Health & Fitness Journal Portal

Introduction

The system provides the requested feature using different modules and development frameworks. The implementation is divided into two main projects,

1. Frontend
2. Backend

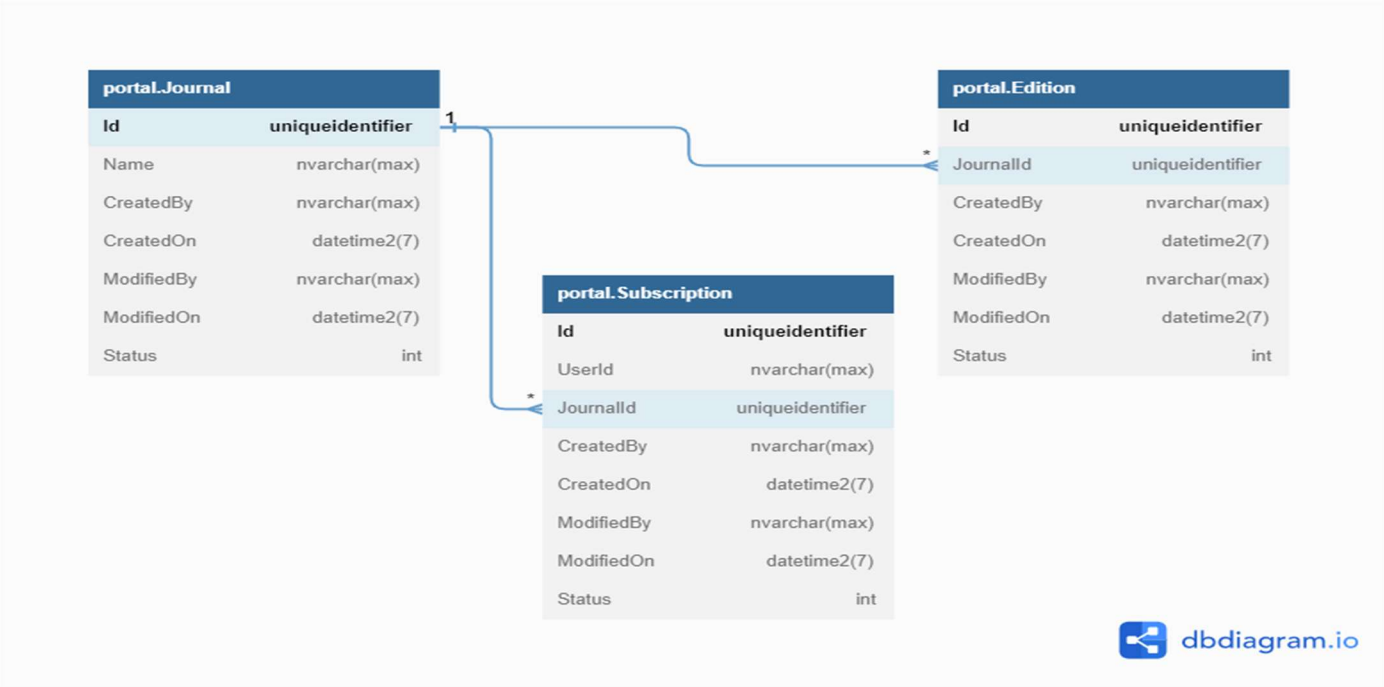
The frontend provides UI or view to the system, which interactions with controllers. The controllers are managed by the backend application. The controllers then use models to store or retrieve data from a datastore.

The project implements the UI with help of a single page application build with Angular 15. The theming is implemented using Angular Material 15.

The project implements the backend system with,

1. MVC based project in .net core 6.0
2. Identity management using ASP.Net Identity
3. Secure access via jwt tokens
4. Database interaction using ORM, EntityFramework Core.

