1.2 NLayer Architecture

1.2.1 Introduction

Layering of an application's codebase is a widely accepted technique to help reduce complexity and improve code reusability. To achieve layered architecture, StudioX follows the principles of **Domain Driven Design**. In Domain Driven Design there are four fundamental layers:

- **Presentation Layer**: Provides an interface to the user. Uses the Application Layer to achieve user interactions.
- **Application Layer**: Mediates between the Presentation and Domain Layers. Orchestrates business objects to perform specific application tasks.
- **Domain Layer**: Includes business objects and their rules. This is heart of the application.
- **Infrastructure Layer**: Provides generic technical capabilities that support higher layers mostly using 3rd-party libraries. An example of the Infrastructure Layer can be a Repository implementation used to interact with a database through an ORM framework, or an implementation for an email provider to send emails.

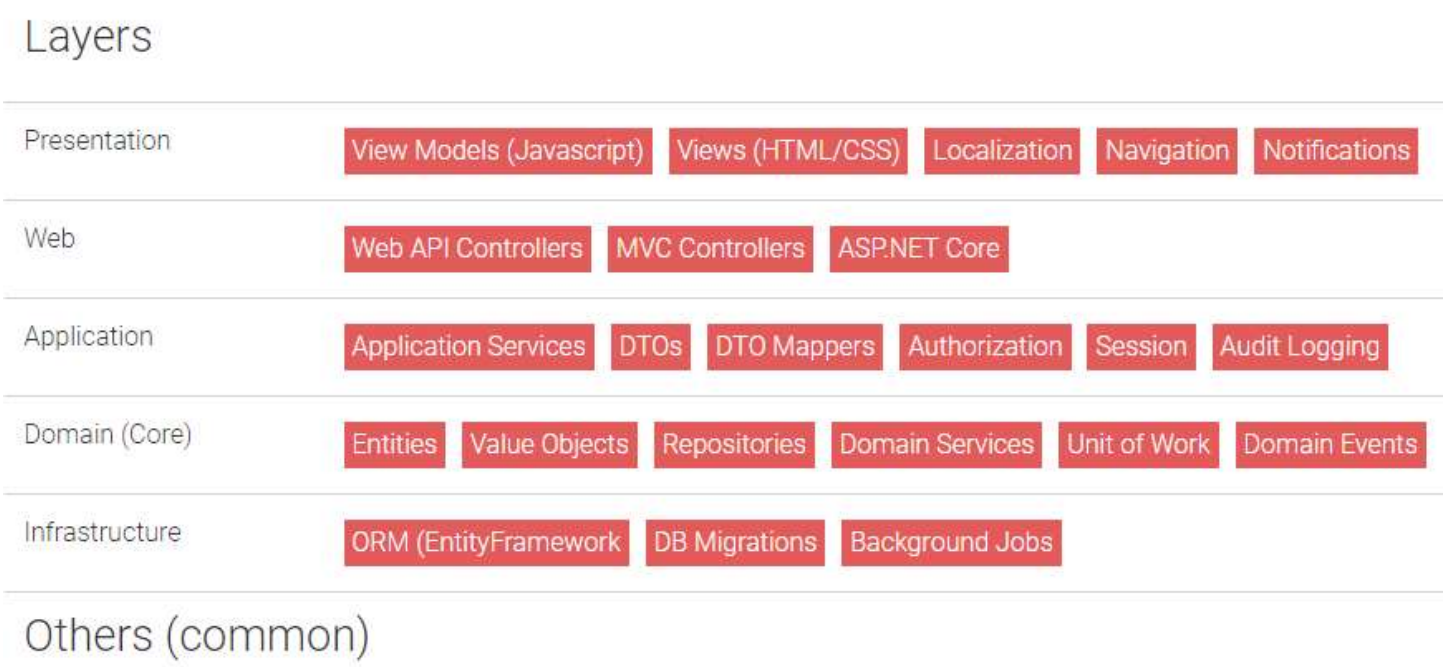
There may be additional layers added as necessary. An example being:

- **Distributed Services Layer**: Used to expose application features to remote clients. There are tools like ASP.NET Web API and WCF that can provide this layer.

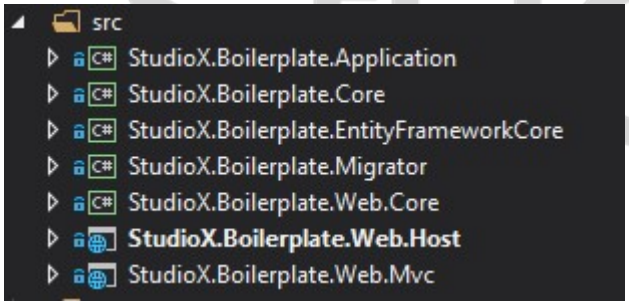
These are all common layers of a domain-centric architecture. There may be minor differences based on implementation.

1.2.2 StudioX Architecture

Overview of layers and structures are shown below:



Here is a solution with projects for a simple layered application:



A layer can be implemented as one or more assemblies. It may be good to create more than one assembly for third-party dependencies (like Entity Framework here) for larger projects. Also, there may be bounded contexts where every context has it's own layers.

1.2.3 Domain (Core) Layer

Domain Layer is where all business rules should be implemented.

- **Entities** represent data and operations of the business domain. Generally they are mapped to database tables in practice.

- **Repositories** are collection-like objects and are used to retrieve and persist entities on a data source (database). Domain Layer defines repositories, but does not implement them. They are implemented in the Infrastructure Layer.
- **Domain Events** are used to define domain-specific events as well as to trigger and handle them. Domain services work with entities (and other domain objects) and implement business rules which do not belong within a single entity.
- **Unit of Work** is a design pattern used to manage database connection and transaction, track entity changes and save changes to a data store. It's defined in domain layer, but implemented in infrastructure layer.

This layer should be independent of third-party libraries as much as possible.

1.2.4 Application Layer

Application Layer contains application services those are used by the Presentation Layer. An application service method can receive a DTO (Data Transfer Object) as input, uses this input to perform some specific domain layer operation, and may return another DTO, if needed. It should not receive or return entities. An application service method is generally considered a Unit of Work. User input validation is also implemented in this layer. It's suggested to use a tool for mapping entities to DTOs such as the AutoMapper library. We have also session here to obtain information on current user.

1.2.5 Infrastructure Layer

While Domain Layer defines interfaces for repositories, unit of work and other services, Infrastructure Layer implements those interfaces. It implements repositories using ORM tools like EntityFramework. StudioX provides base classes to work with these two ORM frameworks. Infrastructure Layer is used to abstract away dependencies on third-party libraries from the other layers.

Beside database access, we may have abstractions for service providers. For example, we may use a vendor to send SMS messages. We can define an interface in domain or application layer to abstract it from our code. Then we can implement that interface in the infrastructure layer.

1.2.6 Web & Presentation Layers

Web Layer is implemented using ASP.NET MVC, Web API. Two different approaches can be implemented here: Single-Page Applications or Multi-Page Applications. Startup templates support both of them.

In a **Single-Page Application (SPA)** all resources are loaded once (or a core resource is loaded and others are lazy loaded when needed) to the client (browser) and then all subsequent interaction with the server is made using AJAX calls. HTML code is generated on the client-side with data received from server. The entire page is never refreshed; views are just swapped in and out as necessary. There are many Javascript SPA frameworks, such as AngularJs, BackboneJs, and EmberJs. StudioX can work with any of them, but provides samples and some helper mechanisms to easily work with Angular.

In a **Multi-Page Application (MPA)**, client makes a request to the server, the server side code (ASP.NET MVC Controllers in general) gets data from the database, and Razor views generate HTML. These generated pages are sent back to the client to display it. Each new page results in a full page refresh. Client then can make additional AJAX requests for better user experience.

SignalR is a perfect tool to send push-notifications from the server to the client. It can be used to provide a rich, real-time experience to the user.

StudioX provides infrastructure to automatically create a Web API Layer from your application services and easily use it with Javascript (see documentation). Additionally, it provides infrastructure to manage application menus, localization, and language switching. Also included is a simple and unified Javascript API to simplify showing system messages and notifications.

StudioX automatically handles exceptions in server-side and returns an appropriate response to the client.

StudioX uses and supports Dependency Injection through the Castle Windsor framework. It also uses Log4Net for logging server-side, however, easily supports swapping in other logging libraries without code change by the help of Castle's abstract logging facility.