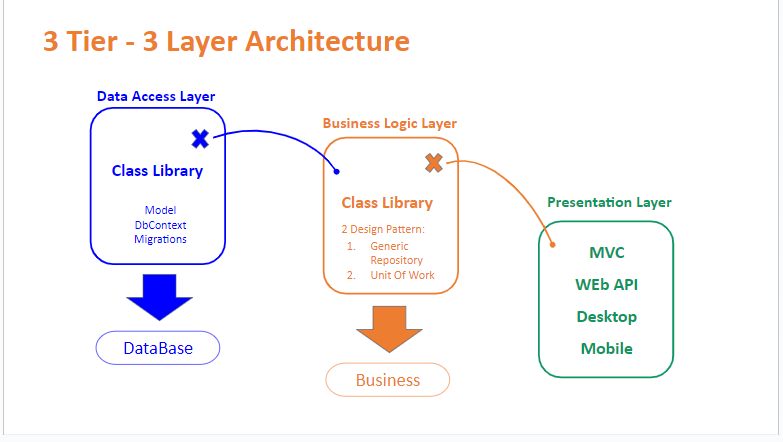
3 Tier – Architeture



Simply we can say it’s a way to organize our application into 3 Sections

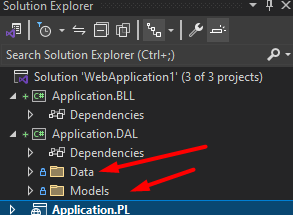
1 – Data Acess Layer -> This will hold all information about our models and database

2 – Bussiness Logic layer -> this will hold all logic behind our Models

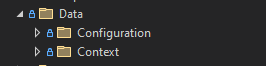
3 – Presination layer -> will hold all view to Our Client

**Data Acess Layer**

**This access layer will hold all Infromation About OUR DataBase configuration**



**Data Folder -> will Hold Context Folder and Configuration Folder**



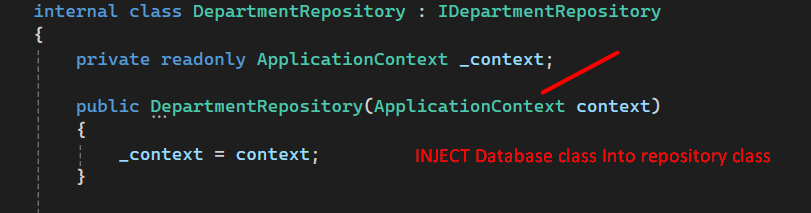
**In our Data access layer we create two Folders**

**Data -> will hold all Data connetions info**

**Models -> will hold all Models Info**

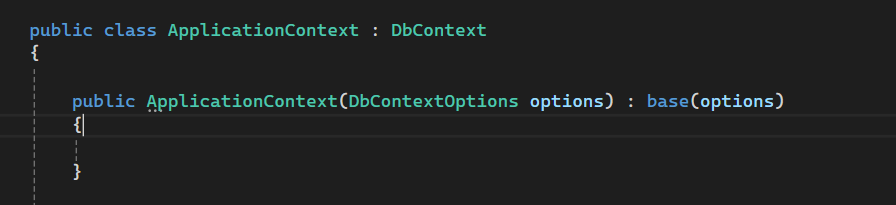
**DataConnection Configuration:**

**When we create A repository we told him to create object from ApplicationContext class to establish a database connection to make an operations on it**

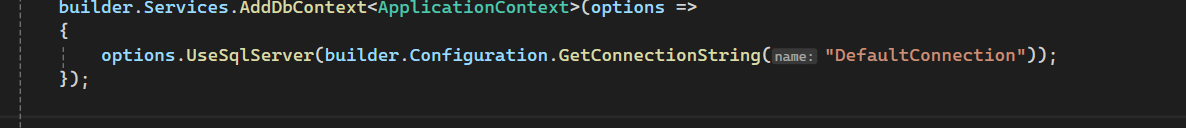


**When we run the application and request the Department Repository the CLR Will inject ApplicationContext then try to create object from it.**