Database ERD



Backend:

The system is separated into various modules/projects to gain separation of concern. Each module has a single responsibility.

portal.Domain:

This project contains the domain services, as well as interfaces for interacting with these objects. It contains the database migrations & database context to interact with the databases.

portal.Domain.Entities:

This project contains the data models or entity classes that represent the data stored in the database. Code first approach is used for database design. These classes are mapped to database tables using an object-relational mapping (ORM) framework.

portal.IO:

This project provides common input/output (I/O) functionality for the system. It defines classes and interfaces for reading and writing files, working with streams, and other I/O operations. It defines the IStorageProvider to provide an interface for storage of files. This is implemented by a FileStorageProvider which helps in storing the files on file storage. The IStorageProvider can be implemented for various other file storage options e.g. cloud storage.

portal.IO.Common:

This project contains common utility classes and functions used in the system. In the system the implementation to check if the file uploaded is a valid PDF is defined in this project.

portal.Security:

This project provides security-related functionality for the system. It defines classes and interfaces for user authentication, authorization, encryption, and other security-related tasks. The system relies on token-based authentication. IJWTProvider is an interface defined for the system. An implementation JWTProvider is defined to generate jwt token.

portal.Security.Identity:

This project provides user authentication and authorization functionality using an identity and access control framework ASP.NET Identity. The project is responsible for all the objects, business logic for authentication and secure access to endpoints.

portal.WebAPI:

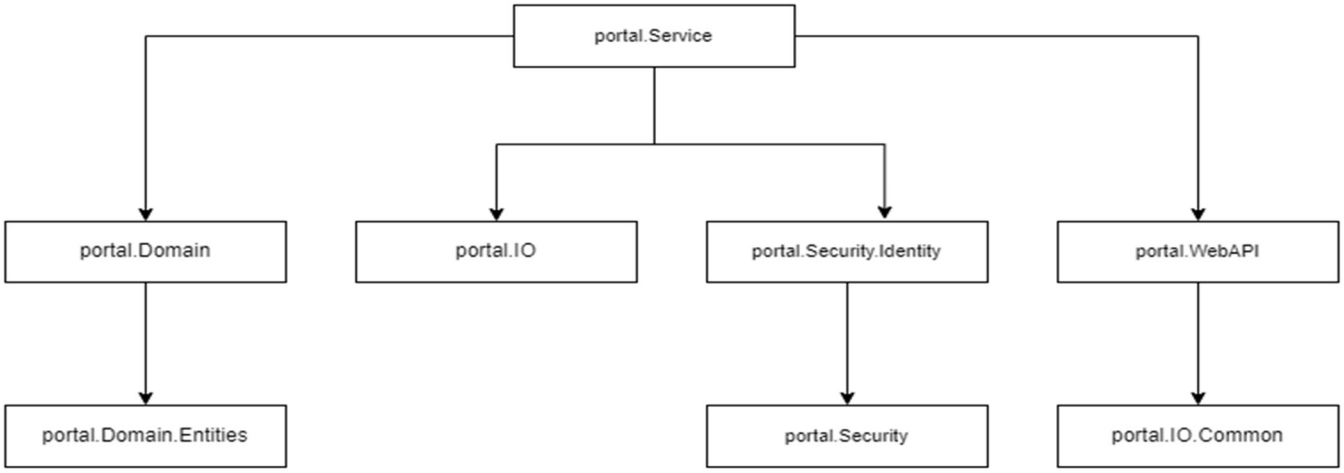
This project provides a RESTful web API for accessing the system's functionality over HTTP. It may define controllers and action methods that map to HTTP requests and use dependency injection to access other components in the system.

portal.Service:

