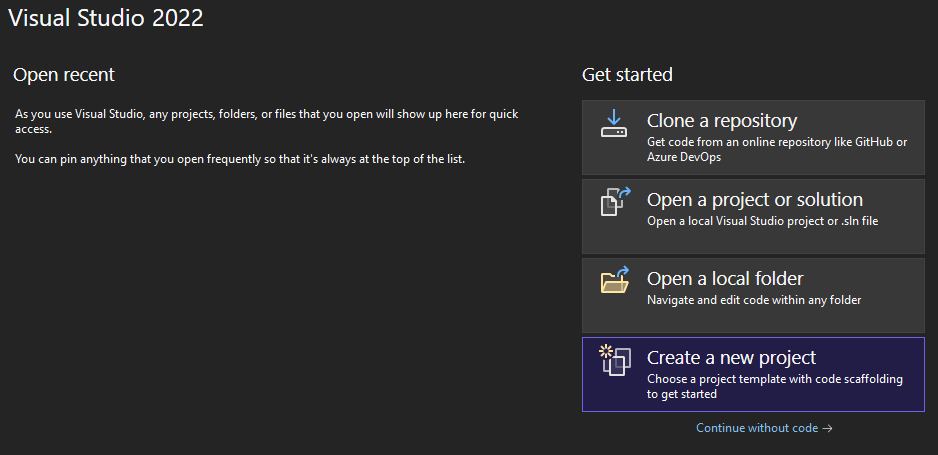
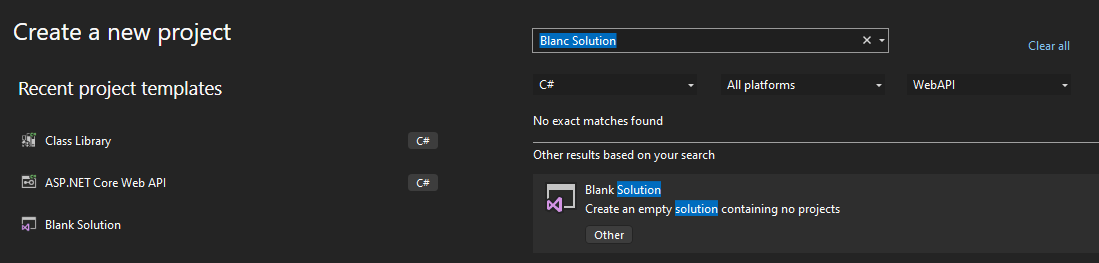
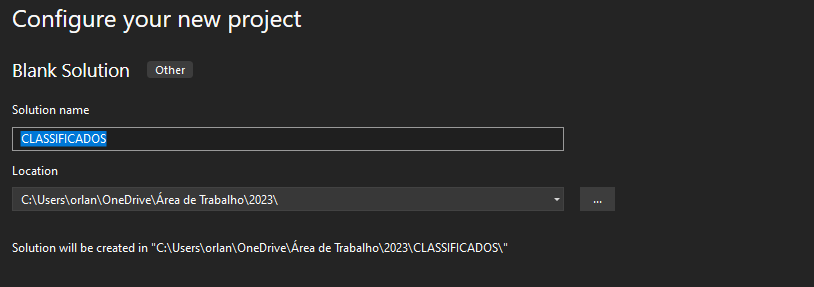
**Criação da App de Classificados**

