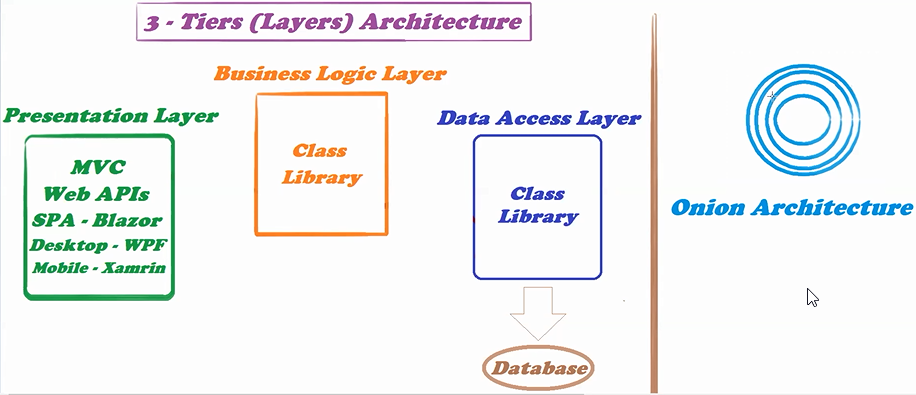
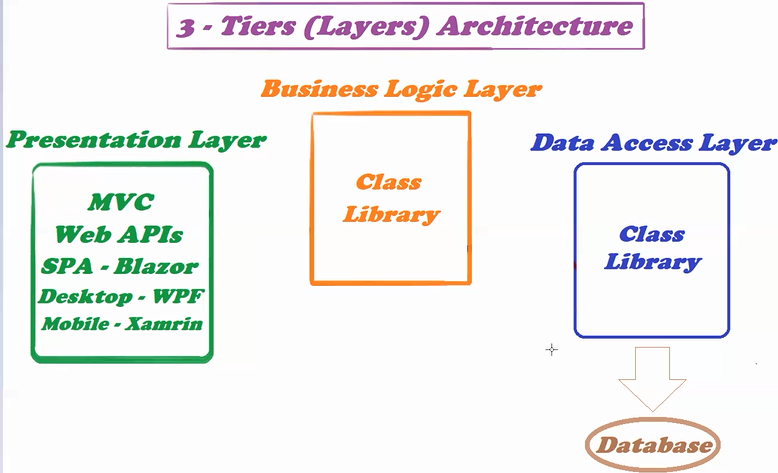
**Architecture Pattern:**

* Before starting any project decide the architecture pattern you will use to build that project
* Software architect or development manager is responsible for choosing the architecture pattern for the project
* We have 2 categories for projects classification:
* Monolithic Architecture: the project is divided modules or services in the same project /or solution /or system 🡺 what we’ll cover on MVC course, Examples for monolithic architecture pattern are 3-layer and onion
* Micro Services: each group of services in separate project [each project may be implemented with different technology] usually how large-scale projects work



* Onion architecture has 4 layers, the inside layer called core layer 🡺 this code layer has the whole project not implemented[has all the interfaces of the project] the implementation of those interfaces are in the other upper layers
* The onion architecture pattern is better than 3-layer architecture pattern in some cases 🡺 discuss them in the web API project

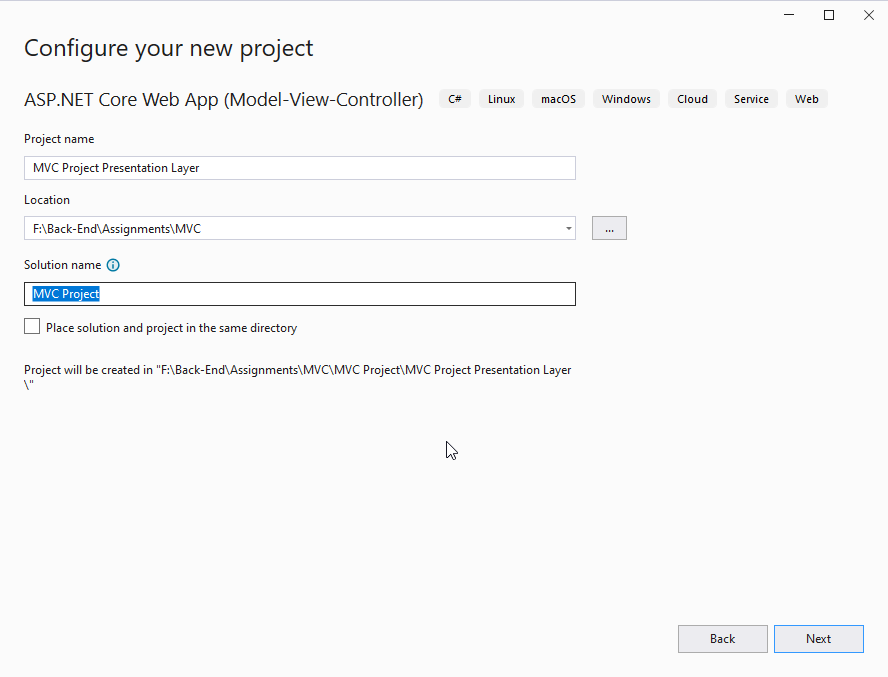
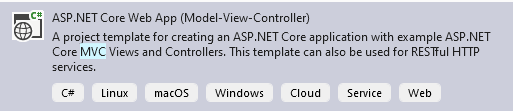
**3 Layers Architecture Pattern:**



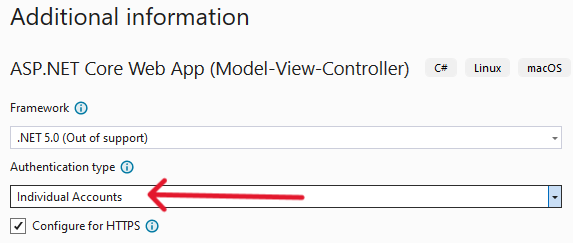
* Why we are using Architecture Pattern despite of implementing the principle of separation of concern in the MVC Project?
  + Example IKEA: needs to make dashboard for its employees to manage the system [adding/deleting/editing products or HR needs to change salary for employee…]
  + We can choose to build the project as server side application [MVC/Razor pages] or client side application, why not choosing single page application? As the users are limited to the Company employees
  + We’ll choose MVC project [just MVC app without using 3-layer architecture pattern]
  + Then IKEA asked for 2 applications for end user[customers] 🡺 web [SPA-Angular] and mobile [flutter] applications
  + For those 2 end user applications we need the same database and the same logic, so instead of building the same project database [data access layer] and the same logic [business logic layer] again, we’ll separate each one of them in a layer and let the web and mobile and the dash board application in a separate layer 🡺 that’s the 3-layer architecture pattern
* This architecture implements the principle of separation of concern
* It’s the simplest architecture patters that achieves 🡺 design maintainability clean design/testable [test each layer separately]
* Separating layers allowed them to be re-usable [using the same DAL and BLL with different PL 🡺 IKEA example web/mobile/dashboard PL projects with the same DAL and BLL
* To test the business for example we need to test the logic of creating order, no need to go through the whole cycle and go through the security issues to test it … just test the functionality of creating the order in the BBL
* 3-Layer architecture patter has disadvantages with large business projects that has third-parties 🡺 discuss those problems in the web API project

**Creating the 3-layer Project:**

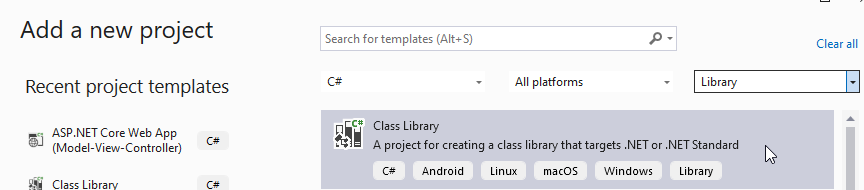
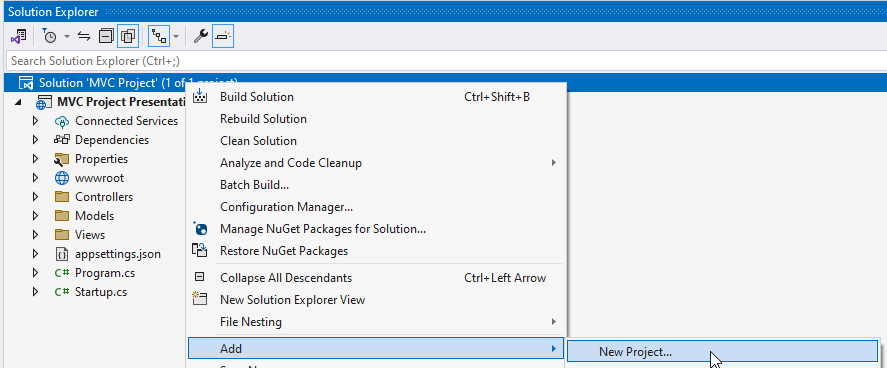
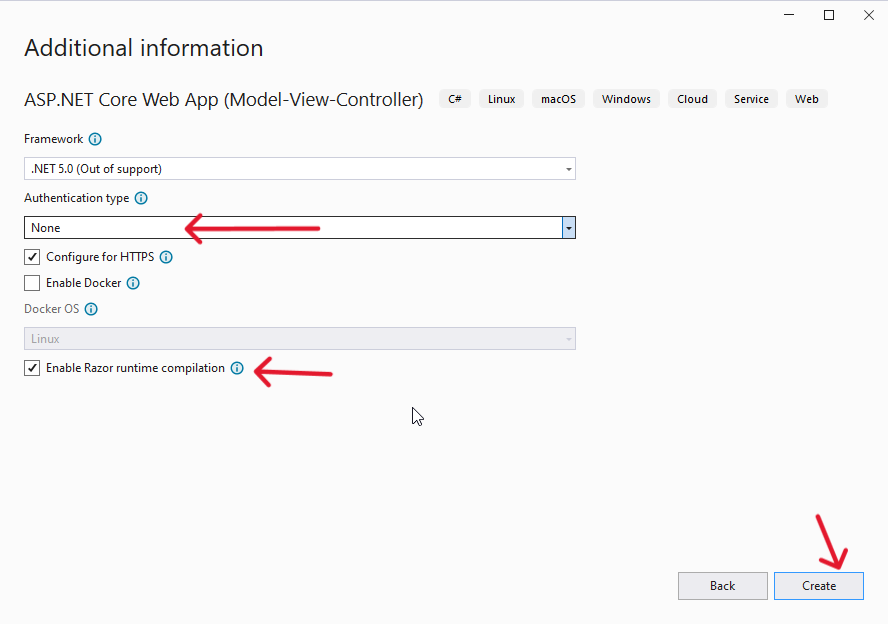
* **We need the 3 project[3-tiers/layers] to deal with each other 🡺 the DAL deal with database 🡺 BLL deals with DAL 🡺 add project reference from DAL to BLL, BLL deals with PL 🡺 add project reference from BLL to PL**
* **NOTE: any package we install in DAL will be seen in BLL and PL=> any installed package will be seen in that layer and the layer taking reference from it 🡺 so PL can see DAL through BL**



The security module is implemented in the .net in the razor pages [sign-in/sign-up/forget password/confirm account /reset password …] if you want to include the security module select Individual Accounts in Authentication Type drop down list 🡺 will make area[asp.net core applications are divided to areas] each area contains group of controllers [area is something like schema in DB/ like module in angular] 🡺 so it will create Security area in razor pages … you will also need to customize this security area

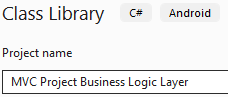
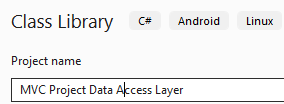


We have razor runtime compilation option 🡺 the razor page [view/partial view/ layout] enables the compilation in the runtime which means the changes [in the development mode]will be appear live after editing in the code and saving it … no need to restart debug

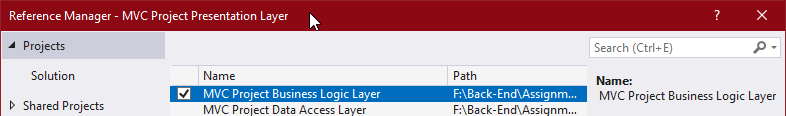
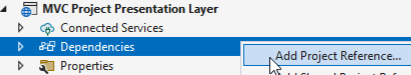
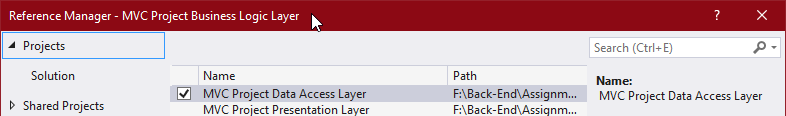
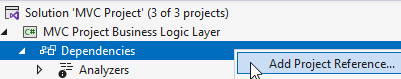


Add 2 library projects 🡺 one for DAL and the other for BLL

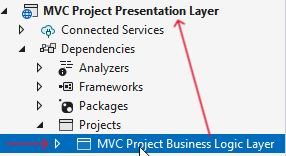
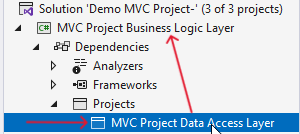
&



* Add reference from data access layer to the business access layer and add reference from the business access layer to the presentation access layer



* Each installed package in one layer is seen in the other layers
* Add reference then build the solution so that the changes in packages takes place



* We are going to divide our project as modules 🡺 start with Department module
* Each module has more than one entity 🡺 there are relationships between entities
* Our work flow starts from DAL 🡺 then BLL 🡺 then PL

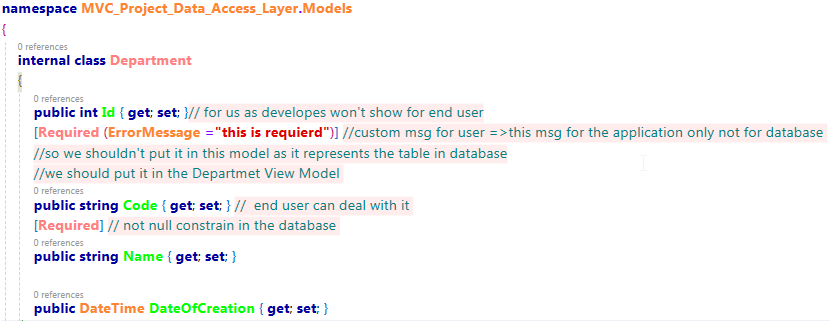
1. **Data Access Layer:**

Is project of type class library

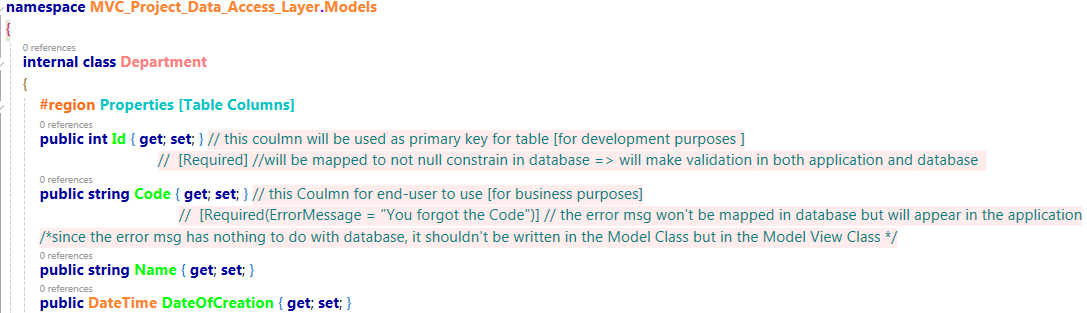
This project is responsible for dealing with database

Has 2 folders

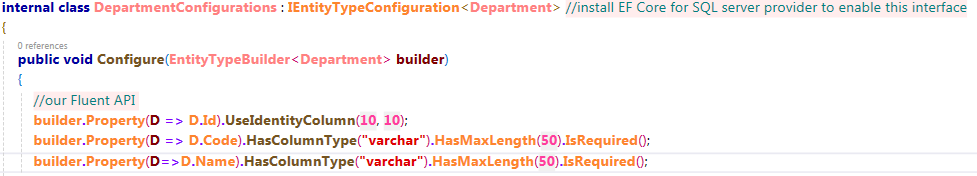
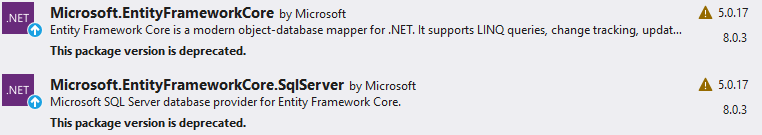
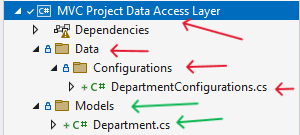
* Data: contains DbContext class/Configuration classes/ Migrations
* Models/domains/entities: contains the domain models
* Create model Department



Any validations mapped in database we prefer writing them with fluent API, so the [Required] data annotation will be removed from the model class

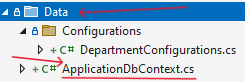


Creating the configuration class for the fluent API validations 🡺 new folder [Configurations] in the Data folder 🡺 DepartmentConfigurations class 🡺 implements the generic interface IEntityTypeConfiguration<Department> 🡺 remember to use the name space of the Department Model & install the package Microsoft.Entity.FrameworkCore.SQLServer and use the namespace 🡺 Microsoft.EntityFrameworkCore

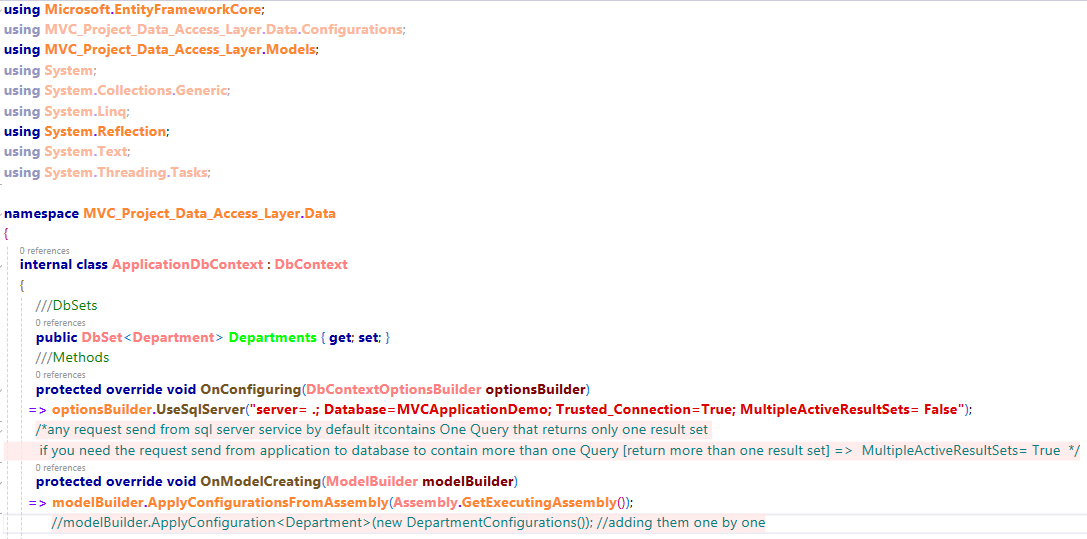


Applying the configurations in the OnModelCreating in the DbContext class

* Creating the DbContext class in the Data folder directly



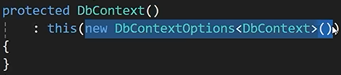
this method using reflection [using metadata] it gets all classes that implements IEntityTypeConfiguration. The ExecutingAssembly is the assembly we are working on [data access layer]



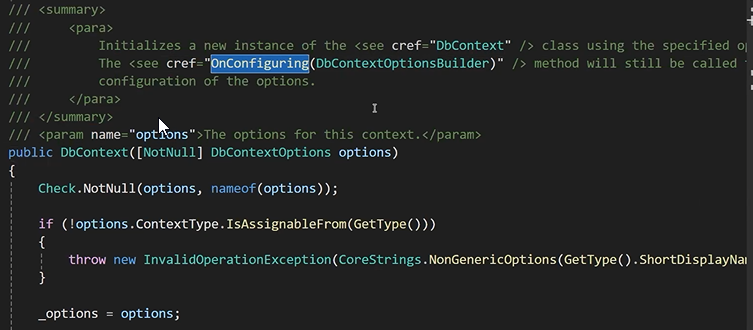
**DbContext – Dependency Injection: [see it again]**

Anyone needs to create object [we need object to deal with database] from class ApplicationDbContext we create by the auto generated parameter-less constructor 🡺 and any child class by default chain on the parameter-less constructor of the parent class

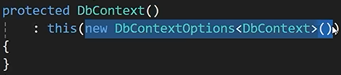
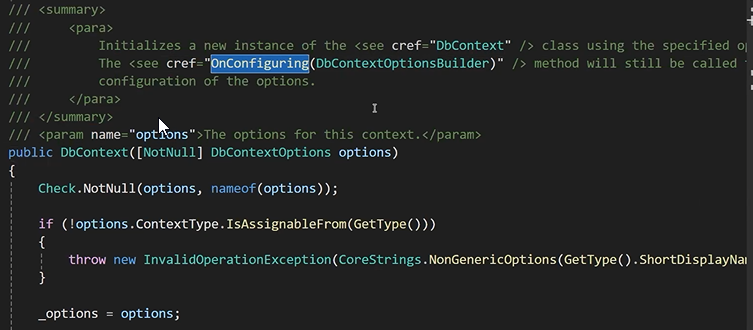
Parent parameter-less constructor also chains on another constructor from the same parent class takes object from DbContextOptions generic class from type DbContext

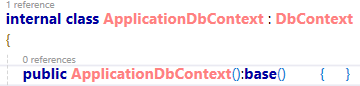


This other constructor that the parent chains on, takes object from DbContextOptions🡺 uses the OnConfiguring to see the connection string and connect on the SQL Server Service



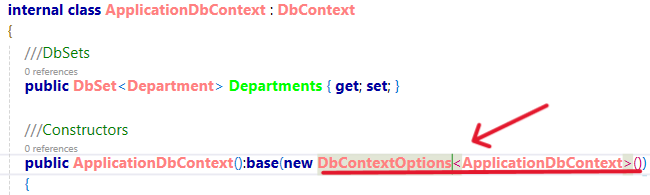
So, our ApplicationDbContext class constructor 🡺 chains on an empty parameter-less parent constructor 🡺 that chains on another constructor in the same DbContext[parent] class and takes object

🡺  🡺 

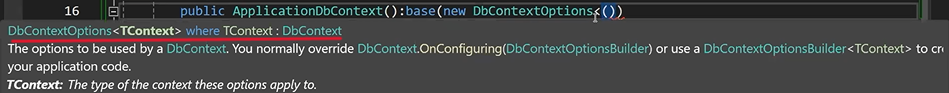
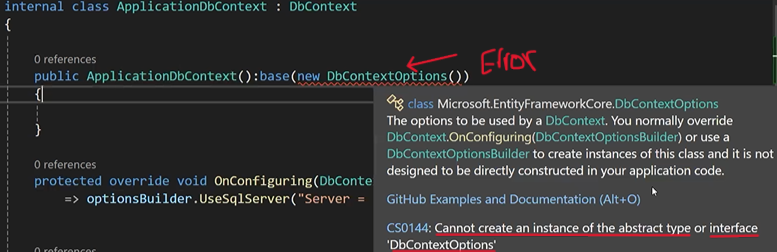


Why not we chain directly on that constructor that takes object from class DbContextOptions <DbContext>

The constructor needs object from DbContextOptions [the options of the DbContext not any options] … because in the OnConfiguring method it takes parameter of type DbContextOptionsBuilder which builds the DbContextOptions of the DbContext [ApplicationDbContext] 🡺 that’s why we send to the constructor object from DbContextOptions<ApplicationDbContext>

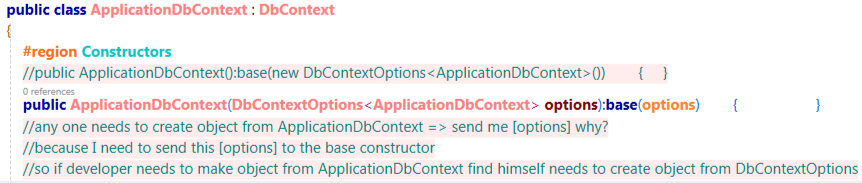


Why we don’t just send object from DbContextOptions without sending the type <ApplicationDbContext>? Why it gives us error? As the DbContextOptions is abstract class



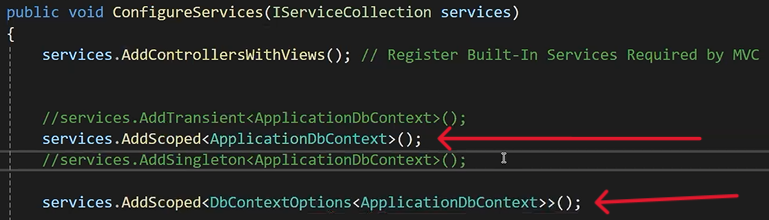
Note: creating the object in the constructor that way will lead to creating object each time we need object 🡺 the DbContextOptions object won’t change in each time we create new one [as the connection string we send is the same each time] so we don’t need to create new object each time we need object, instead use the same object created 1st time, creating objects in that way is not the optimal solution, we’d better let the CLR deal with creating objects and do the work behind creating this objects like creating other objects depends on them 🡺 so we use Dependency Injection in case we need an abject while application is running and tell the CLR what we’d like the object lifetime to be

For the CLR to create object from class ApplicationDbContext it will need to create object from class DbContextOptions



To allow the CLR to create us object from this DbContextOptions<ApplicationDbContext> we need to register this class in the services container [ConfigureServices method in the Startup]

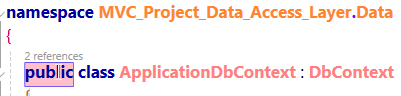
Allow DI for ApplicationDbContext 🡺 in the presentation layer 🡺 Startup class 🡺



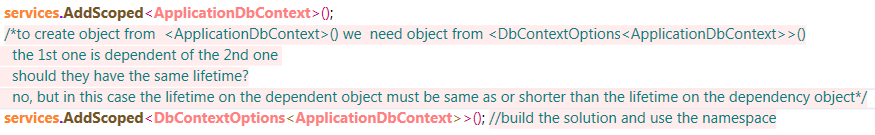
Note that class DbContextOptions<ApplicationDbContext> is in the namespace of Microsoft.EntityFrameworkCore which we installed its package in the DAL so we need to build the solution so that the PL can see that package



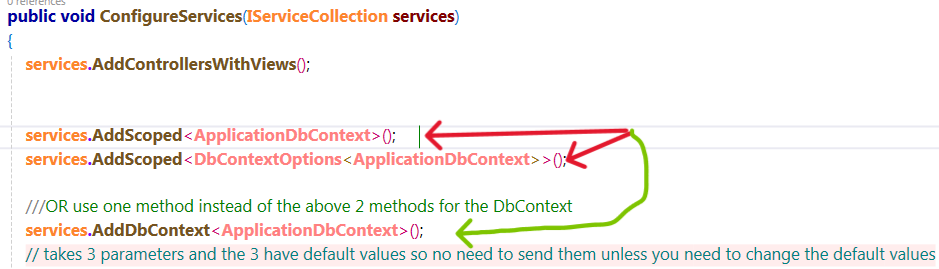
Remember to make class ApplicationDbContext public so that we can use it in the dependency injection in the Startup class

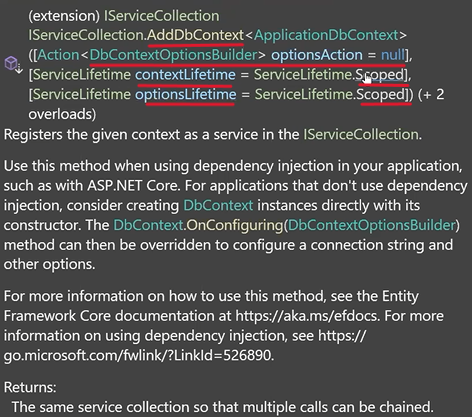


Allow 2 services in the DI container🡺

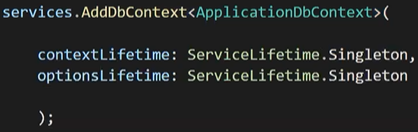


One Service instead of 2🡺

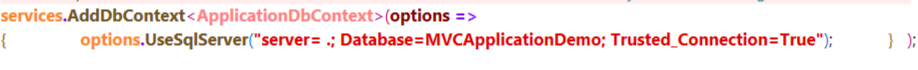
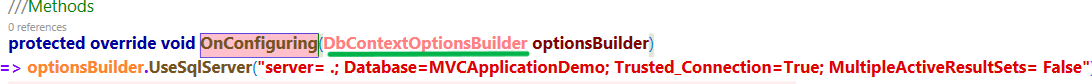




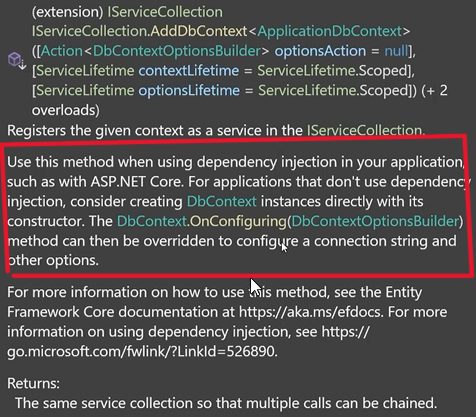
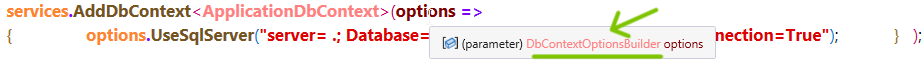
* To change the default values of the parameter send them by name 🡺



* The 1st parameter action void method DbContextOptionsBuilder which is in the ApplicationDbContext class in the OnConfiguring method 🡺 which means we can send the connection string in this method instead of overriding the OnConfiguring method

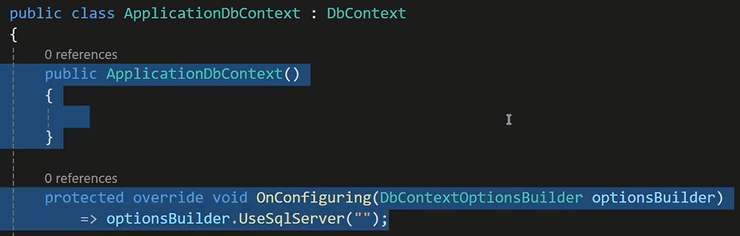


* What happens is when someone needs object from class ApplicationDbContext 🡺 CLR will find out we are allowing dependency injection for ApplicationDbContext so CLR will create object from it with Scoped life time [the default, we didn’t change it in the dependency injection AddDbContext method] 🡺 creating object from ApplicationDbContext depends on dependency injection from object DbContextOptions[remember the Constructor] which we also allowed its dependency injection service through the same method [AddDbContext] 🡺 CLR will create object from class DbContextOptions with lifetime Scoped 🡺 and when creating object from DbContextOptions it uses options-builder 🡺 so now we configured the OptionsBuilder no need to configure it in the OnConfiguring method again



So we have 2 ways one is for the dependency injection way and the other normal way

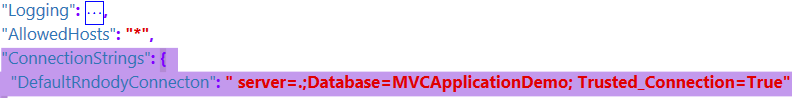
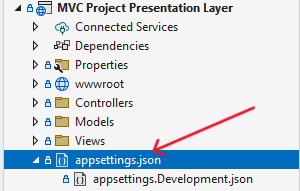
🡺normal way , we used it in the console application as in console we don’t have dependency injection



**App Settings – Connection String:**

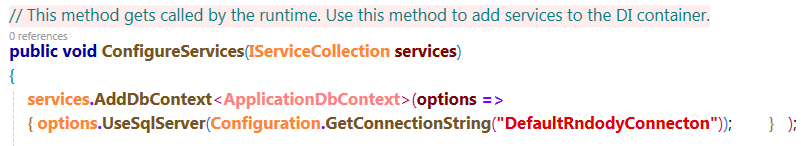
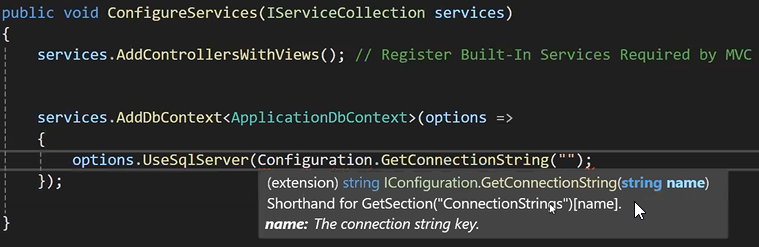
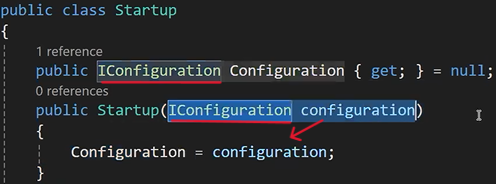
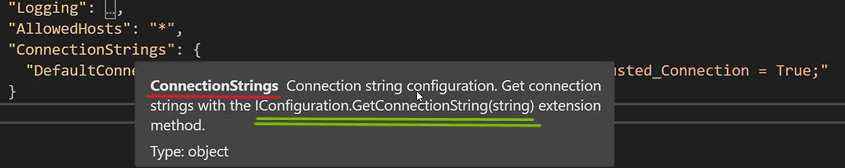
Writing the connection string in source code is not valid as it changes from environment to another and it’s available for anyone to see it and it should be encrypted as well

We write the connection string in that file



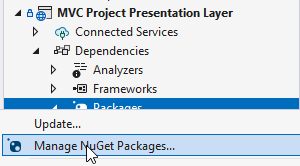
Now we need in the Startup class to get the Connection String from the appsetting file, how? If you want to get the connection string 🡺 use object from class implements the IConfiguration interface inside it method GetConnectionString

in the Startup class we have read-only property Configuration from type IConfiguration use it to add the connection string in the .AddDbContext<ApplicationDbContext> method in the action 1st parameter give it the name of the connection🡺 remember when creating new object from class Startup the new keyword will give the reference type objects null value, so this property is equal null, it takes value in the Constructor 🡺 what happen in the constructor? Dependency Injection 🡺 when ? when we call constructor🡺 when? When we create object from class Startup 🡺 in the Program class in the Main function [entry point] we create object Do we allow the dependency injection of the IConfiguration? Yes we do🡺 in the AddControllersWithViews method



**Generate Migration:**

We need to run the migration on the project which has appsetting as it has the connection string so we will add the package in that project [MVC Project /presentation layer]



Installing package on the startup project [MVC/presentation layer] and adding migration on the data access layer

The Migrations Folder will be in the DAL in the Data folder create Migrations folder🡺 the -context parameter if we have more than DbContext class we use it to mention which one we need, the –Output parameter is for specifying the place of the Migration folder , use it only the 1st migration then it will know it by default



1. **Business Logic Layer**

Is project of type class library

In this layer we use 2 design patterns

* Generic repository design pattern [for each domain 🡺 repository] that repository helps to deal with database tables [product repository deals with data access layer that deals with product table in DB]
* Unite of work

Create 2 folders in the business logic layer project [Interfaces & Repositories]

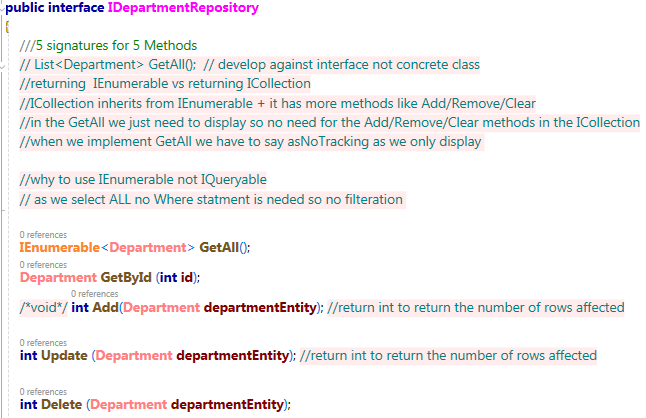
Repository design pattern 🡺 each domain has repository, through that repository we can add/update/delete/get by id / get all entities

Repositories is service contain crud operations or behaviors for table it is related to

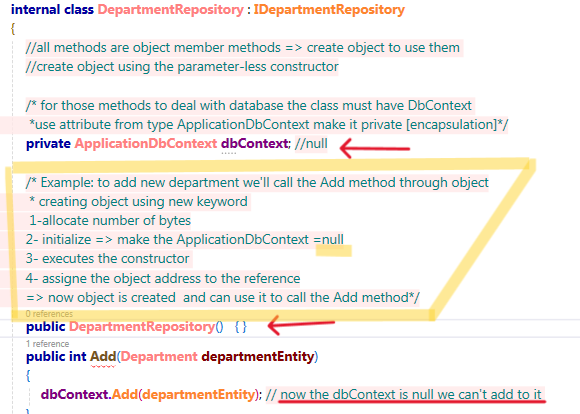
Repository deal with table using DbContext class

We don’t deal with DbContext directly because we are in the BLL not the DAL so for each DbSet we make repository

In the Interfaces folder we add new contract for each repository

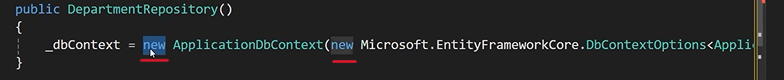


In the Repositories folder we add repository class for each DbSet and let it implements the interface related to its type [Note: we are not working with Generic Type till now]



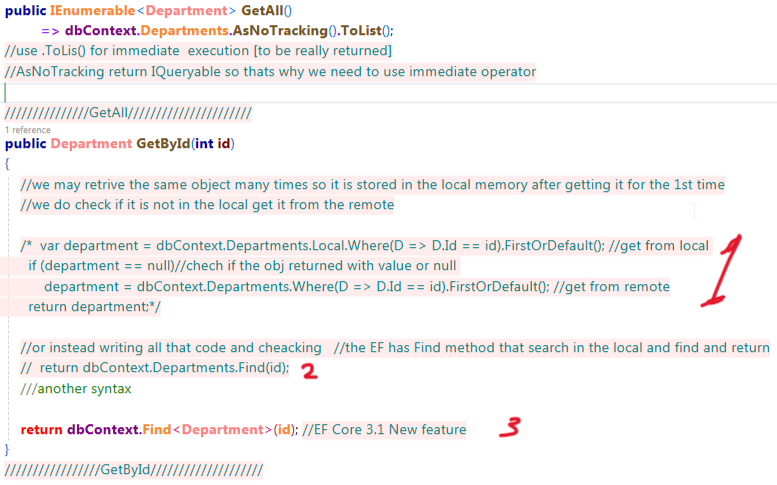
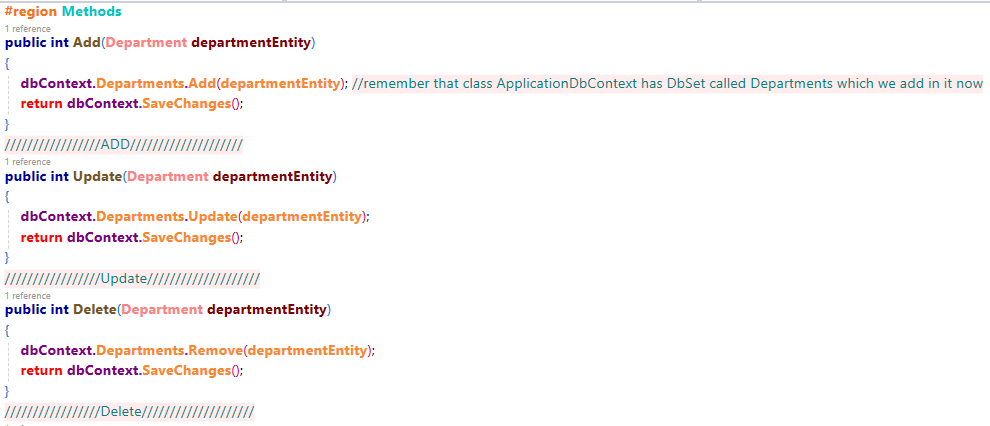
Since the methods are object members so we use them through object🡺 to get object the constructor is executed 🡺 so let the constructor initialize object and send it to the dbContect attribute we have so that each time we call the constructor we are sure that we have object from class ApplicationDbContext.

But remember that creating object from ApplicationDbContext class depends on another object from DbContextOptions

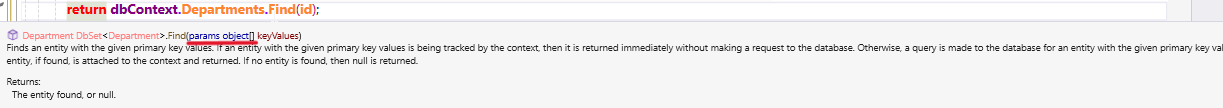


As we mentioned before, creating objects this way is not preferred as if we created more than one object from DepartmentRepository class in one request it will open more than one connection which is unneeded

To solve the dependency problem let the CLR create us the object we need in the constructor by adding dependency injection in the constructor parameter … remember we allowed the dependency injection for the ApplicationDbContext class in the DI container so we can use it here directly



The Find method takes the primary key as array … because the PK may be composite key example how it looks🡺



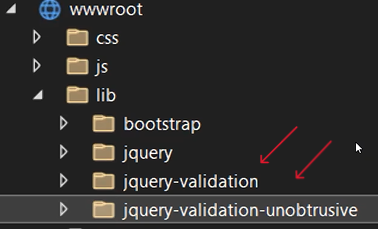
Or send them in anonymous type if you are not sure of the order on the values send not sure of this way try it



1. **Presentation Layer – MVC Template**

* Can be more than one layer/project
* Server Side Application MVC
* Server Side Application Razor Pages
* Web API
* Single page application - Blazor or Server Side Application – Blazor
* Desktop –WPF
* Mobile – Xamrin or MAUI
* Now the MVC Model View Controller:
* the Model is in the Data Access Layer
* the Controllers that contains the endpoints/actions that contains the business logic 🡺 is abstracted and added to the Business Logic Layer
* remaining the end point that will call the business in the BLL and the View

we use JQuery files for frontend validation



**Department Controller – Dependency Injection:**

**Department Controller – Index**

**Department Controller - Create**