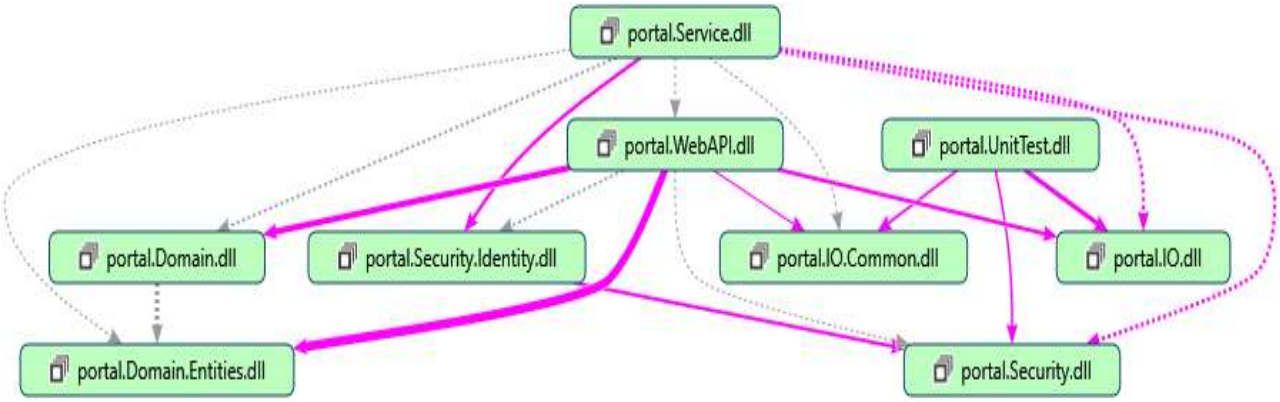
This project provides business logic and application services for the system. It defines classes and interfaces for performing complex operations and coordinating interactions between multiple components. This project is the application host which is created in .net core. This project is the startup project which is responsible to provide functionality to main app.

Data flow & component interaction

Below is a high level new of the project, from a portal.Service perspective.



Below is the detailed view of interaction between different projects.



Based on these dependencies, the data flow can be described as below

1. The portal.Service brings all individual modules together to start the application, based upon all the configuration.
2. The user sends a request, the portal.Service delegates it to the portal.WebAPI component, specifying the operation they want to perform and any relevant data.
3. The portal.WebAPI component receives the request and validates the user's credentials using the portal.Security.Identity component. If the user is authorized, the request is passed to the appropriate handler method.
4. The handler method in portal.WebAPI interacts with the portal.Domain component to retrieve or modify data, and with the portal.IO component to store or retrieve files.
5. Once the operation is complete, the portal.WebAPI component returns a response to the user, including any data or status codes.

Front End:

The application structure is in the following pattern,

The **core** folder contains the services, interceptors, and guards used throughout the application. The auth folder contains the authentication-related services and interceptors, while the **http** folder contains an HTTP interceptor to handle errors. The services folder contains a shared data service to fetch data from the API.

The **features** folder contains subfolders for each major feature of the application, in this case journals and subscriptions. Each feature subfolder contains subfolders for each component, including a list component, details component, and form component. These components use the shared **portal.service** to interact with the API.

The **shared** folder contains shared components and models used throughout the application.

The app.component is the root component of the application, and it contains the header component and the router outlet to switch between components.

Design Patterns

Repository pattern:

The portal.Domain component uses the repository pattern to encapsulate data access logic and provide a simple and consistent API for interacting with domain objects. The repository pattern separates the domain objects from the underlying data storage and provides a layer of abstraction that makes it easy to change the data access implementation without affecting the rest of the system.

Dependency injection:

The system may use dependency injection to manage component dependencies and promote loose coupling between components. Dependency injection allows components to specify their dependencies through constructor parameters or setter methods, which are then injected at runtime by a container or framework. IStorageProvider, IJWTProvider are examples implemented in the project.

Model-View-Controller (MVC) pattern:

The portal.WebAPI component is likely to use the MVC pattern to separate concerns and promote maintainability. The MVC pattern separates the user interface (view) from the business logic (model) and the controller that mediates between them. This approach makes it easier to modify or extend the system without affecting other parts of the system.