUIREMENT_SPECIFICATION) | 10 |
| 4 | [Diagrams](#_DIAGRAMS) | 15 |
| 5 | [Database Design](#_DATABASE_DESIGN) | 23 |
| 6 | [Snapshots](#_SNAPSHOTS) | 26 |
| 7 | [Conclusion](#_CONCLUSION) |  |
| 8 | [References](#_REFERENCES) |  |

# **INTRODUCTION**

## **Background of the Problem**

The process of renting and purchasing properties has traditionally been time-consuming, costly, and prone to inefficiencies. In conventional systems, property seekers often relied on word of mouth, newspaper advertisements, or local brokers to find suitable accommodation or real estate investments. This approach often led to limited choices, lack of transparency, and increased risks of fraudulent transactions.

With the rapid growth of urbanization and migration to metropolitan cities, the demand for reliable property renting and selling services has surged significantly. However, most individuals continue to face several key challenges:

* **Fragmented Listings:** Properties are scattered across different platforms or brokers, making it difficult to compare options.
* **Lack of Verification:** Many listings lack authenticity, leading to cases of misrepresentation or scams.
* **Manual Processes:** Without digital solutions, property hunting remains tedious, involving physical visits and multiple intermediaries.
* **Limited Accessibility:** In smaller towns and cities, advanced property platforms are often unavailable, leaving users with outdated methods.
* **Poor User Experience:** Even when digital platforms exist, many lack advanced filters, secure authentication, or efficient booking systems.

These challenges highlight the pressing need for a **modern, secure, and user-friendly platform** that bridges the gap between property owners and seekers, providing transparency, reliability, and efficiency.

## **Importance of Real-Estate Platforms**

Real estate is not only a fundamental human necessity but also a critical driver of the economy. With the increasing shift towards digitalization, online real-estate platforms have become a vital medium for connecting property owners, buyers, and tenants. The significance of such platforms can be understood through the following aspects:

* **Convenience & Accessibility:** Users can search, compare, and book properties anytime and from anywhere without physical visits.
* **Transparency & Trust:** Platforms with verified users and secure systems reduce risks of fraudulent listings.
* **Wider Reach:** Property owners can reach a larger pool of potential buyers or tenants, reducing dependency on brokers.
* **Advanced Search & Filtering:** Digital platforms allow users to filter properties by price, location, type, furnishing status, and amenities.
* **Integration with Financial Services:** Some platforms integrate payment gateways, EMI calculators, and mortgage support, providing end-to-end solutions.

In India and across the globe, platforms like **99acres, MagicBricks and Housing.com** have revolutionized how people buy, sell, or rent properties. However, these platforms still face issues related to affordability, data verification, and scalability. A well-designed solution using modern technologies like **microservices architecture, JWT authentication, and NoSQL databases** can address these gaps effectively.

## **Objective of the Project**

The primary objective of this project is to develop a **Property Renting and Selling Platform** that is **secure, scalable, and user-friendly**, addressing the major shortcomings of existing solutions. Specifically, the project aims to:

1. **Enable Property Owners** to easily list properties with detailed descriptions, pricing, and high-quality images.
2. **Empower Users** to search and filter properties based on various criteria such as price, location, furnishing, category, and property type.
3. **Ensure Security** through JWT-based authentication, preventing unauthorized access.
4. **Facilitate Bookings & Payments** by providing a seamless booking service and integrating payment handling.
5. **Adopt a Microservices Architecture** to ensure modularity, scalability, and ease of maintenance.
6. **Provide Image Management** through Supabase integration, allowing reliable and scalable media storage.

In essence, the project seeks to create a platform that not only simplifies the property renting and selling process but also establishes a secure and reliable digital marketplace.

## **Scope & Improvements**

## **Scope**

* **User Registration & Authentication:** A secure login/registration process using JWT tokens.
* **Property Management:** Owners can create, update, view, and delete property listings.
* **Advanced Property Search:** Users can filter properties by type, location, price, area, and amenities.
* **Booking Service:** Allows users to request property bookings with confirmation.
* **Payment Handling:** A dedicated payment service ensures transactions are processed (currently using mock APIs).
* **Image Uploads:** Supabase integration enables reliable image hosting and retrieval.
* **Scalability:** The microservices architecture ensures that the platform can handle increased traffic and expand with new features.

## **Improvements**

* **Integration with Real Payment Gateways: Add support for live gateways like Razorpay, PayPal, or Stripe for real transactions.**
* **Chat & Communication Module: Enable real-time chat between property owners and seekers for faster negotiations.**
* **Admin Dashboard: Expand admin capabilities for monitoring transactions, managing disputes, and verifying listings.**
* **Map Integration: Use Google Maps or OpenStreetMap APIs to display exact property locations and nearby amenities.**
* **Deployment with Kubernetes: Move from Docker-only deployment to a fully orchestrated Kubernetes cluster for auto-scaling and high availability.**

# **PROJECT ARCHITECTURE**

**Detailed Description of the Architecture**

The Property Renting and Selling Platform follows a **microservices architecture**, ensuring modularity, independent deployment, and easy scalability. Each service communicates via REST APIs and is independently responsible for its own data storage.

Instead of a single monolithic application, the system is divided into multiple microservices:

* **Authentication Service** →Manages login, registration, and JWT token issuance.
* **User Service** → Manages user profiles and related information.
* **Property Service** → Handles property listings, updates, and searches.
* **Booking Service** → Manages bookings for properties.
* **API Gateway** (entry point for clients) → Central entry point for all requests, performs routing and JWT token validation.

All services use **MongoDB** as their primary data store, which is well-suited for handling dynamic, semi-structured property and booking information. MongoDB’s document-based model allows for flexibility in handling complex data structures, while still ensuring scalability.

For media storage (like property images), the platform uses **Supabase Storage**, which provides secure and scalable object storage with easy integration.

The **API Gateway** routes incoming requests to the appropriate microservice, validates JWT tokens for security, and ensures smooth communication between clients and backend services.

This approach ensures:

* **Scalability:** Each service can scale independently.
* **Security:** JWT authentication prevents unauthorized access.
* **Maintainability:** Changes in one service don’t affect others.
* **Flexibility:** MongoDB allows dynamic schema evolution.

**Description of Each Service**

**1. Authentication Service**

**Overview**  
The Authentication Service manages user registration, login, and token-based authentication. It issues and validates **JWT tokens** to secure communication across services.

**Responsibilities**

* Register new users.
* Authenticate login credentials.
* Generate and validate JWT tokens.
* Communicate with the User Service for profile linking.

**2. User Service**

**Overview**  
The User Service manages **user profiles and related data** beyond authentication. It stores preferences, contact details, and account settings.

**Responsibilities**

* Maintain user profiles.
* Store and update user preferences.
* Handle profile updates and deactivation.

**3. Property Service**

**Overview**  
The Property Service manages property listings and supports advanced search and filtering features. It integrates with **Supabase Storage** for property image uploads.

**Responsibilities**

* Allow owners to list, update, and delete properties.
* Provide property search with filters (price, location, type).
* Store and manage property images via Supabase.

**4. Booking Service**

**Overview**  
The Booking Service handles **property bookings**, ensuring availability and preventing conflicts.

**Responsibilities**

* Create and manage property bookings.
* Track booking statuses (Pending, Confirmed, Cancelled).
* Link bookings with users and properties.

### **5. API Gateway Service**

**Overview**  
The API Gateway Service is the **single entry point** for all client requests, responsible for routing, security, and centralized logging.

**Responsibilities**

* Route client requests to the appropriate microservice.
* Validate JWT tokens via the Authentication Service.
* Apply CORS, request filtering, and rate limiting.
* Provide centralized logging and monitoring.

# **SOFTWARE REQUIREMENT SPECIFICATION**

### **1. Introduction**

### **1.1 Purpose**

The purpose of this document is to define the requirements for the Property Renting and Selling Platform. This system enables property owners to list properties for rent or sale, while users can search, book, and make payments for properties. The system ensures secure authentication, reliable data management, and scalable architecture using microservices.

**1.2 Scope**

The platform provides:

* Property listing and management
* User registration and profile management
* Secure authentication using JWT
* Booking system for properties
* Payment handling with transaction records
* Image upload and storage using Supabase

The system is web-based and accessible via browsers. It follows a **microservices architecture** for scalability and maintainability.

### **1.3 Definitions, Acronyms, and Abbreviations**

* **JWT**: JSON Web Token
* **API**: Application Programming Interface
* **CRUD**: Create, Read, Update, Delete
* **ERD**: Entity Relationship Diagram
* **Supabase**: A backend-as-a-service platform providing authentication, storage, and databases

### **2. Overall Description**

### **2.1 Product Perspective**

The platform is a standalone web application built using **React (frontend)**, **Spring Boot microservices (backend)**, **MongoDB (database)**, and **Supabase (media storage)**. The **API Gateway** acts as a single entry point for all requests.

### **2.2 Product Features**

* **Authentication Service**: User registration, login, and JWT validation
* **User Service**: Profile management and user preferences
* **Property Service**: Listing, updating, deleting, and searching properties
* **Booking Service**: Property booking and cancellation
* **Payment Service**: Payment processing and transaction history
* **API Gateway**: Request routing and centralized security

### **2.3 User Classes and Characteristics**

* **Property Owner**: Lists and manages properties
* **Customer**: Searches for properties, makes bookings, and payments
* **Admin (optional for future)**: Monitors and manages users, properties, and transactions

### **2.4 Operating Environment**

* **Frontend**: Modern browsers (Chrome, Firefox, Edge)
* **Backend**: Spring Boot microservices deployed on a server or cloud environment
* **Database**: MongoDB 6.0+
* **Storage**: Supabase Storage
* **API Gateway**: Spring Cloud Gateway

### **2.5 Design and Implementation Constraints**

* System must use **JWT-based authentication**.
* All services must use **MongoDB as database**.
* Images must be stored in **Supabase Storage**.

### **2.6 Assumptions and Dependencies**

* Users have stable internet access.
* Supabase and MongoDB services are up and running.
* The system may integrate with third-party payment gateways in the future.

### **3. Functional Requirements**

1. **Authentication**
   * Users must register and log in securely.
   * JWT tokens must be generated and validated for all requests.
2. **User Management**
   * Users can update their profile and preferences.
   * Users can deactivate their accounts.
3. **Property Management**
   * Owners can create, update, and delete property listings.
   * Users can search/filter properties.
4. **Booking Management**
   * Users can book available properties.
   * Users can cancel bookings.
5. **Payment Management**
   * Payments must be recorded securely.
   * Transaction history must be retrievable.
6. **Media Storage**
   * Property images must be uploaded to Supabase Storage.

## **4. Non-Functional Requirements**

* **Performance**: System should respond within 3 seconds for most operations.
* **Scalability**: Services should scale independently.
* **Security**: All requests must be authenticated using JWT.
* **Availability**: 99.5% uptime expected.
* **Maintainability**: Microservices can be updated without downtime.

