**PJLI: REAL ESTATE**

**TECHNICAL DOCUMENTATION**

**AUTHORS**

DELSON JAMES M.TUBIERA

Collabera, .NET Full Stack Trainee

[delsonjames17@gmail.com](mailto:delsonjames17@gmail.com)

MARK KIM BLAS

Collabera, .NET Full Stack Trainee

mrkkmbls@gmail.com

**TABLE OF CONTENTS**

**INTRODUCTION**

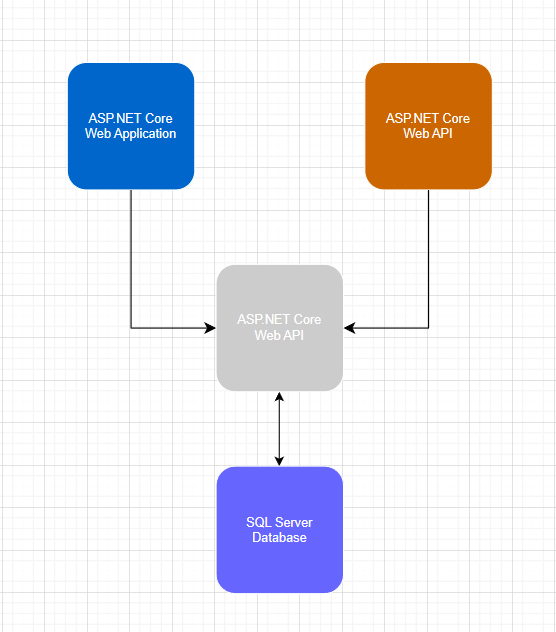
A home is a place where individuals or families reside and live their daily lives. It is a place where people feel a sense of comfort, security, and belonging. A home can be a house, an apartment, a condominium, or any other type of dwelling that provides shelter and a sense of personal space.A home is not just a physical structure but also a place where people form emotional connections and create memories. It is where individuals and families can relax, unwind, and be themselves. So we created the application that can help you buy or rent a home that fits your needs. We present the PJLI Real Estate Application.

The scope of the project is focused on delivering the core and basic functionality of the application given the short timeline of developing it. The key deliverables include a .NET Core MVC 5 web application, a .NET Core Web API, and a SQL Server database. All .NET projects will be created using the version of .NET, .NET 6. The app will also be developed by utilising Drow.IO by creating wireframes in order to offer a user-friendly experience.

Moreover, the application will be developed using agile methodologies to ensure efficient and effective development. Given the short time frame, the developers will focus on providing the application's main functionality within the constrained capstone timeline, with a priority on a user-friendly interface and an improved Real Estate search approach.

**ARCHITECTURE**

The application is composed of two primary components: a Web Application and a Web API, both of which communicate with a SQL Server Database. The Web Application offers a graphical user interface for job seekers and employers to look for and advertise job openings, while the Web API provides a RESTful interface for potential future development of the application. Although they are separate applications, they share the same database. They are developed using ASP.NET and C# and are containerized for convenient deployment to a Kubernetes cluster. The application employs Entity Framework with a Code First methodology to manage interactions with the database. This approach enables the application to define the database schema using C# code, which is then translated to SQL by Entity Framework. The database schema is versioned and can be modified using Entity Framework Migrations. An architecture diagram is provided to illustrate how the Web Application, Web API, and SQL Server Database interact.



**INSTALLATION**

**DEVELOPMENT ENVIRONMENT SETUP**

This section describes the steps required to set up the development environment for the project. Developers can use either Visual Studio or Visual Studio Code. The project uses .NET 6, MS SQL Server for database, and Docker and Kubernetes for containerization and deployment.

**Install Visual Studio or Visual Studio Code**

Visual Studio and Visual Studio Code are integrated development environments (IDEs) for developing .NET applications. Visual Studio is a fully-featured IDE with advanced debugging and testing tools, while Visual Studio Code is a lightweight code editor that supports multiple languages and extensions.

You can download Visual Studio or Visual Studio Code from the official Microsoft website:

Visual Studio: https://visualstudio.microsoft.com/downloads/

Install .NET 6 SDK

.NET 6 is the latest version of the .NET Framework, and is required to build and run the project. You can download the .NET 7 SDK from the official .NET website:

.NET 6 SDK: https://download.visualstudio.microsoft.com/download/pr/38dca5f5-f10f-49fb-b07f-a42dd123ea30/335bb4811c9636b3a4687757f9234db9/dotnet-sdk-6.0.407-win-x64.exe

**Install Microsoft SQL Server**

MS SQL Server is the database management system used by the project. You can download and install MS SQL Server from the official Microsoft website:

MS SQL Server: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

**Install Docker**

Docker is a platform for developing, shipping, and running applications using containers. Docker allows you to package the application and its dependencies into a container, which can then be deployed to any environment that supports Docker. You can download Docker from the official Docker website:

Docker: <https://www.docker.com/products/docker-desktop>

**Install Kubernetes**

Kubernetes is an open-source platform for managing containerized workloads and services. Kubernetes allows you to deploy and scale containers across multiple hosts and provides advanced features such as load balancing and automatic failover. You can download and install Kubernetes from the official Kubernetes website:

Kubernetes: https://kubernetes.io/docs/setup

**Clone the project repository**

Clone the project repository from the version control system using Git. Open a terminal or command prompt and navigate to the directory where you want to clone the project.

