Henry Faulkner
10/03/2024

Amplifund API Architecture

The solution's namespaces follow the [standard convention ascribed by Microsoft]: **[Company].[Product][.<Feature>][.<Subnamespace>]**

Preface:

- **ServiceCollectionExtension** is a convention to supply extension methods to the IServiceCollection.

Per project synopsis:

1. Amplifund.Assignment.API.Core

a. This **ASP.NET Core Web API** standardizes the configuration and startup instructions for running API programs.
b. The **Starter** class initializes the WebApplicationBuilder, which configures the app and hosts it.
c. The **ServiceConfiguration** class offers a common method to configure services, middleware, and endpoints, **ConfigureAPIService**, and a common method to begin the program, **RunApp**.
d. It also supplies common classes:
   i. **BaseController**: Generic base controller, providing common functionality for other API controllers.
   ii. **ApiActionResult and ApiActionResult<T>**: Implementations of IActionResult designed to standardize how API responses are formatted and returned by the application.
   iii. **ApiResponse**: Generic class designed to standardize the structure of API responses.
   iv. **ApiException**: Encapsulates key information about an exception, occuring during API execution.

2. Amplifund.Assignment.API.API1

a. This **ASP.NET Core Web API** is my **running program**. It utilizes the Core API project for configuration and startup instructions. It also uses the BL Service extensions to inject service dependencies (discussed later).
b. It has controllers which handle HTTP requests and responses.

c. The controllers use services to perform business logic and return results to the client.
d. I use constructor dependency injection to inject services and the logger into each controller. Contracts (interfaces) are injected, so as to abstract the underlying implementations.

3. Amplifund.Assignment.Logger

a. This is a class library, whose purpose is to support the logging efforts of the solution.
b. In the current implementation, there is only one class called ServiceCollectionExtension. **ServiceCollectionExtension**, in this context, adds the CreateLogger method to IServiceCollection to integrate Serilog into the service container.

4. Amplifund.Assignment.Model

a. This class library contains **data transfer objects**, used to transfer data between different layers of the application, primarily between the API and the client.
b. Normally, I would implement the **AutoMapper** library to simplify converting entities to DTOs and DTOs to entities. For the sake of time, I had to forego this pattern.

5. Amplifund.Assignment.BL.Common and Amplifund.Assignment.BL.Grants

a. These are my services for performing **business logic**.
b. Services interact with my repositories to fetch and persist data.
c. I use constructor dependency injection to inject the Unit of Work, the logger, and repositories into each service. Contracts (interfaces) are injected, so as to abstract the underlying implementations.
d. I use the **ServiceCollectionExtension** convention to provide a method to set up the services provided by the project and register the data layer dependencies.

6. Amplifund.Assignment.Domain

a. This class library contains the schemas for application's **domain objects** (the app's db table reflections).
b. It also contains the interface contracts for the application's **data repositories**.
c. It also contains the interface contract for the application's **Unit of Work**.

7. Amplifund.Assignment.Data

a. This class library handles the data access layer by implementing **repositories**.
b. I use **EF Core** as an ORM technology to strongly-type my database results and to capitalize on LINQ queries.
c. I use **EF Core Migrations** to construct the database in a Code-First fashion.
d. I use the **Unit of Work** pattern to ensure that a set of database operations are treated as a single unit. If one operation fails, all other operations in that transaction are rolled back, maintaining data integrity.
e. I use **SQLite** as my application's database provider. Therefore, the full database is in this project in the file **app_database.db**.
f. I use the **ServiceCollectionExtension** convention to provide methods to configure the database and to set up dependency injection for the Unit of Work pattern [Scoped - once per request] and the repositories [Transient - created each time it is requested].