# **Software Requirements**

1. **Backend**
   * Java 17 or higher
   * Spring Boot 3.x
   * Spring Cloud Gateway
   * Spring Security (JWT)
   * MongoDB Java Driver
   * Maven
2. **Frontend**
   * React.js (with Vite bundler)
   * Axios (for API calls)
   * Material UI
3. **Database**
   * MongoDB 6.0+
4. **Storage**
   * Supabase Storage account
5. **Development Tools**
   * Visual Studio Code
   * IntelliJ IDEA / Spring Tool Suite 4
   * Postman (for API testing)
   * Git & GitHub (for version control)

# **Hardware Requirements**

### **For Development Environment**

* **Processor**: Intel i5 (or equivalent)
* **RAM**: 8 GB minimum
* **Storage**: 256 GB SSD
* **OS**: Windows 10 / Linux Ubuntu 22.04 / macOS Monterey

### **For Deployment Server**

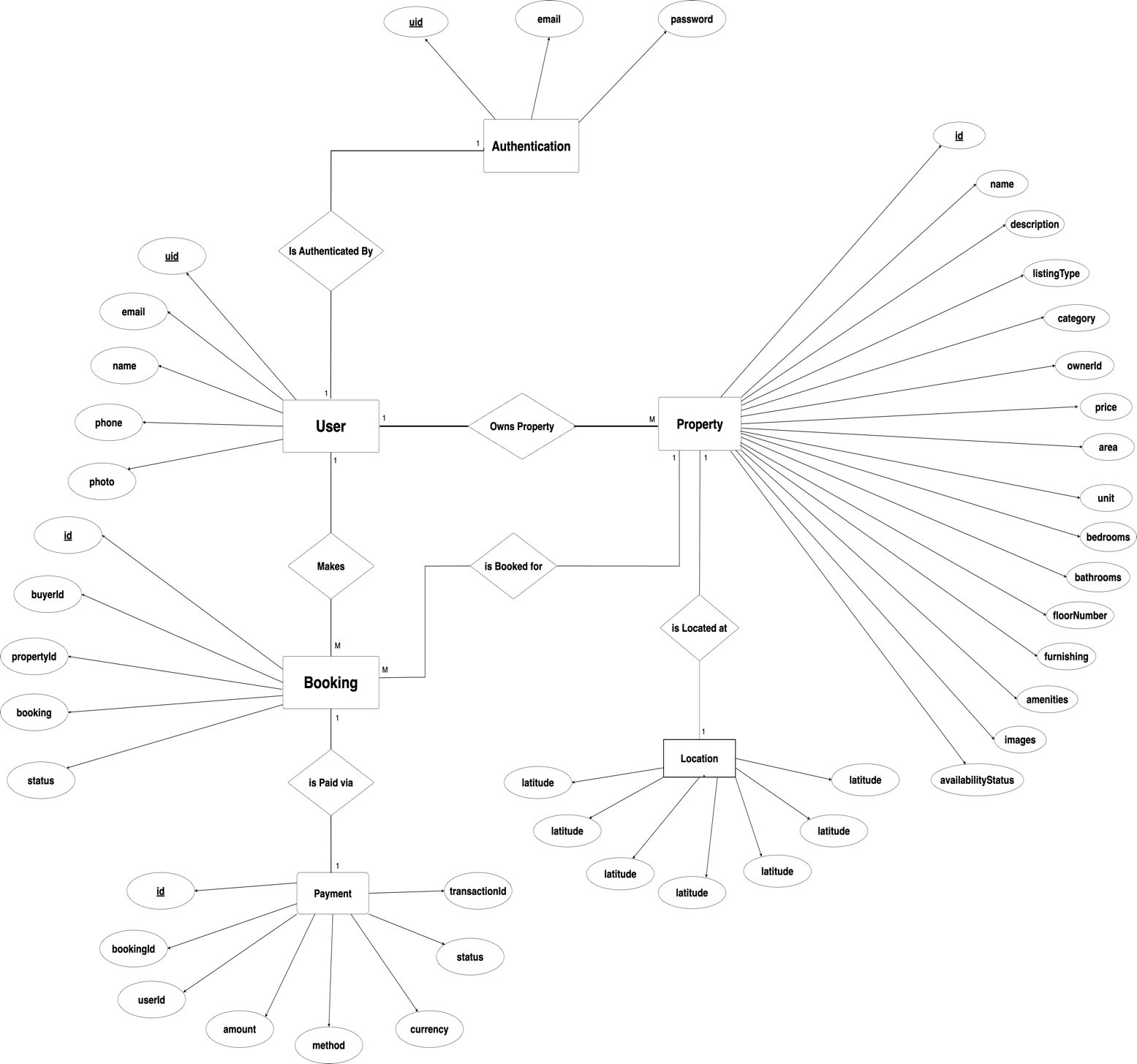
* **Processor**: Intel Xeon or AMD EPYC (Quad-core)
* **RAM**: 16 GB minimum
* **Storage**: 500 GB SSD
* **Bandwidth**: High-speed internet (100 Mbps+)
* **Scalability**: Cloud environment (AWS / Azure / GCP) recommended

### **For Client System**

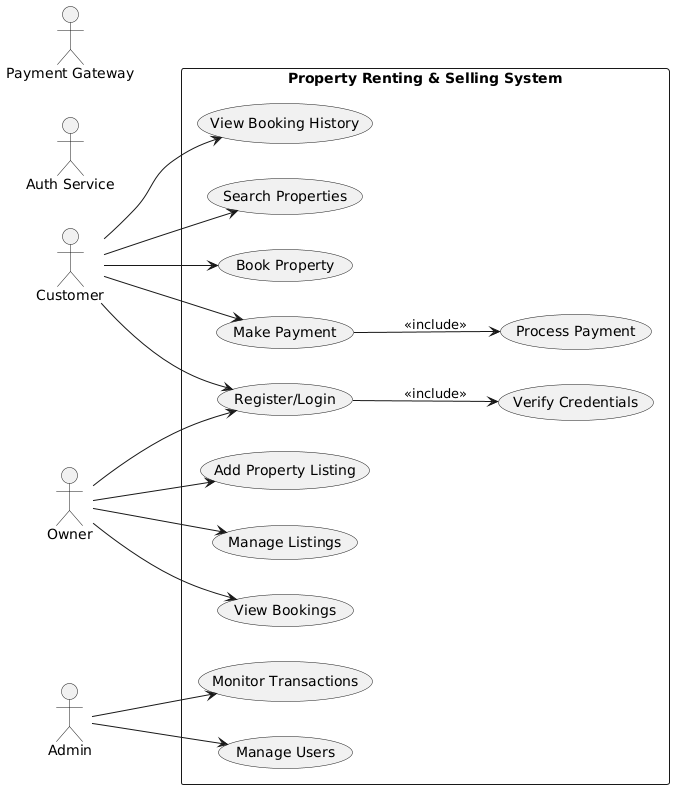
* **Processor**: Intel i3 or higher
* **RAM**: 4 GB minimum
* **Browser**: Latest Chrome / Firefox / Edge
* **Internet**: 2 Mbps minimum

# **DIAGRAMS**

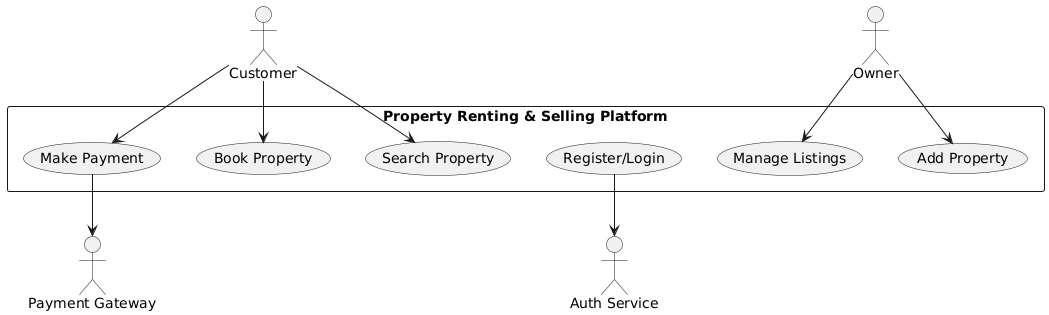
### **Entity Relationship Diagram:**



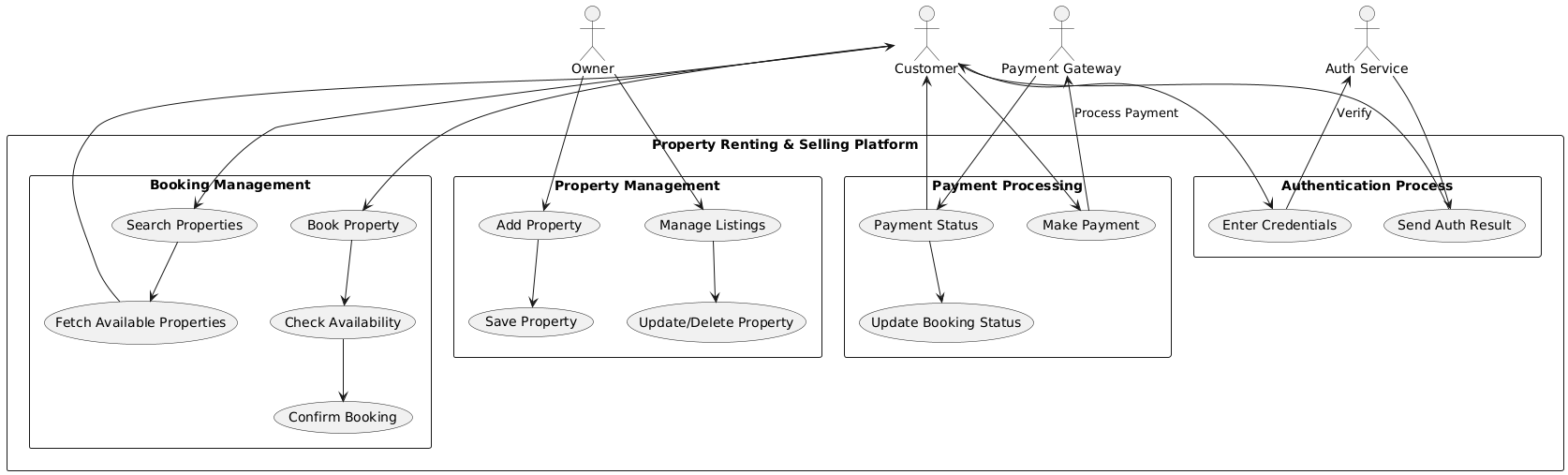
1. **Use Case Diagram**



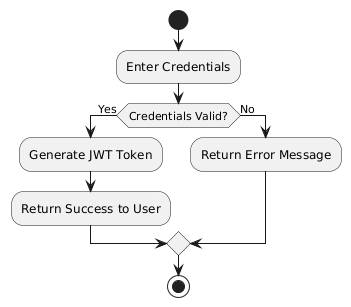
1. **Data Flow Diagram**
   1. **DFD Level 0**



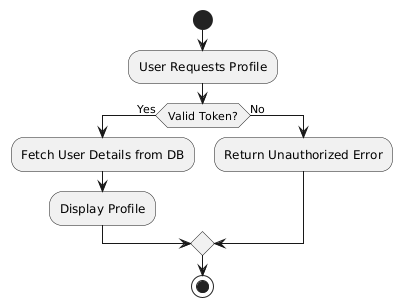
* 1. **DFD Level 1**



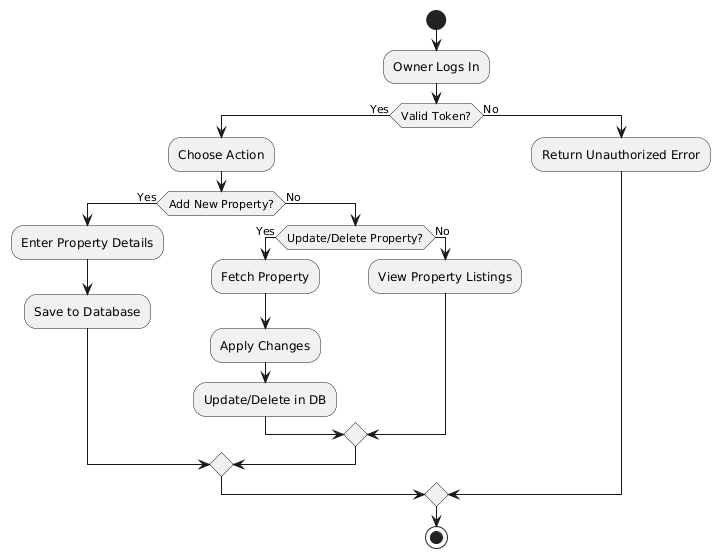
1. **Activity Diagram**
   1. **Authentication Service Activity Diagram**



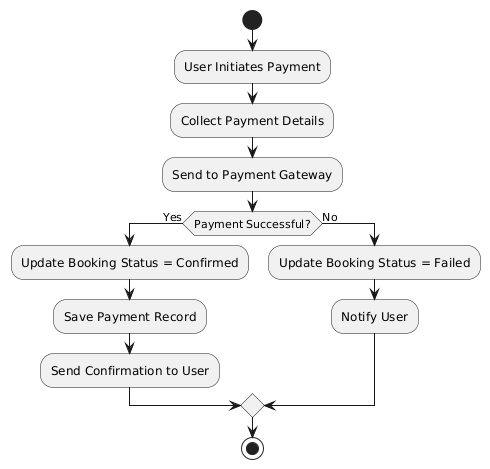
* 1. **User Service Activity Diagram**



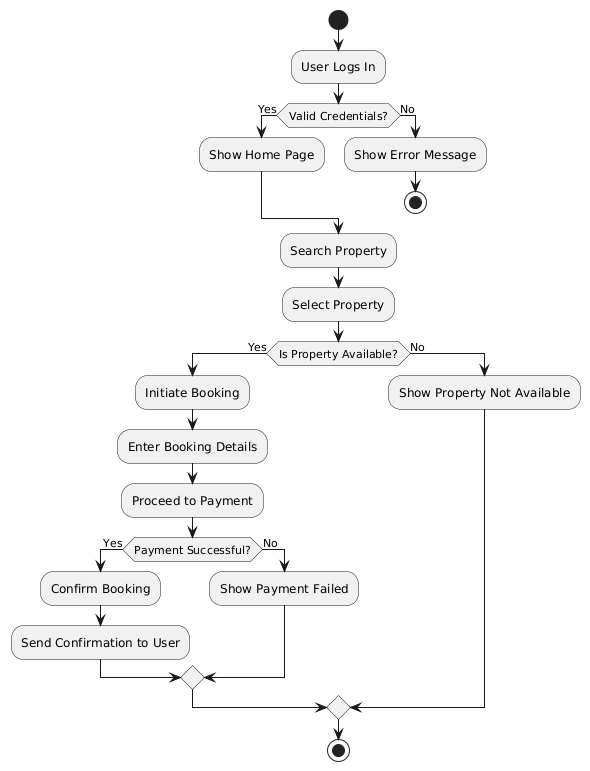
* 1. **Property Service Activity Diagram**



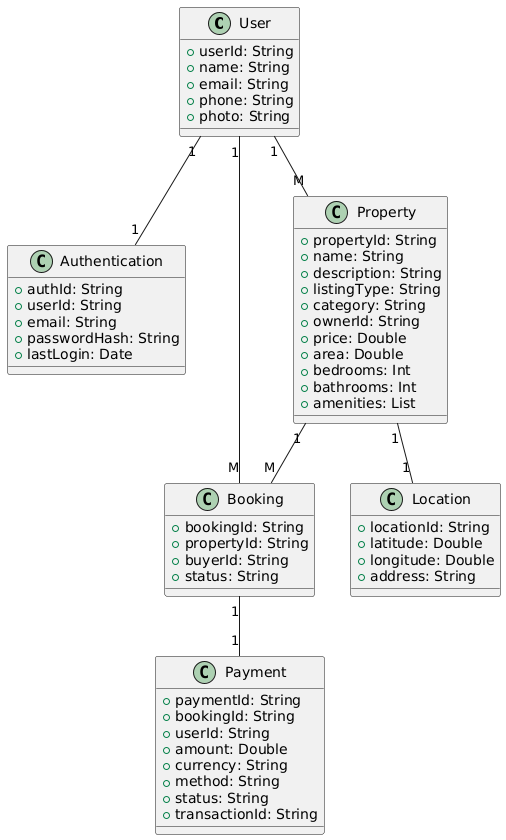
* 1. **Payment Service Activity Diagram**



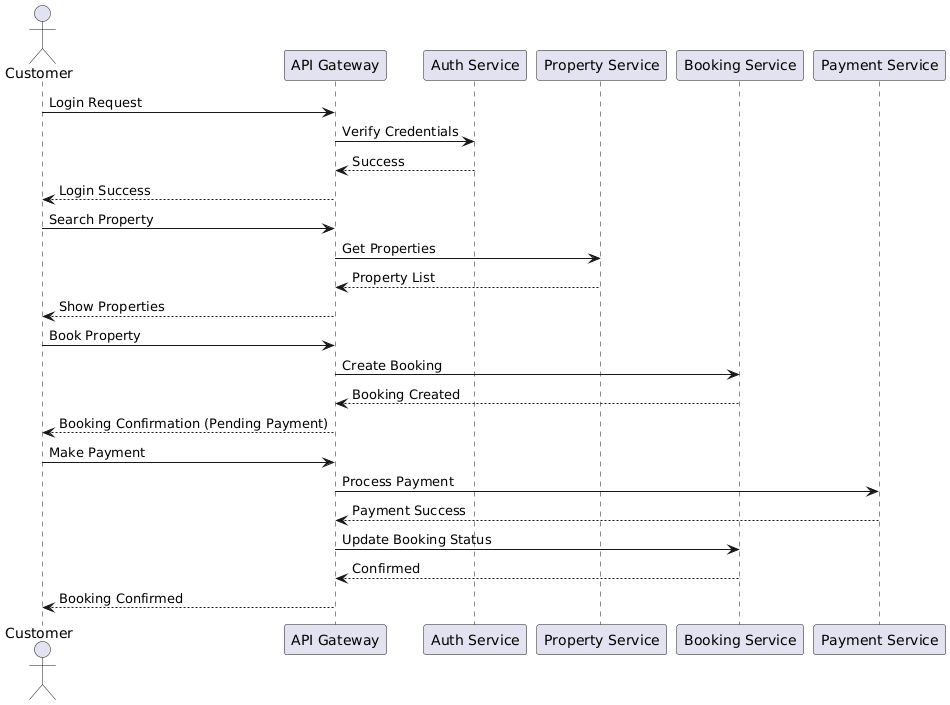
* 1. **Booking Service Activity Diagram**



1. **Class Diagram**

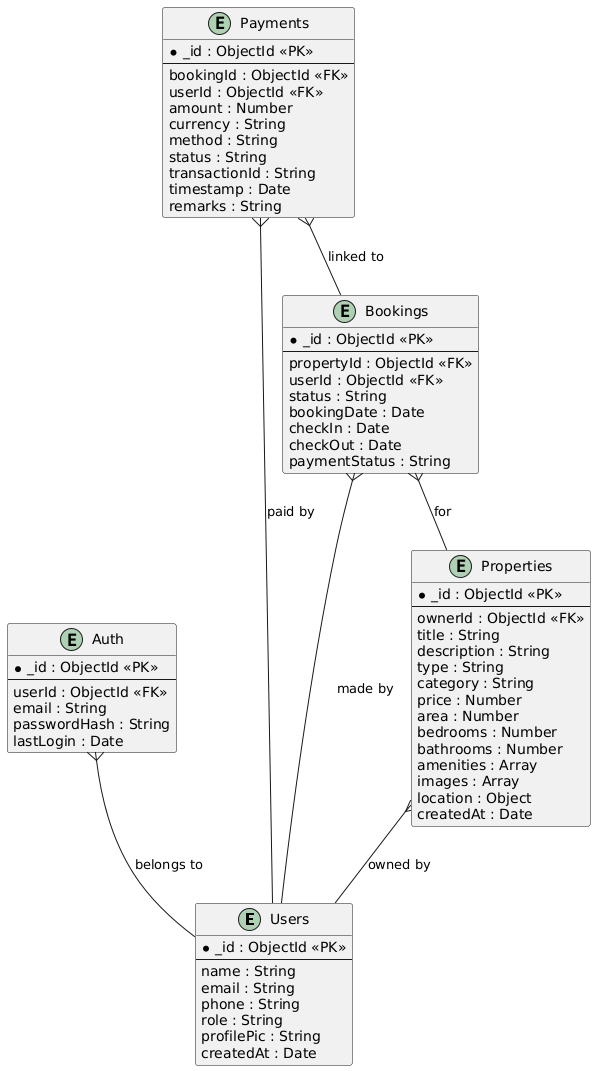


1. **Sequence Diagram**



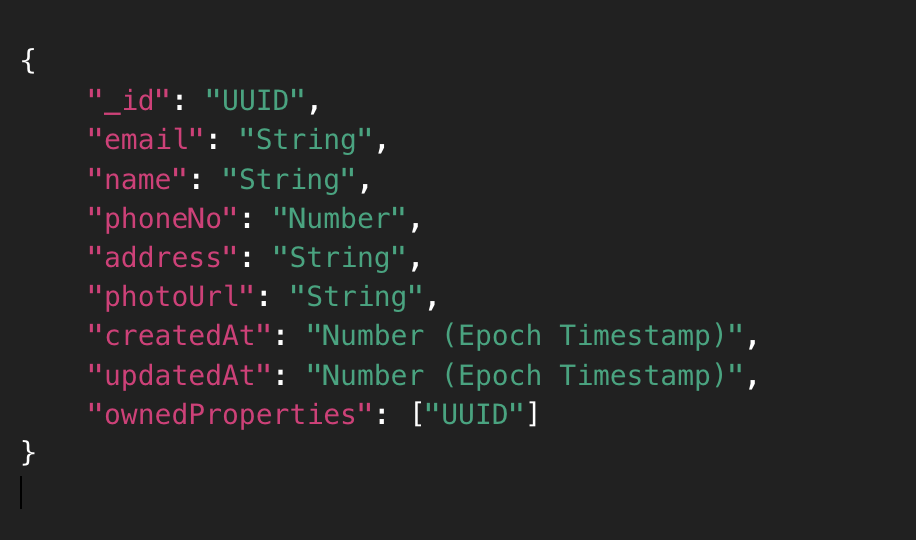
# **DATABASE DESIGN**

**Design**



**Collections**

* 1. **Users Collection**

****

* 1. **Authentication Collection**

**2.1 User-Credentials Collection**

****

**2.2 User-Token Collection**

****

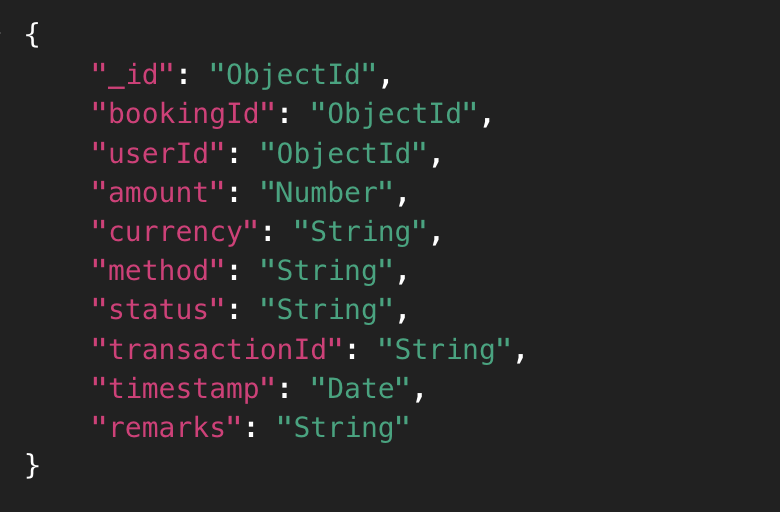
* 1. **Properties Collection**

****

* 1. **Bookings Collection**

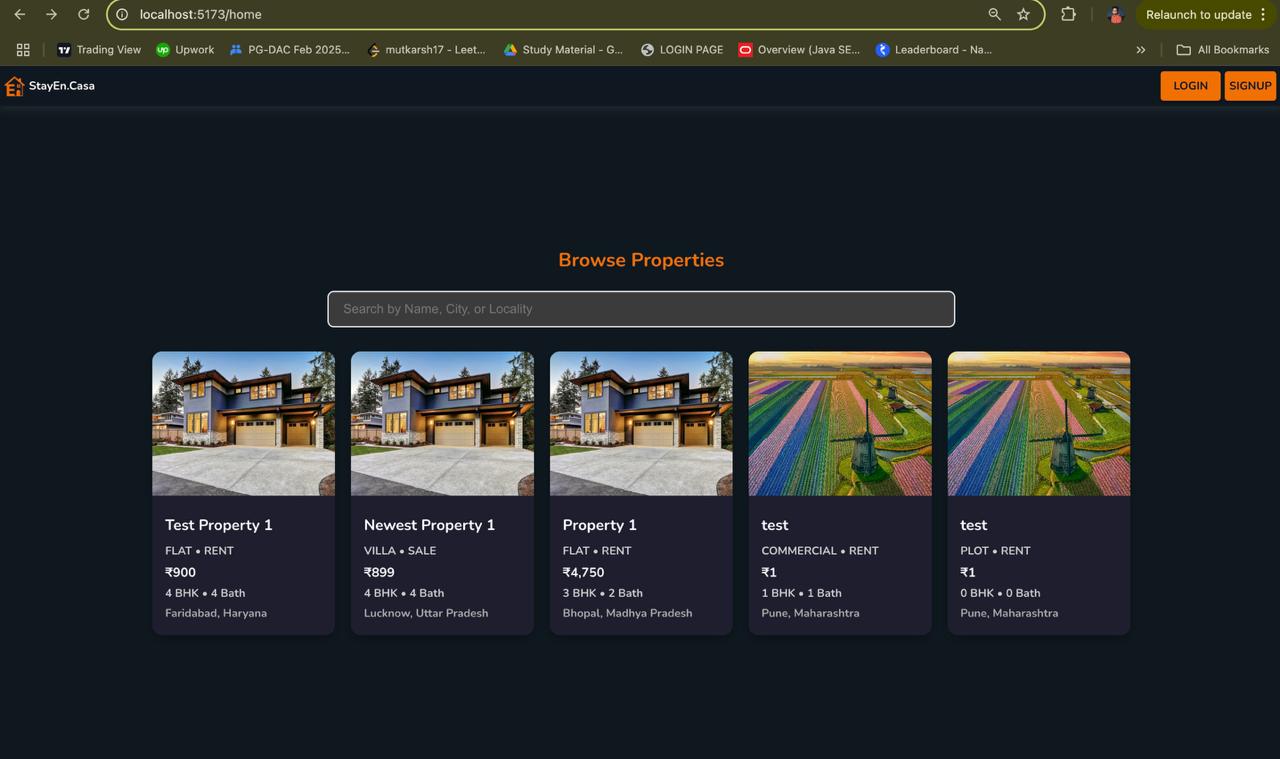
****

* 1. **Payments Collection**

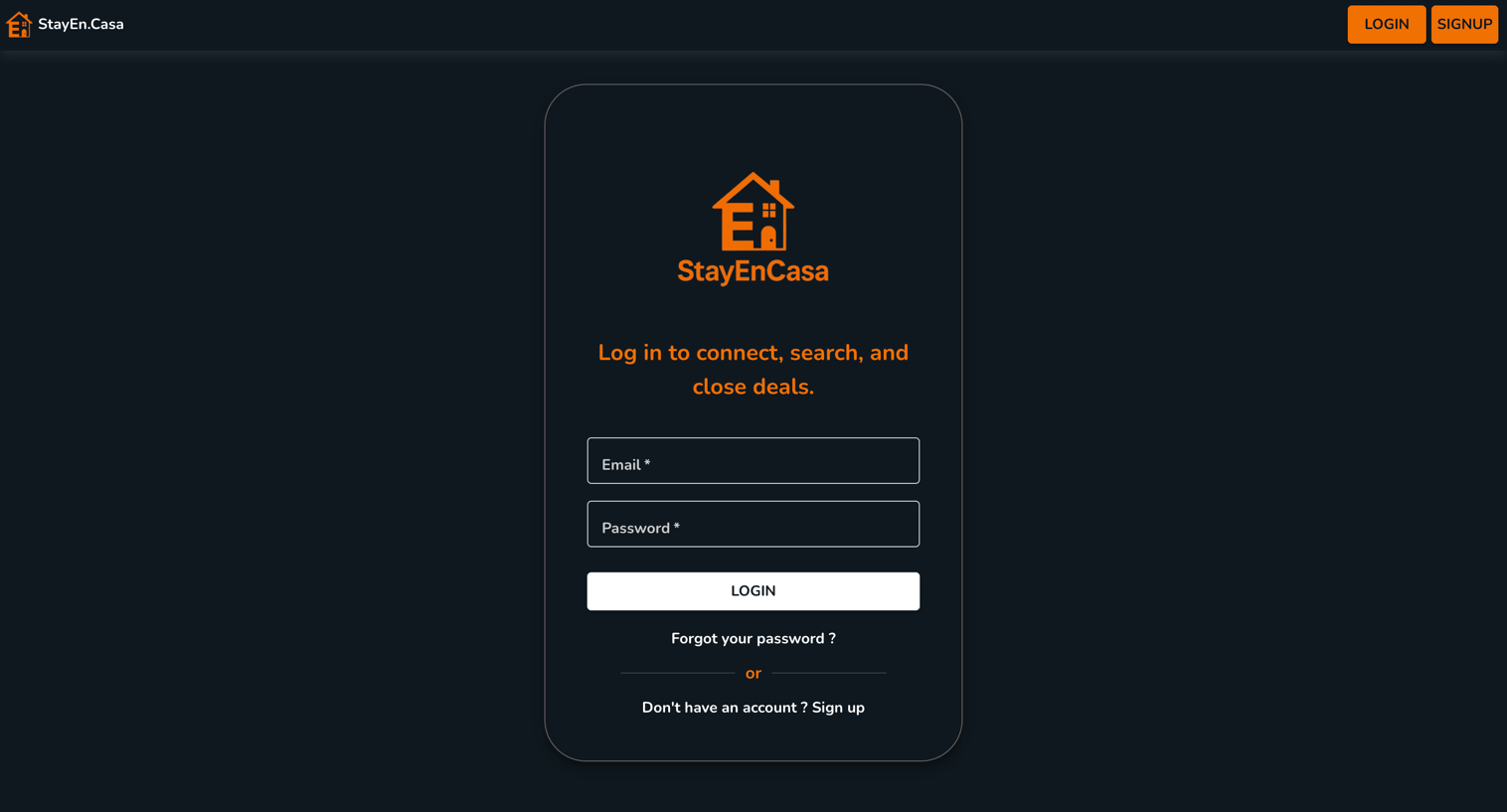
****

# **SNAPSHOTS**

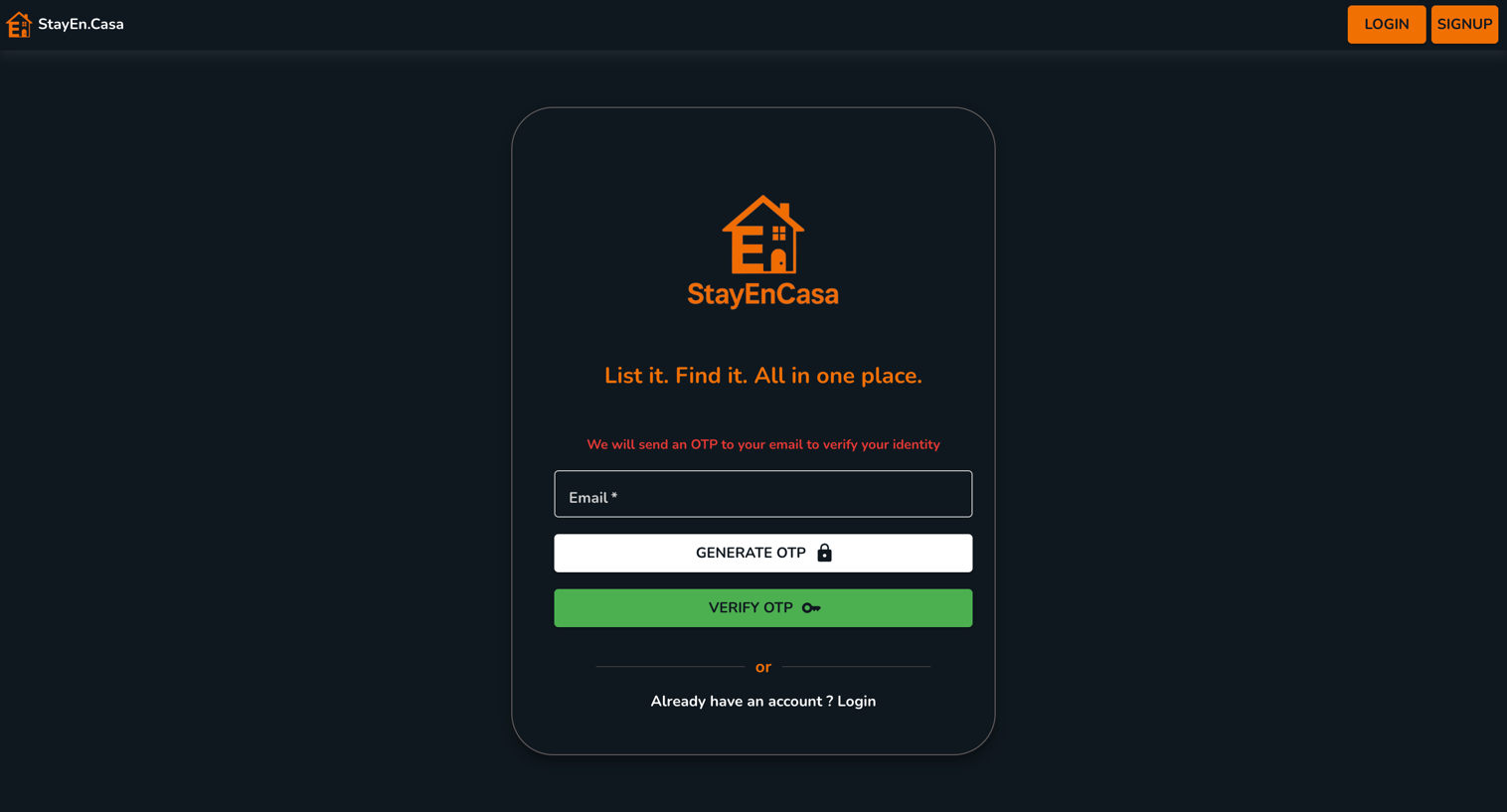
**Home Page**

****

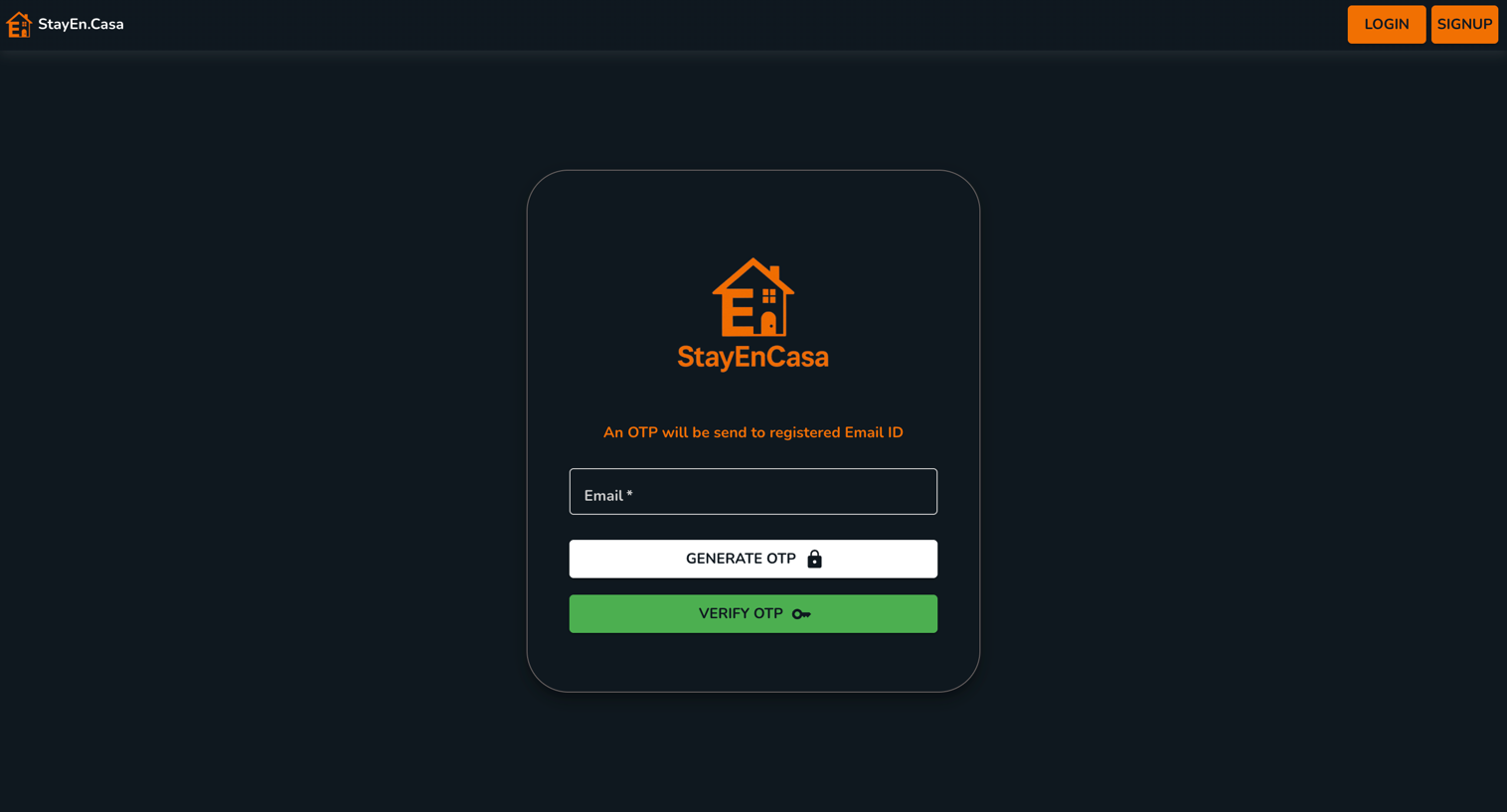
**Login Page**

****

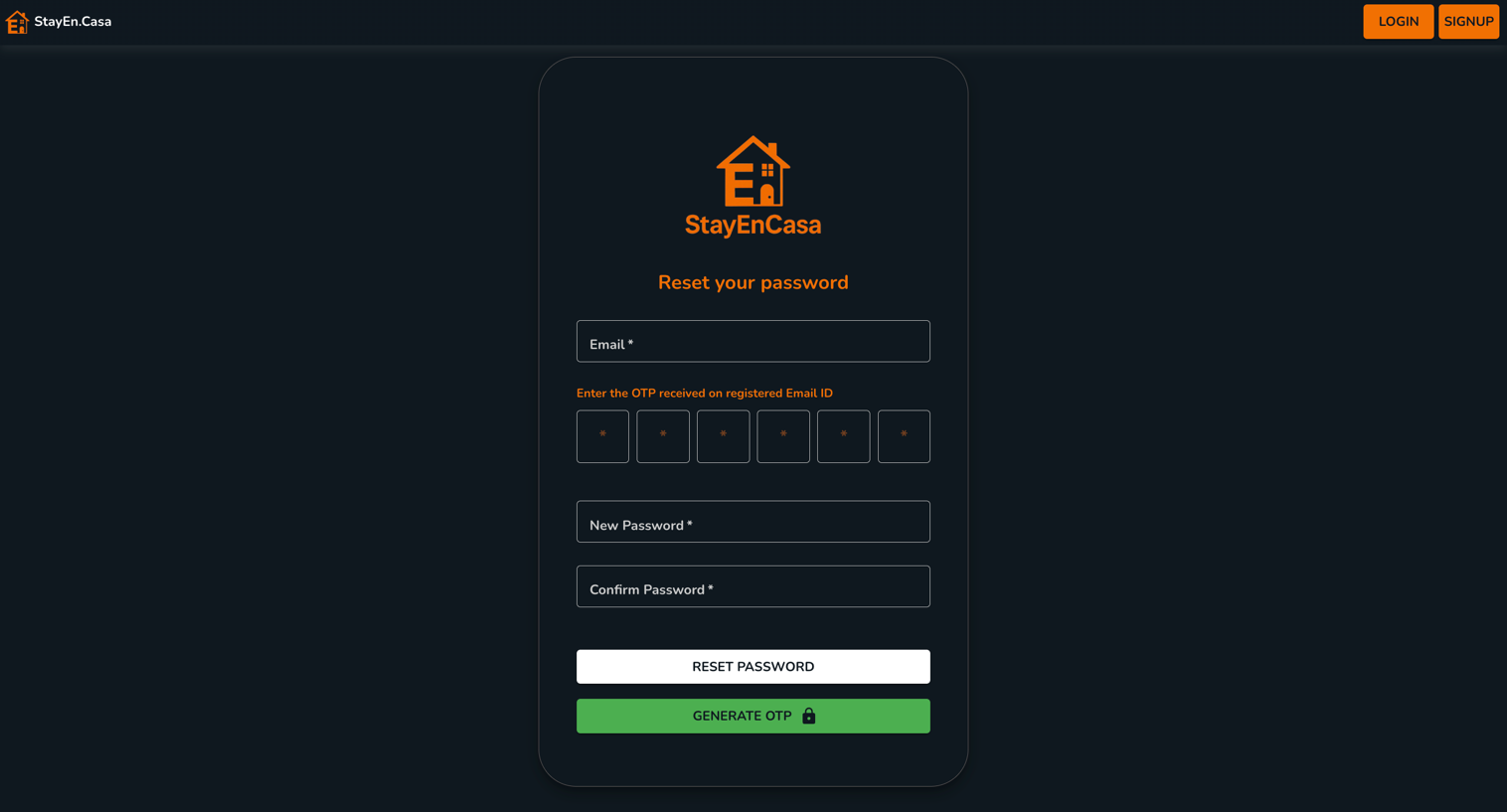