Then, run the following command:

git clone <https://github.com/Delson-James17/REALESTATE_FINAL.git>

**Build and run the project**

Open the project solution in Visual Studio or Visual Studio Code. Restore the NuGet packages and build the project. Then, run the project to start the web server and the database.

You can use the following commands to build and run the project:

dotnet restore

dotnet build

dotnet build

Alternatively, you can use docker-compose to build and run the projects in a container.

You can use the following commands to build and run the Docker container:

docker-compose –f docker-compose.yml up

That's it! With these steps, you should have a fully-functional development environment set up for the project.

**DATABASE CONFIGURATIONS**

The application uses Microsoft SQL Server as its database management system. Here are the details for configuring the database:

**Database Connection String**

The connection string for the database is specified in the appsettings.json file of the project. The following is an example of a connection string:

This connection string specifies that the server is running on the local machine, the database name is JobBoardsDb. Update this connection string as per your SQL Server configuration.

**Database Migrations**

The application uses Entity Framework Core to manage database migrations. To create or update the database schema, you can use the following command in the Package Manager Console:

Update-Database

This command applies any pending migrations to the database.

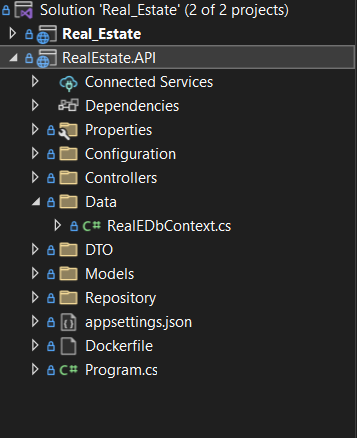
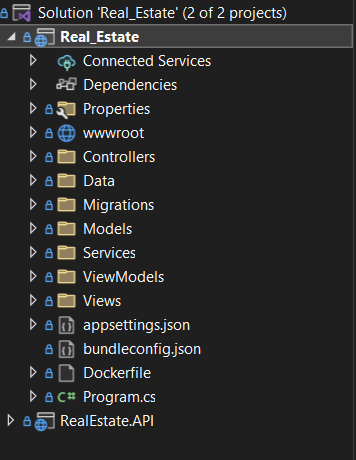
**DEVELOPMENT**

**TECHNOLOGIES**

The application is built using the following technologies:

* **.NET 6**: The long term support version of the .NET framework is used for developing both the Web Application and the Web API.
* **C#**: The primary language used for building the application is C#, a modern, object-oriented language that is used extensively in the .NET ecosystem.
* **ASP.NET**: The application uses ASP.NET to build web applications, which provides a rich set of features for building modern web applications.
* **MS SQL Server**: The application uses MS SQL Server as its database, which provides a powerful and reliable database platform for storing and querying data.
* **Entity Framework Core**: The application uses Entity Framework Core as its Object-Relational Mapping (ORM), which provides a powerful set of tools for working with databases in .NET.
* **Docker**: The application is containerized using Docker, which provides an easy way to package applications and their dependencies into a portable container.
* **Kubernetes:** The application is designed to be deployed to a Kubernetes cluster, which provides a powerful orchestration platform for managing containerized applications.
* **Swagger :** The application uses Swagger to document the Web API. Swagger provides a powerful tool for describing and exploring RESTful APIs, making it easier for developers to understand and work with the API.
* **ASP.NET Identity** ASP.NET Identity is used for authentication and authorization in the application. It provides built-in support for managing user accounts, roles, and claims. The application uses ASP.NET Identity to authenticate users and authorize access to protected resources.

**CODE STRUCTURE**

* This project is organized into two main projects: Real Estate.Api and Real\_Estate.WebApplication.
* The RealEstate.Api project functions as the Web API component of the application, providing an interface for accessing job postings and related data. It's developed using ASP.NET Core and is in charge of handling incoming HTTP requests and returning responses in the JSON format.
  + 
* The Real\_Estate.WebApplication project is responsible for the user interface component of the application, utilising ASP.NET Core Razor Pages. It offers clients and owners a web-based platform to browse for high-quality houses and post buildings that are available for sale or rent.
  + Below is the actual image of the folder structure.

**CODING STYLES**

The team followed a set of coding style guidelines to ensure consistency and readability across the codebase. Here are some of the main conventions that we adhered to:

* PascalCase was used for naming classes, interfaces, and public methods, which means that the first letter of each word is capitalized.
* camelCase was used for naming local variables and method parameters, which means that the first letter of the first word is lowercase and the first letter of subsequent words is capitalized.
* (\_) underscore notation was used as a prefix for private fields, which means that it starts with an underscore followed by a camel-cased variable name.
* "I" was also used as a prefix for interfaces, which means that the interface name starts with an uppercase "I".
* Brackets should be on a separate line for better code readability.
* Comments should be used sparingly but effectively to explain non-obvious code and algorithms.