**1 – Setup**

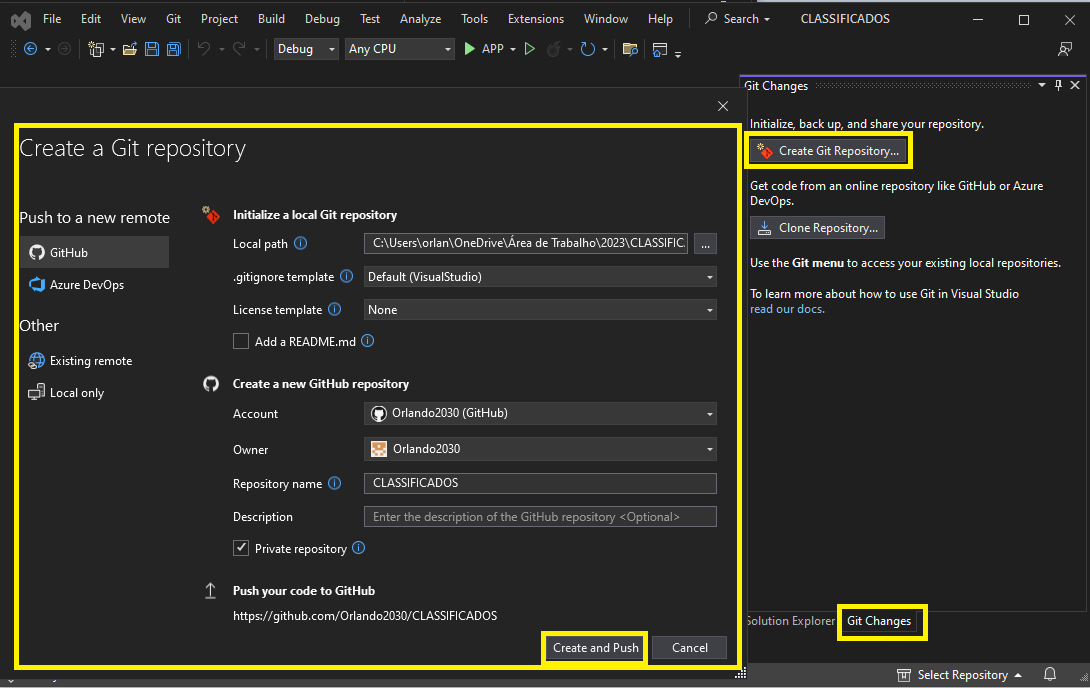
* 1. - Criar uma “Blank Solution”

