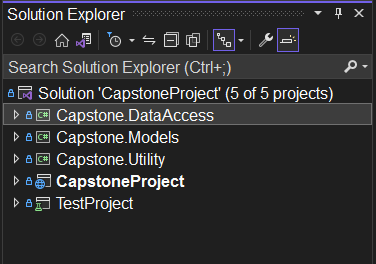
**System Design**

The project uses several design patterns to structure the code around: N-Tier architecture, Repository pattern, Unit of Work pattern and M.V.C.(Model, View, Controller).

N-tier architecture, also known as multi-tier architecture, is a software design pattern that divides an application into separate logical layers, each responsible for specific functionality. This architectural style is commonly used in the development of large-scale enterprise applications. The "n" in n-tier represents any number of layers or tiers, as the architecture can be adapted to the specific requirements of the application. For the purpose of my project, these layers are put into separate projects for ease of access outside the main project in the same solution.



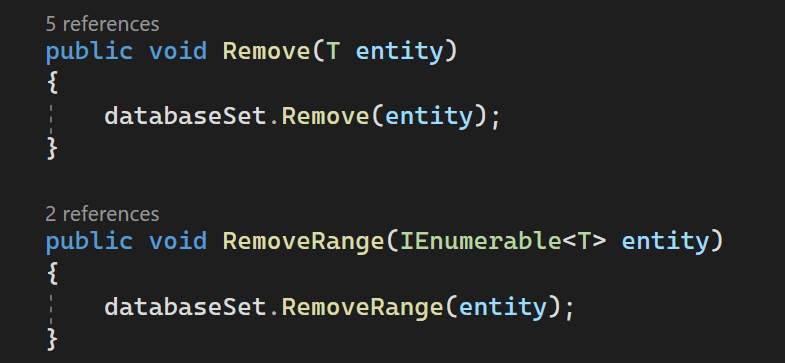
The Model layer represents all the classes by which the data will be structure and manipulated from. These classes strictly hold the properties of the class and contain no functions, methods or any sort of functionality. They are used to bind and map out objects that represent elements from the database.

The Utility layer merely holds static constants that can be used and accessed anywhere and contain no class specific logic.

The Data Access layer is responsible for handling the database connections and contains the SQL logic in the form of Repository patterns. Furthermore, the Data Access layer contains all databases migrations which represent a history of major changes to the database structure and can seed data into the database.

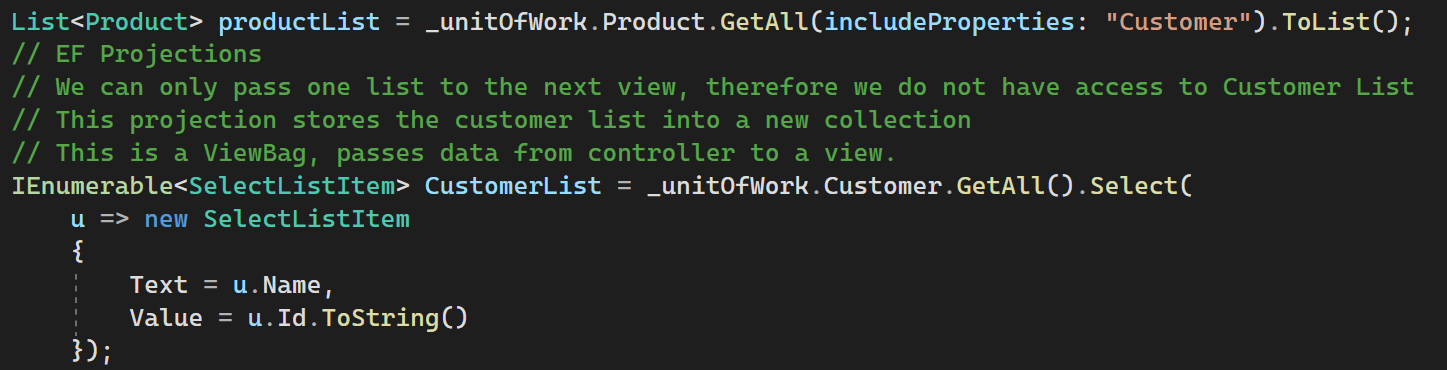
The Repository Pattern is a design pattern commonly used in software development, particularly in the context of data access and persistence. It provides a way to separate the business logic of an application from the underlying data storage implementation, such as a database.

The main purpose of the Repository Pattern is to create an abstraction layer between the application and the data access code. It defines a set of standardized methods and interfaces for performing CRUD (Create, Read, Update, Delete) operations on data objects without exposing the specific details of the data storage mechanism.

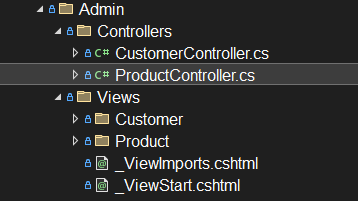
The above screenshot represents common functionality that is shared among all the data models. Rather than having each class have its own implementation of a simple CRUD operation, these operations will be contained within repositories that are shared between all the models.

The Unit of Work design pattern is a software design pattern commonly used in applications that involve data persistence, such as database operations. It provides a way to manage multiple database operations within a single transaction and ensures consistency and integrity of data.

The main objective of the Unit of Work pattern is to group related operations together and treat them as a single unit, typically within a transaction. It helps to maintain data consistency by ensuring that all changes are either committed together or rolled back as a whole. This pattern usually works hand in hand with the repositories. All repositories are interfaced in the unit of work class and interface and from the same class, you can access all the data models and their respective methods.

 In the screenshot above, you see that through the same instance of \_unitOfWork we can access the Product model and the Customer model and perform their different operations from the same object.

The MVC (Model-View-Controller) design pattern is a widely used architectural pattern for developing software applications with a user interface (UI). It separates the application's concerns into three interconnected components: the model, the view, and the controller. Each component has a specific responsibility within the overall structure of the application.



In this example, the model data is retrieved from the Product class in the Model folder mentioned from previously. The ProductController will then take this model and manipulate it and transfer that information in the Product View to display to the user.