**When clr go to ApplicationContext**

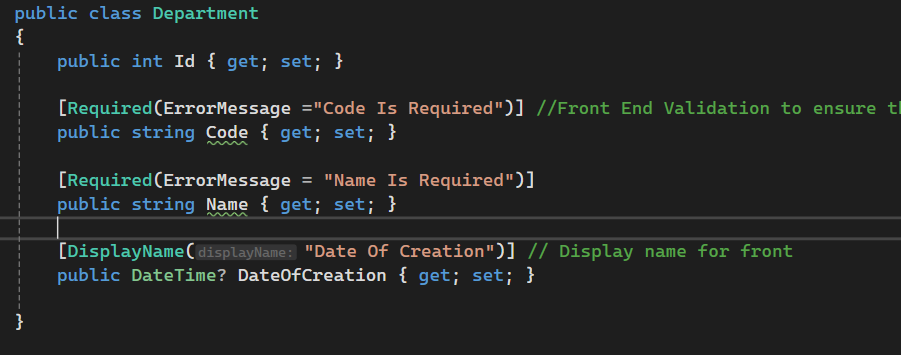


**He will find that we inject another DbcontextOption class to it so he need to create another object then pass it to the parent.**



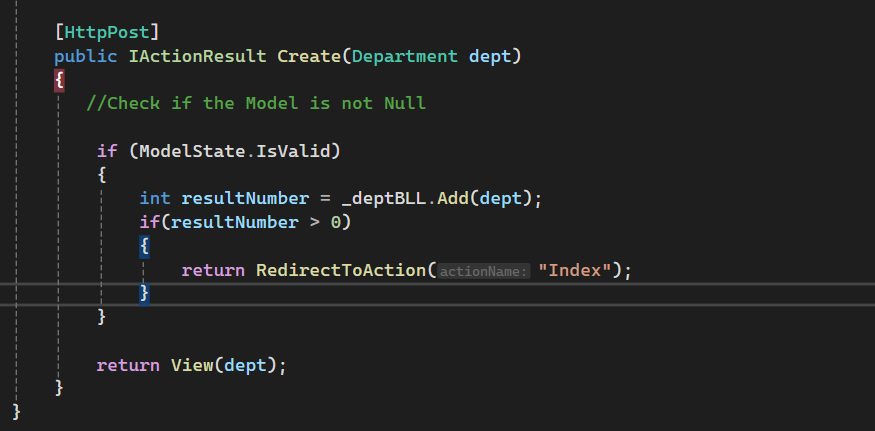
**In this clase we told the compiler that clr will handle creating object OF dBCONTEXTOPTION and use the connection string in the appsetting.json**

**Validate Inputs And Insert Data into Database:**



**[Required] attributes make this field is required into database and front end**

**And Error message this message will show if we put null into this field**



**[HttpPost] -> meaning this will be a Post request into database**

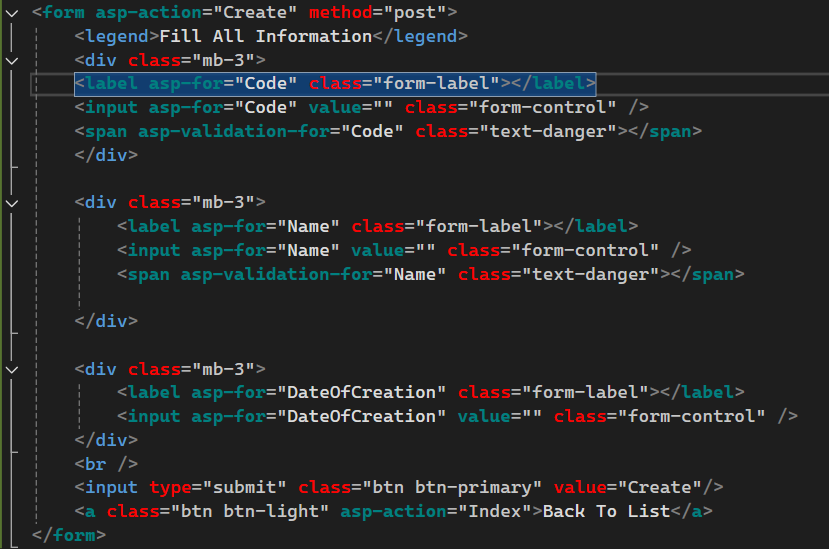
**Modelstate.IsValid -> this check if the model is empty or not and validate the Model in our database**