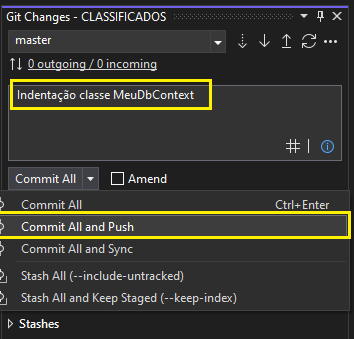


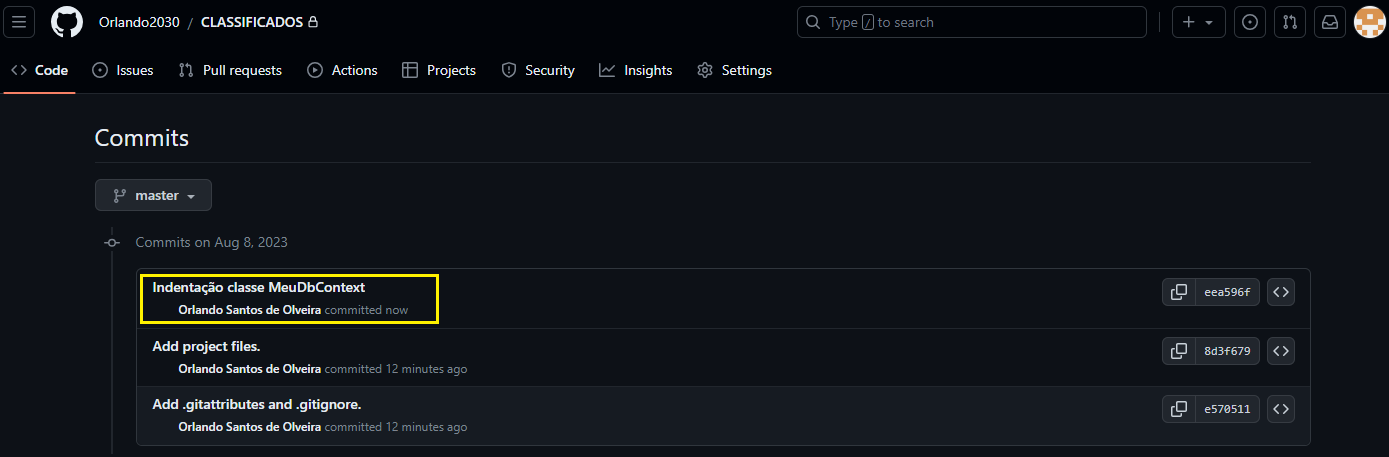




* 1. – Criação de repositorio Git





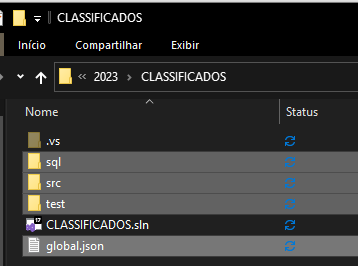


**git push:** Realiza o upload do conteúdo do repositório local para o remoto.

**git sync:** Primeiro executa um git pull que baixa o conteúdo do repositório remoto e atualiza o local.

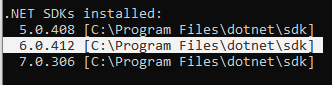
**git commit**: Permite que você crie um commit, ou seja, você consegue guardar o estado do seu repositório naquele momento.

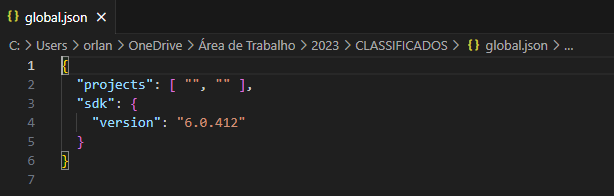
* 1. - Criar as pastas (sql, src, tests) na pasta da Solution.



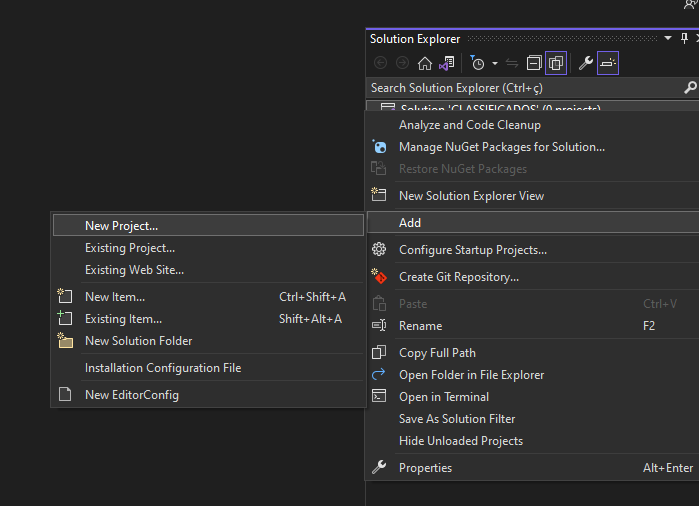
1.3 - Criar o arquivo “global.json” (Define qual versão do seu SDK vi ser utilizado)

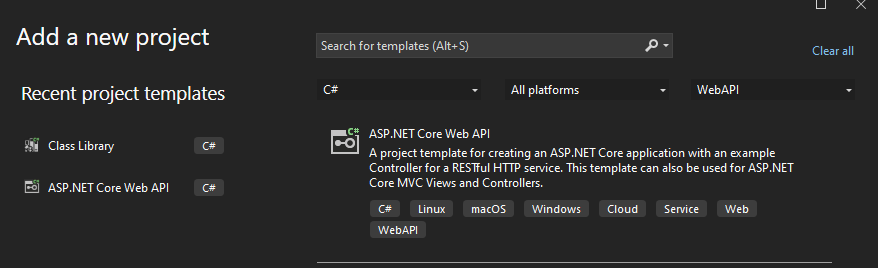
Verificando a versão do SDK instalado na maquina no CMD com o comando **dotnet - - info**

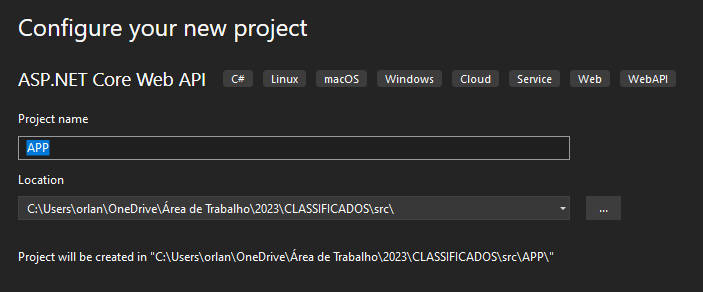


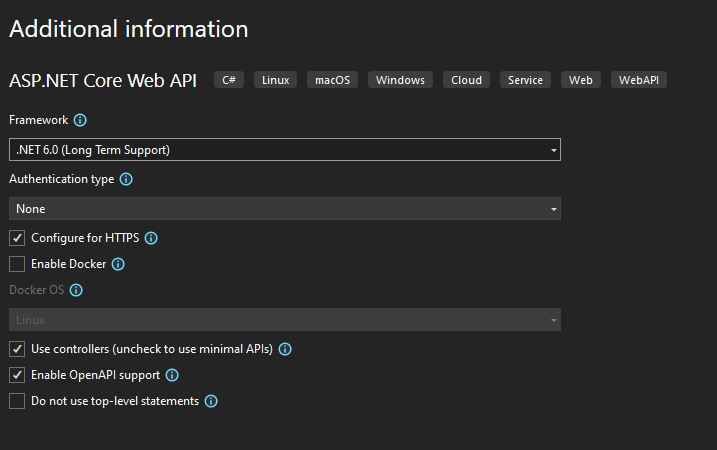


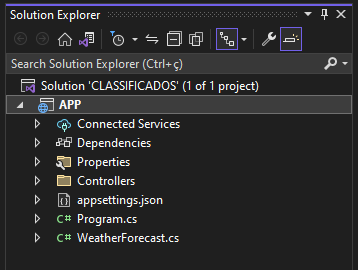
* 1. – (CAMADA DE APRESENTAÇÂO “App”) Add um novo projeto na Solution (ASP.NET Core Web Aplicattion (MVC)) dentro da pasta src



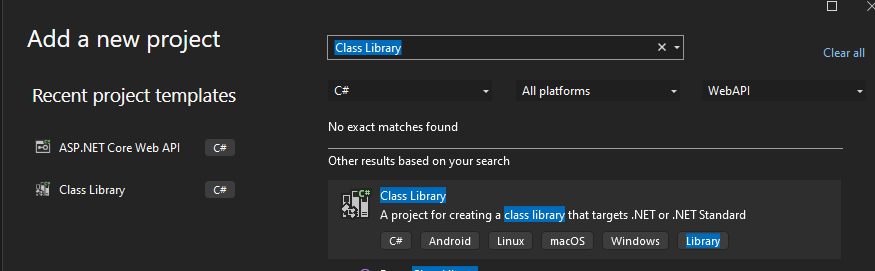


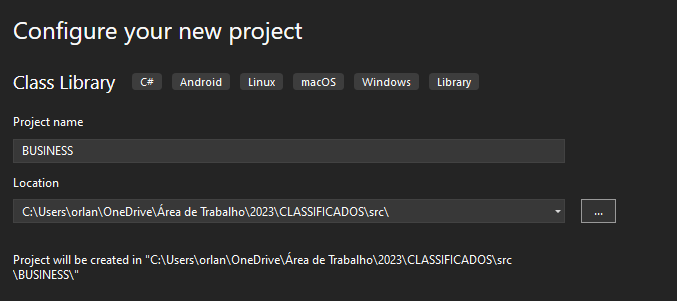


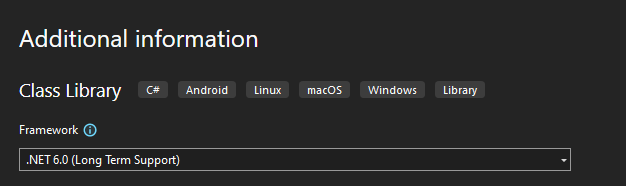


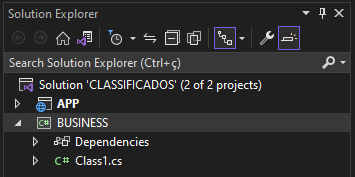


* 1. – (CAMADA DE NEGOCIO(Business) “Business”) Add um novo projeto na Solution (Class Library .Net Core) dentro da pasta src