By following these conventions, we aimed to create code that is easy to read and understand for all team members involved in the development process.

**VERSION CONTROL**

We use Git for version control and GitHub as a remote repository to store and manage our codebase. With Git, developers can collaborate on code changes and track its history. GitHub provides a user-friendly interface for managing repositories, issues, and pull requests. We create a new branch for each feature or bug fix and submit pull requests for review before merging them into the main branch. This workflow allows us to ensure the code is thoroughly tested and vetted before integrating it into the master branch. Overall, our use of Git and GitHub facilitates efficient collaboration and ensures our codebase is well-maintained and up-to-date.

**API REFERENCES**

The project has multiple API endpoints that are used to support the application's functionality. These endpoints only accept and return data in JSON format. In order to access these endpoints, a valid access token must be included in the Authorization header of the HTTP request. Any unauthorized attempts to access the endpoints will be denied and the server will respond with a 401 Unauthorized HTTP status code.

**AUTHENTICATION API**

This API provides endpoints for user authentication and authorization, the following endpoints are available:

**[POST] /api/Account/Login**

Login an existing user and returns an access token.

Example Request Body:

{

"email": "admin@gmail.com",

"password": "P@ssword123"

}

Example Response: 200 OK

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9lbWFpbGFkZHJlc3MiOiJhZG1pbkBnbWFpbC5jb20iLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBZG1pbiIsImV4cCI6MTY4MjM5MjAzMywiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDozMDAxMCIsImF1ZCI6IlVzZXIifQ.yO9t3gPz9EO390gv-etBwidqLahPWenSOb-bzDQtUIg"

}

**ESTATE PROPERTIES API**

This API provides endpoints for managing Real Estate. The following endpoints are available:

**[GET]/apiI/EstateProperties**

Retrieves a list of all EstateProperties.

Query Parameters

pageNumber (INTEGER), itemsPerPage (INTEGER)

Example Response: 200 OK

**[POST]/apiI/EstateProperties**

Creates a new EstateProperties.

Example Request Body:

Example Response: 201 Created

**[GET]/apiI/EstateProperties/{id}**

Retrieves a specific EstateProperties by its ID.

Example Reponse: 200 OK

**[DELETE] /api//EstateProperties/{id}**

Deletes a specific EstatePropertie by its ID.

Example Response: 204 NO CONTENT

**APPOINTMENT API**

This API provides endpoints for managing Appointment. The following endpoints are available:

**[GET]/apiI/Appointment**

Retrieves a list of all Appointment..

Query Parameters

pageNumber (INTEGER), itemsPerPage (INTEGER)

Example Response: 200 OK

**[GET]/apiI/Appointment/{id}**

Retrieves a specific Appointment by its ID.

Example Reponse: 200 OK

**[DELETE] /api/Appointment//{id}**

Deletes a specific Appointment by its ID.

Example Response: 204 NO CONTENT

**PROPERTY CATEGORIES API**

This API provides endpoints for managing Property Categories. The following endpoints are available:

**[GET]/apiI/PropertyCateogry**

Retrieves a list of all PropertyCateogry.

Query Parameters

pageNumber (INTEGER), itemsPerPage (INTEGER)

Example Response: 200 OK

**[POST]/apiI/PropertyCateogry**

Creates a new PropertyCateogry.

Example Request Body:

Example Response: 201 Created

**[GET]/apiI/PropertyCateogry/{id}**

Retrieves a specific PropertyCateogry by its ID.

Example Reponse: 200 OK

**[DELETE] /api//PropertyCateogrys/{id}**

Deletes a specific PropertyCateogry by its ID.

Example Response: 204 NO CONTENT

**SALE OR RENT CATEGORIES API**

This API provides endpoints for managing **SALE OR RENT CATEGORIES**. The following endpoints are available:

**[GET]/apiI/SaleorRentModel**

Retrieves a list of all SaleorRentModel.

Query Parameters

pageNumber (INTEGER), itemsPerPage (INTEGER)

Example Response: 200 OK

**[POST]/apiI/SaleorRentModel**

Creates a new SaleorRentModel.

Example Request Body:

Example Response: 201 Created

**[GET]/apiI/SaleorRentModel/{id}**

Retrieves a specific SaleorRentModelby its ID.

Example Reponse: 200 OK

**[DELETE] /api//SaleorRentModel/{id}**

Deletes a specific SaleorRentModelby its ID.

Example Response: 204 NO CONTENT

**DATABASE DESIGN**

PJLI Real Estate uses Microsoft SQL Server as the primary database management system. The database schema is designed using Entity Framework Code-First approach, which allows for easy modification and maintenance of the database structure.

**Entity Relationship Diagram (ERD)**

Below is the ERD of the PJLI Real Estate database: