SmartRealEstateManagementSystem

Project Documentation

---

1. Introduction

SmartRealEstateManagementSystem is a full-stack web application for managing real estate properties, users, payments, and inquiries. The backend is built with ASP.NET Core (C#), using a modular architecture with Application, Domain, Infrastructure, and Identity layers. The frontend is implemented in Angular.

---

2. System Architecture

- **Backend:** ASP.NET Core Web API, layered into Application, Domain, Infrastructure, and Identity projects.
- **Frontend:** Angular SPA (in `smart-real-estate-management-system/`).
- **Database:** PostgreSQL (main), SQLite (for identity and testing).
- **Testing:** xUnit for backend, Jasmine/Karma for frontend.

---

3. Modules Overview

3.1. Backend Modules

- **Application Layer:**
  Contains business logic, command/query handlers (CQRS), DTOs, and ML integration (property price prediction).
    - Subfolders: `AIML/`, `Authentication/`, `CommandHandlers/`, `Commands/`, `Contracts/`, `DTOs/`, `Interfaces/`, `Queries/`, `QueryHandlers/`, `QueryReponses/`, `Utils/`.
- **Domain Layer:**
  Defines core entities, value objects, enums, repositories, and domain logic.

- Entities: `Property`, `UserInformation`, `Payment`, `Inquiry`, `PropertyImage`.
- Types: `PropertyType`, `UserRole`, `PaymentType`, etc.

- **Infrastructure Layer:**
  Implements data access (EF Core), repositories, services, and database context.
  - Contains `Persistence/`, `Repositories/`, `Services/`, `Migrations/`, `Filters/`.

- **Identity Layer:**
  Handles user authentication, authorization, and identity management.
  - Contains `UsersDbContext`, user repositories, and migrations.

- **Web API (SmartRealEstateManagementSystem/):**
  Exposes REST endpoints via controllers:
  - `PropertiesController`
  - `UserInformationController`
  - `PaymentsController`
  - `InquiriesController`

3.2. Frontend Modules

- **Angular Application:**
  - Located in `smart-real-estate-management-system/`.
  - Organized into `src/app/components/` (e.g., property-list, property-detail), `src/app/pages/` (e.g., profile-page), and `src/app/services/` (e.g., property.service.ts).
  - Uses Angular routing, forms, and HTTP client for API integration.
  - Implements authentication, property management, user profile, and inquiry features.

---

4. Database Design

- **Entities and Relationships:**
  - `UserInformation`: Stores user data (username, email, role, etc.).
  - `Property`: Real estate properties, linked to users.
  - `Payment`: Payment records, linked to properties and users (buyer/seller).
  - `Inquiry`: Inquiries about properties, linked to properties, agents, and clients.

- `PropertyImage` : Images for properties (one-to-many with Property).

- **Relationships:**
  - One-to-many: UserInformation → Properties, UserInformation → Inquiries, Property → PropertyImages.
  - Many-to-one: Payment → Property, Payment → Buyer/Seller, Inquiry → Property/Agent/Client.

---

5. Key Features

- **User Management:**
  - Registration, login, JWT authentication.
  - User roles: CLIENT, PROFESSIONAL, ADMIN.
  - Profile management.
- **Property Management:**
  - CRUD operations for properties.
  - Property images upload and management.
  - Filtering and searching properties.
- **Payments:**
  - Record and manage payments for property transactions.
  - Payment types: SALE, RENT, etc.
- **Inquiries:**
  - Users can send inquiries about properties.
  - Agents and clients can manage inquiries.
- **Machine Learning Integration:**
  - Property price prediction using ML.NET model (`model.zip` in Application/AIML/).
- **Security:**
  - JWT-based authentication.
  - Role-based authorization in controllers.

---

6. Testing

- **Backend:**
  - Unit tests: Located in `SmartRealEstateManagementSystem.Application.UnitTests/`.
    - Test command/query handlers, mapping profiles, repositories, and business logic.
  - Integration tests: Located in `SmartRealEstateManagementSystem.IntegrationTests/`.
    - Test API endpoints and database interactions using in-memory databases.
- **Frontend:**
  - Unit tests: Jasmine/Karma, in `*.spec.ts` files under `src/app/`.
  - End-to-end tests: (Setup available, framework can be chosen as needed).

---

7. Extensibility & Maintenance

- **CQRS pattern** for clear separation of commands and queries.
- **Modular architecture** for easy feature addition and maintenance.
- **Automated database migrations** (EF Core).
- **Centralized error handling** and logging.
- **API versioning** can be added for backward compatibility.

---

8. Technologies Used

- **Backend:**
  - ASP.NET Core 7+
  - Entity Framework Core (PostgreSQL, SQLite)
  - MediatR (CQRS)
  - ML.NET (for property price prediction)
  - Swashbuckle (Swagger for API docs)
  - xUnit, FluentAssertions (testing)
- **Frontend:**
  - Angular 16+
  - TypeScript
  - RxJS
  - Angular Material (or Bootstrap, as configured)
  - Jasmine, Karma (testing)

- **Other:**
    - Docker (for deployment)
    - SonarQube (code quality, optional)
    - Firebase (optional, for hosting and images bucket)
    - Stripe (for payments)