**Signup Page**



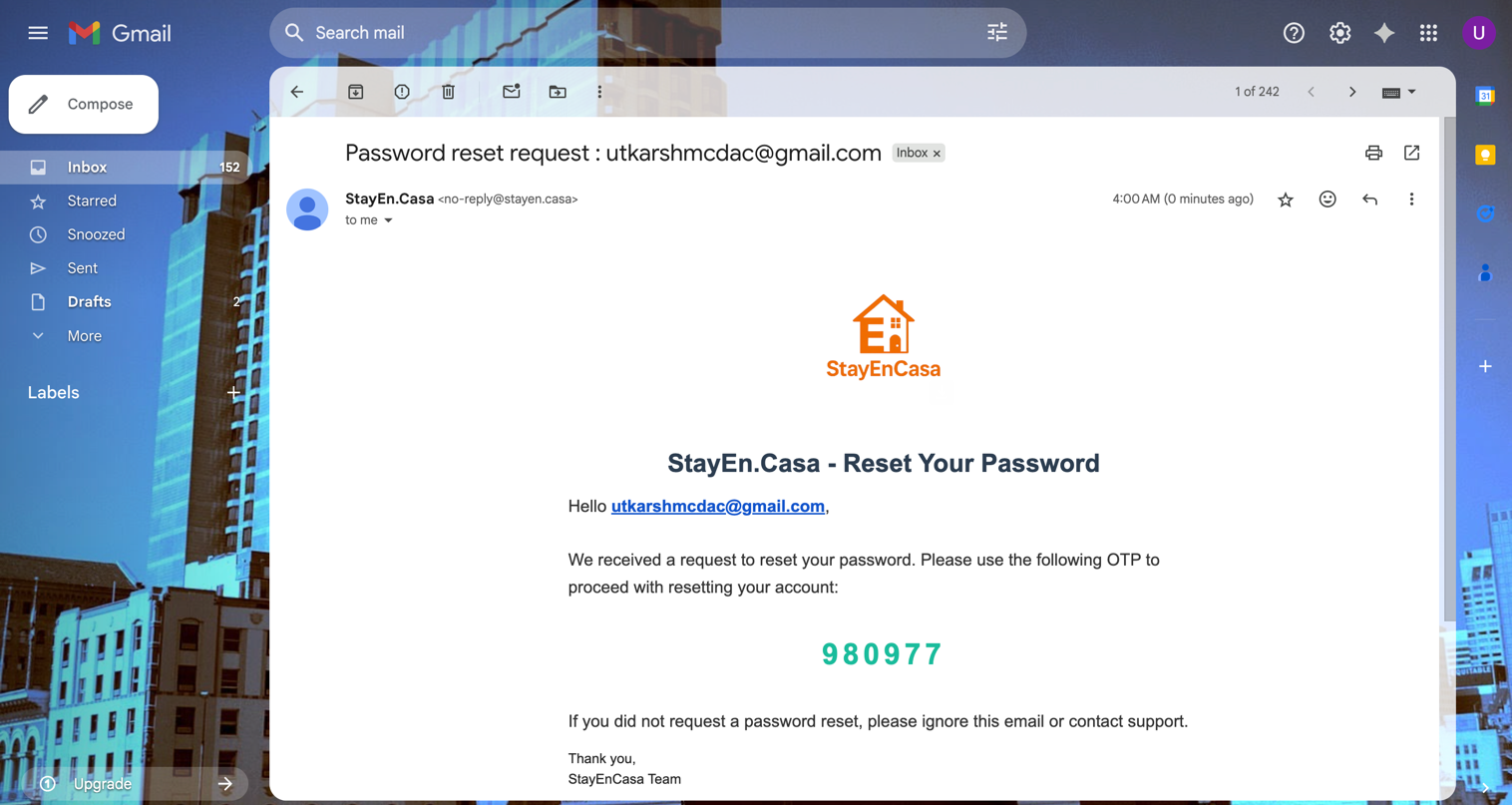
**Forgot Password Page**

****

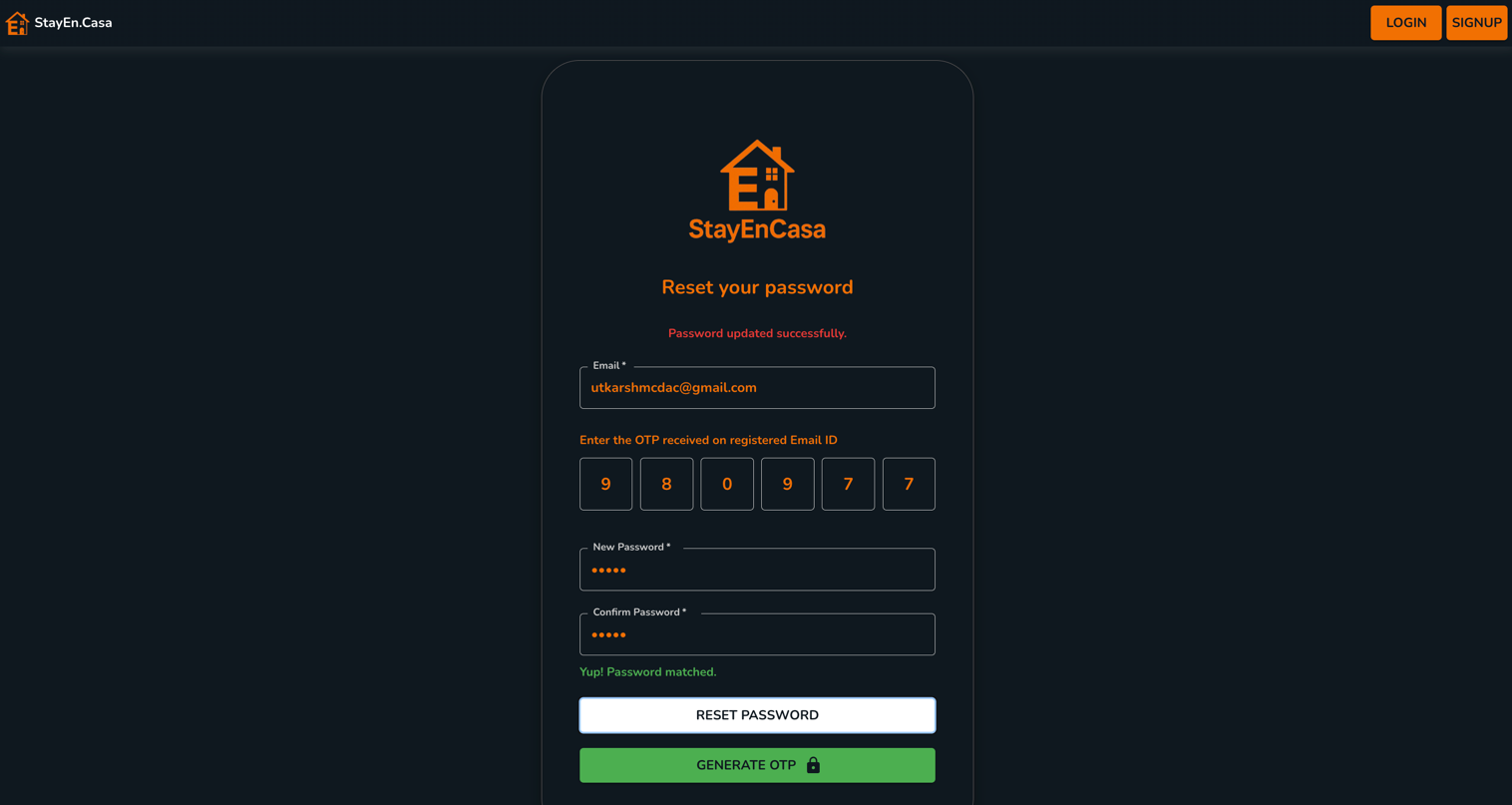
**Change Password Page**

****

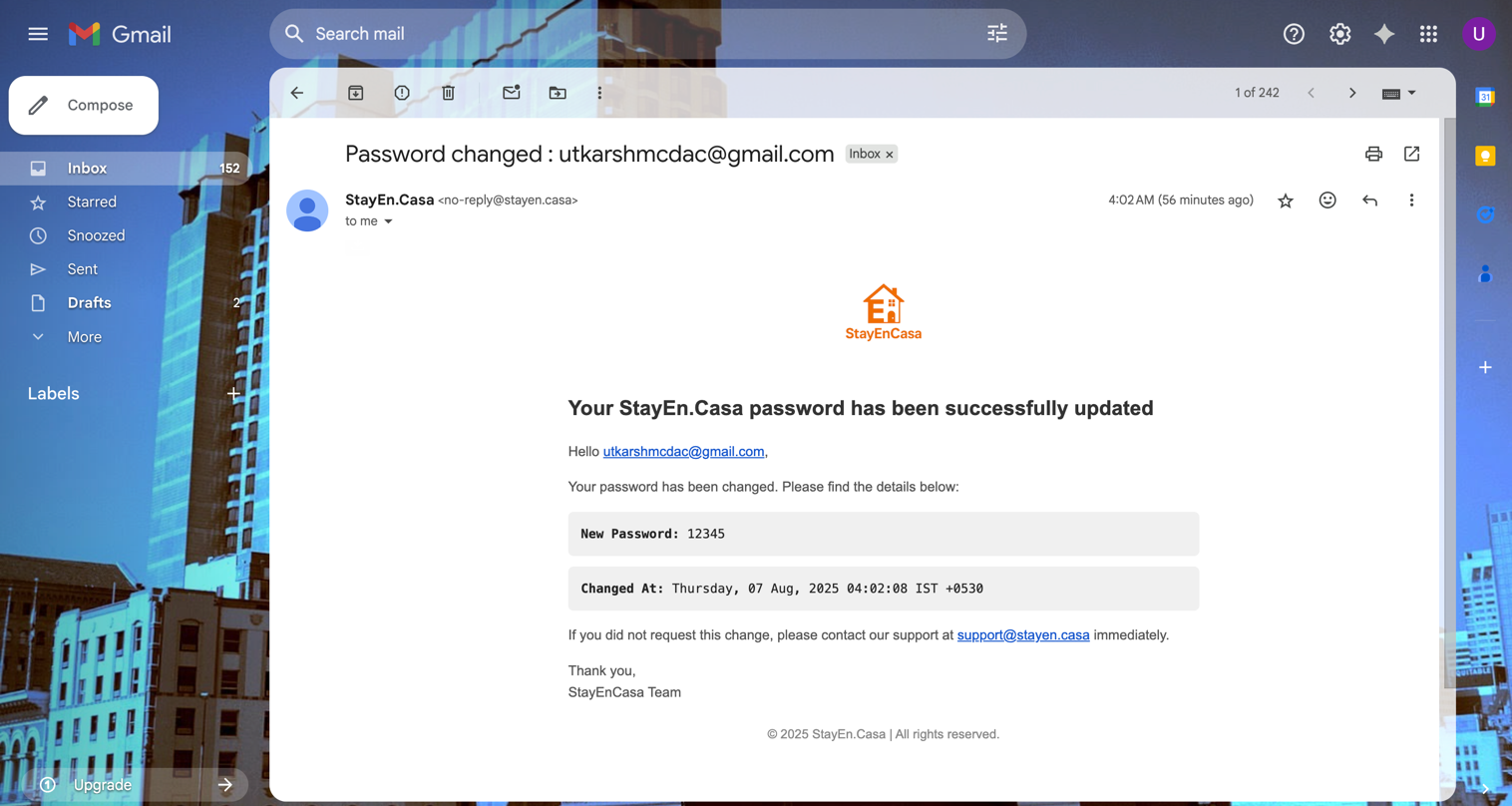
**OTP to Reset Password**

****

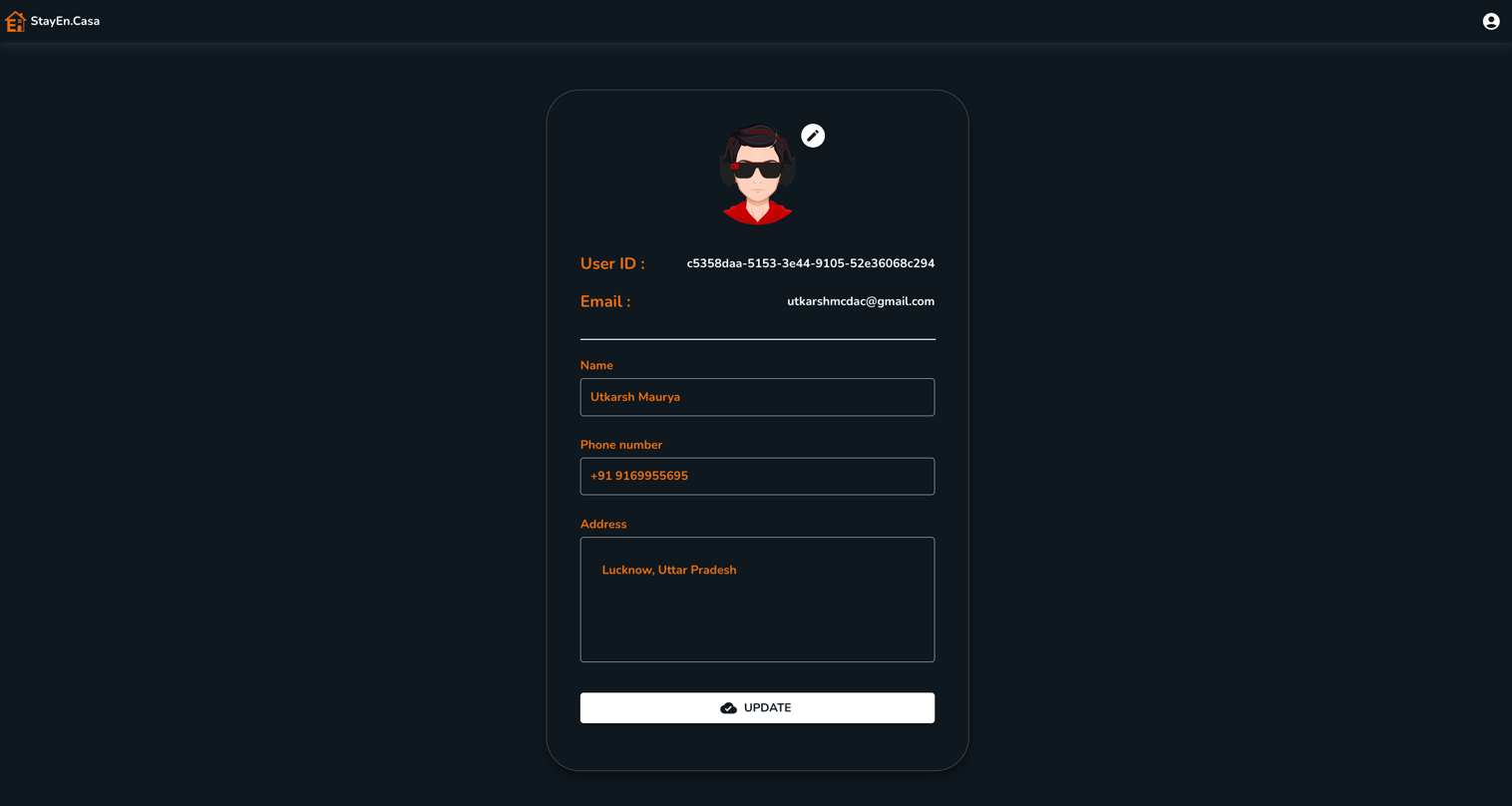
**Password Reset Successfully Page**

****

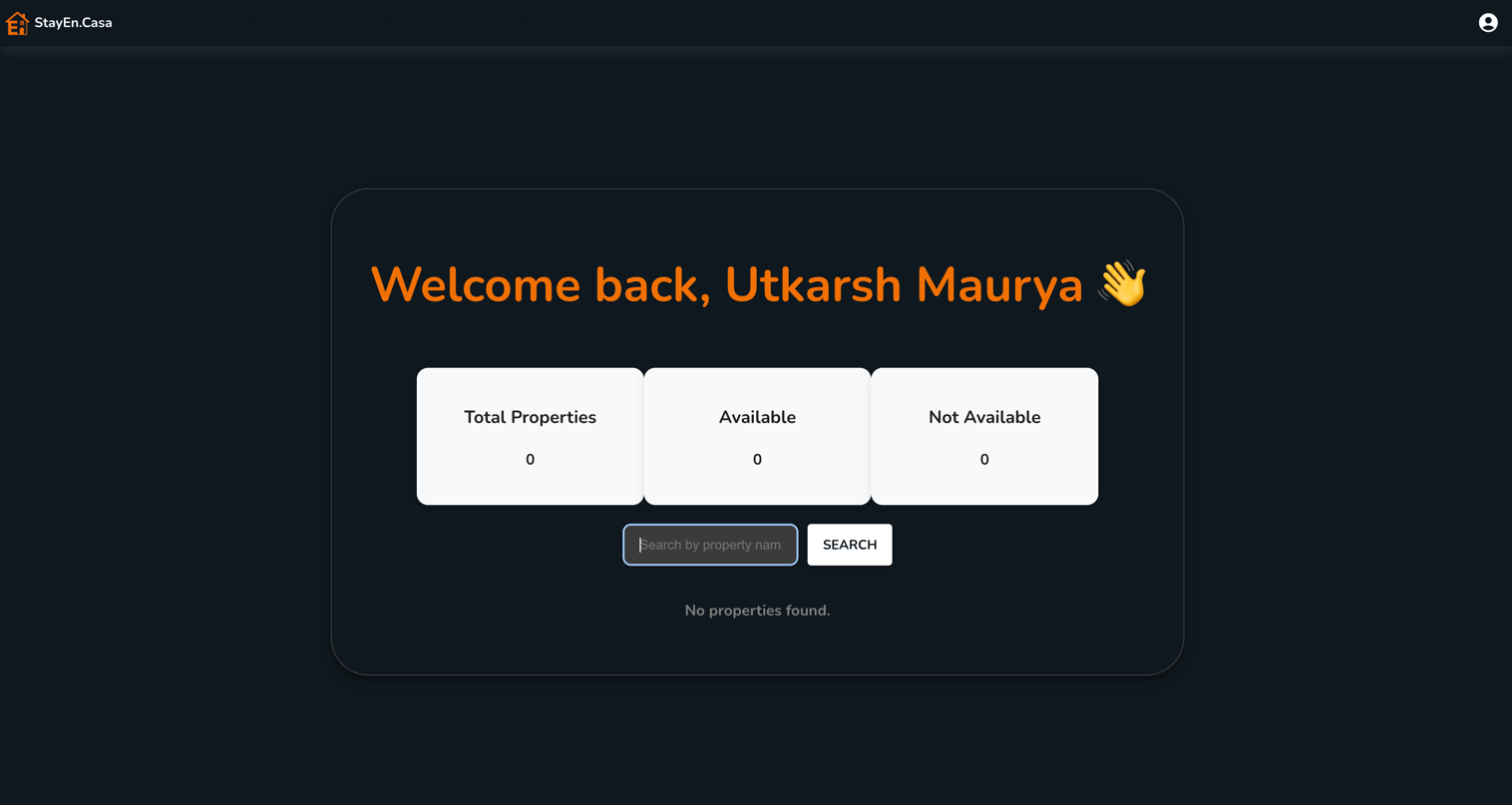
**Password Change Successfully Mail**

****

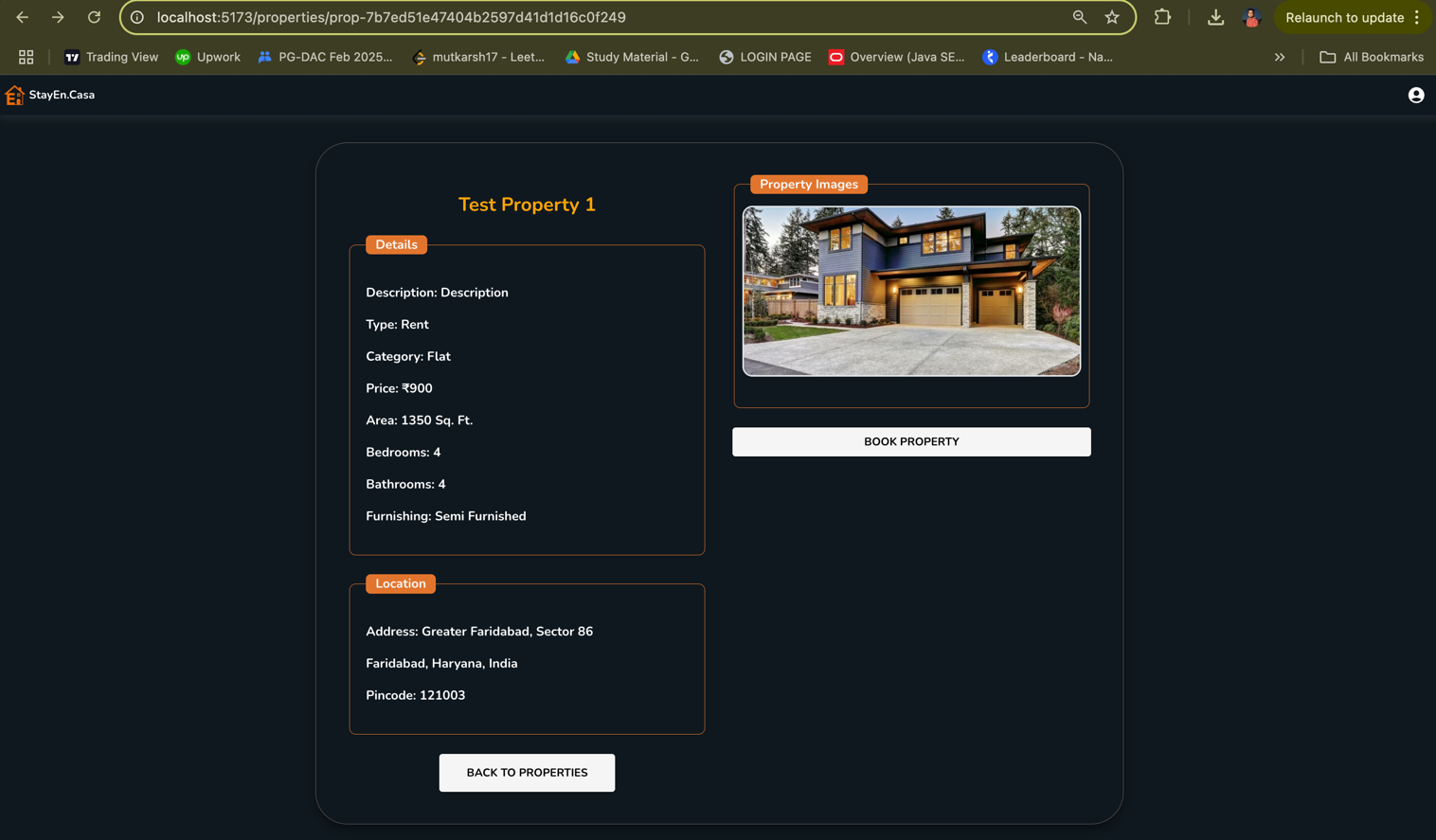
**User Profile Page**

****

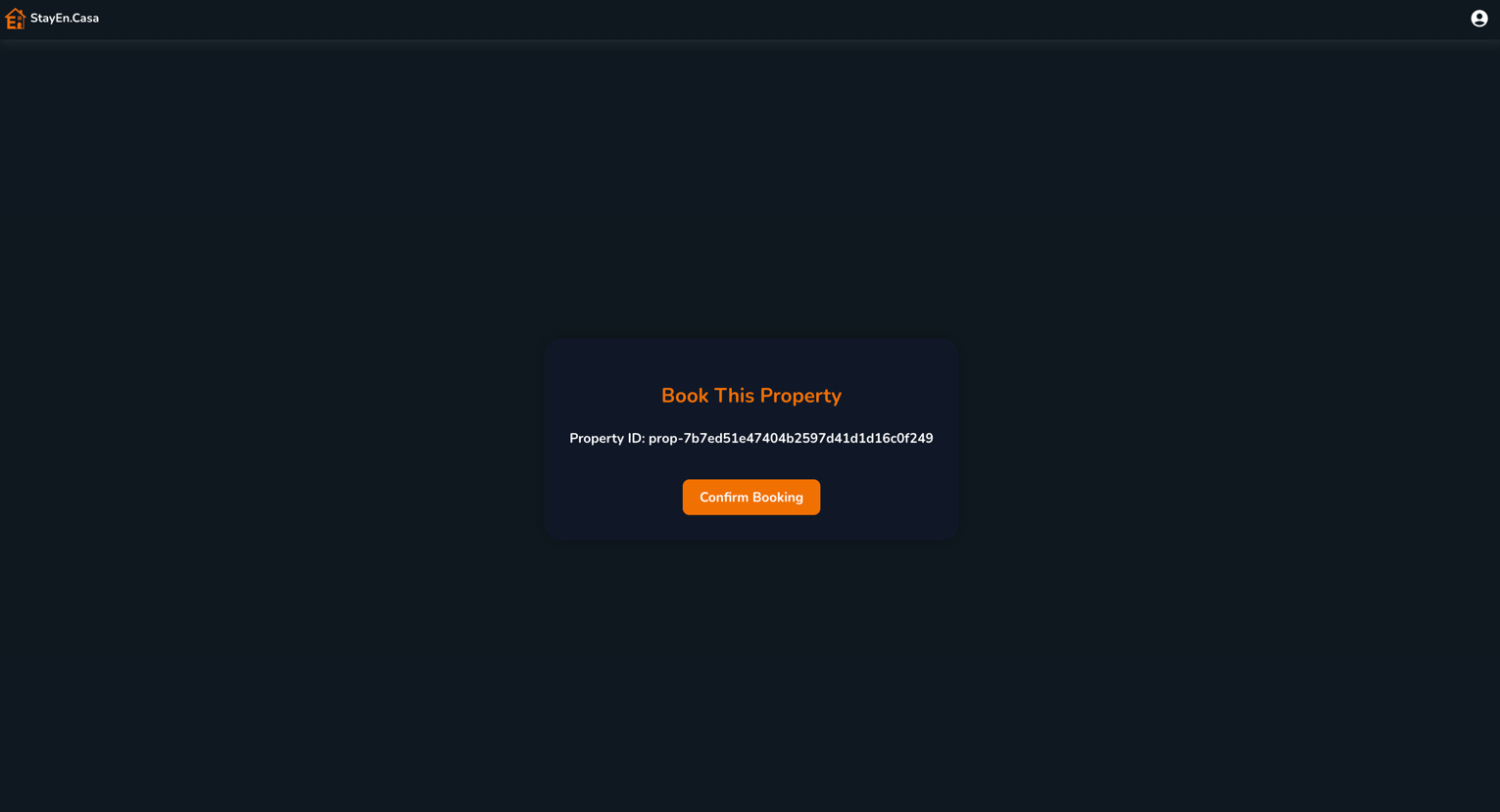
**Dashboard Page**

****

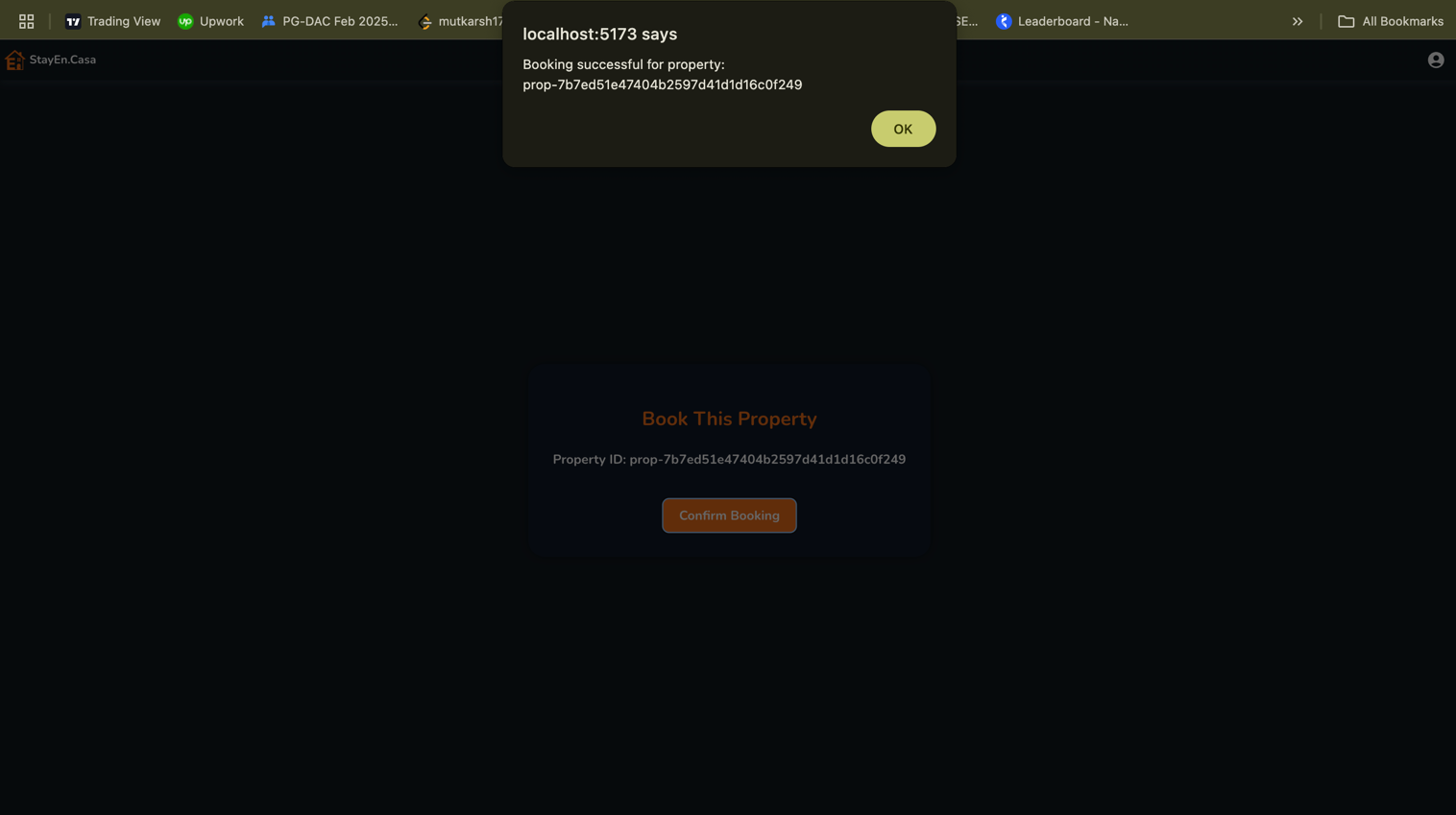
**Property Details Page**

****

**Booking Page**

****

**Booking Successful Page**

****

# **CONCLUSION**

The real-estate management platform developed through this project successfully addresses