**Or our Model in DAL**

**If true then check if the number returned from database is larger than 0 that meaning it’s inserted sucssesfuly**

**So redirect the user to the index**

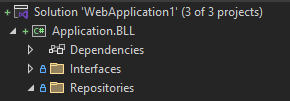
**Else make the request to the same Create page and show error message to him**



**Business Logic Layer:**

**This layer act as a kitchen it will cook all the meals.**

**In our case This layer act as a waiter take the Request from the controller in PL layer then go to talk to the Data acess layer then back with the information**



**Interface -> will hold the Defenetion or the Bluebrint of the Method in the repository**

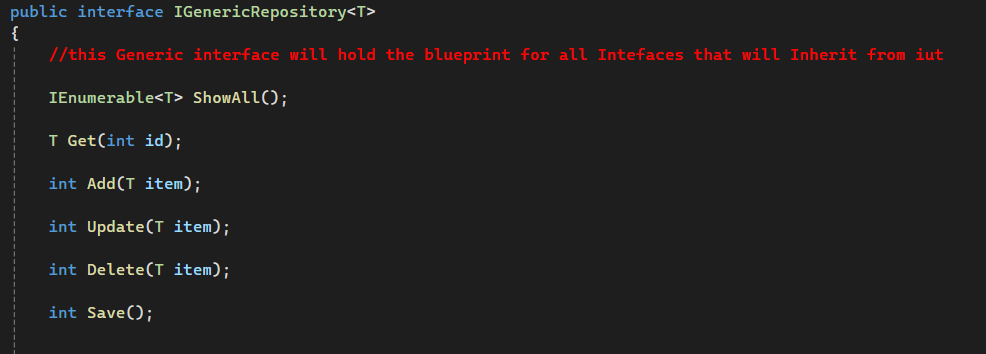
**Repository -> each model will has a repository (Waiter)**

**Asp.Net core MVC Session 04 Notes :**

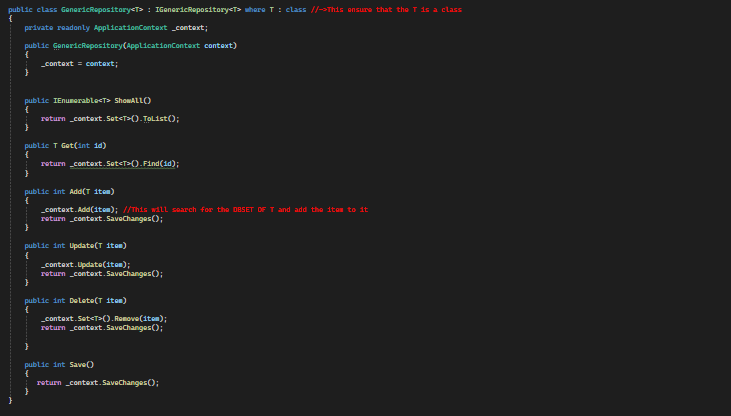
**Generic Repository**

**The Generic Repository is a design pattern that abstracts the application’s data layer, making it easier to manage data access logic across different data sources.**

* **Why Use a Generic Repository?** 
  + **Reduces Code Duplication: Common CRUD operations (Create, Read, Update, Delete) are written once in the generic repository and can be reused across all entities.**
  + **Centralizes Data Access Logic: All data access logic is managed in a centralized location, making the codebase easier to manage and understand.**
  + **Promotes DRY Principle: "Don't Repeat Yourself" by reducing redundancy in data access code.**
  + **Testability: Allows for easier unit testing by using dependency injection and mocking.**



**Implementing It**



* **A Partial View is essentially a view that renders a portion of HTML and can be reused across multiple views or layouts.**
* **They allow you to break down large views into smaller, reusable components that can be included in different views or layouts.**
* **Partial Views are similar to regular views, but they do not run on their own, they must be embedded within other views.**
* **Partial views are perfect for rendering small, reusable parts of the UI like navigation bars, forms, or lists that are used across different parts of an application.**