the key challenges faced by property owners, tenants, and administrators in the digital era. By leveraging a microservices architecture with dedicated services for **User Management, Authentication, Property Listings, Bookings, and Payments**, the system achieves modularity, scalability, and maintainability.

Using **MongoDB** as the database for all services ensures flexibility in handling heterogeneous property data and user interactions. Integration with **Supabase for media storage** and **JWT-based authentication** strengthens both usability and security. The inclusion of features like user-owned properties, saved properties, and token-based multi-device authentication further enhance user experience and system reliability.

The platform demonstrates a smooth workflow, from property registration to booking and payment processing, while maintaining data integrity and security. Moreover, the architecture leaves scope for future improvements, such as:

* Adding AI-driven property recommendations,
* Implementing real-time notifications for bookings and payments, and
* Enhancing analytics dashboards for property trends.

In conclusion, the project provides a **robust, secure, and scalable solution** for modern real-estate needs, aligning technology with practical requirements. It not only showcases technical proficiency in **Spring Boot, MongoDB, React, and Microservices architecture** but also lays a strong foundation for future enhancements.

# **REFERENCES**

1. MongoDB Documentation. Data Models and Schema Design.  
   <https://www.mongodb.com/docs/manual/core/data-model-design>
2. Spring Boot Documentation. Building Microservices with Spring Boot.  
   <https://docs.spring.io/spring-boot/index.html>
3. React Official Documentation. Getting Started with React.  
   <https://react.dev/learn>
4. Supabase Documentation. Authentication and Storage.  
   <https://supabase.com/docs>
5. Fowler, M. (2014). Microservices Resource Guide.  
   <https://martinfowler.com/microservices>
6. Baeldung. Spring Security and JWT.  
   <https://www.baeldung.com/spring-security-oauth-jwt>
7. MongoDB University. M001: MongoDB Basics.  
   <https://learn.mongodb.com>
8. DigitalOcean Community. Deploying Spring Boot with MongoDB.  
   <https://www.digitalocean.com/community/tutorials>