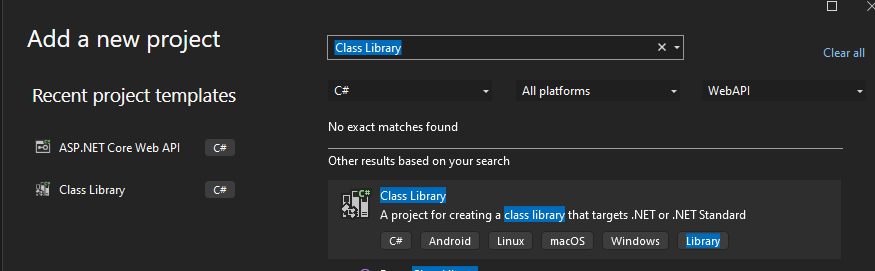


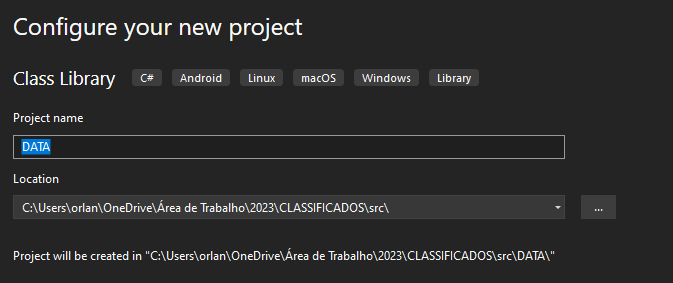


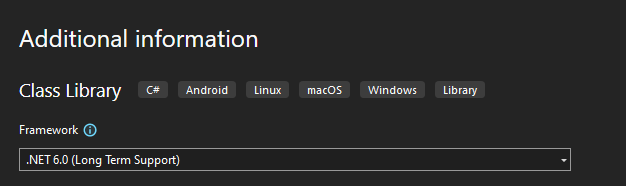


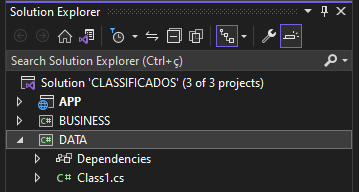
**Pode deletar o Class1.cs**

1.6 – (CAMADA DE DADOS(Data) “Data”)Add um novo projeto na Solution (Class Library .Net Core) dentro da pasta src

****

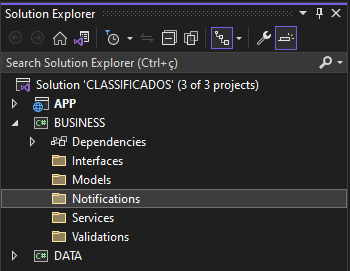
****

****

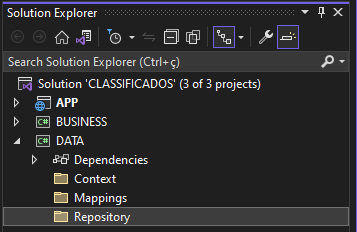


**Pode deletar o Class1.cs**

1.7 – Criar as pastas (Interfaces, Models, Notifications, Services, Validations) dentro da camada (Business)

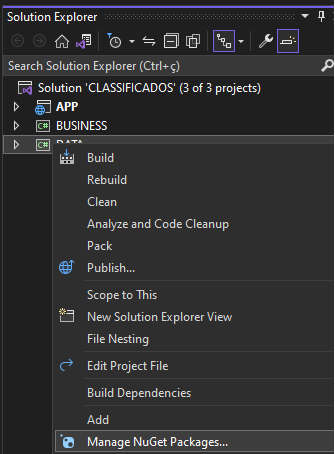


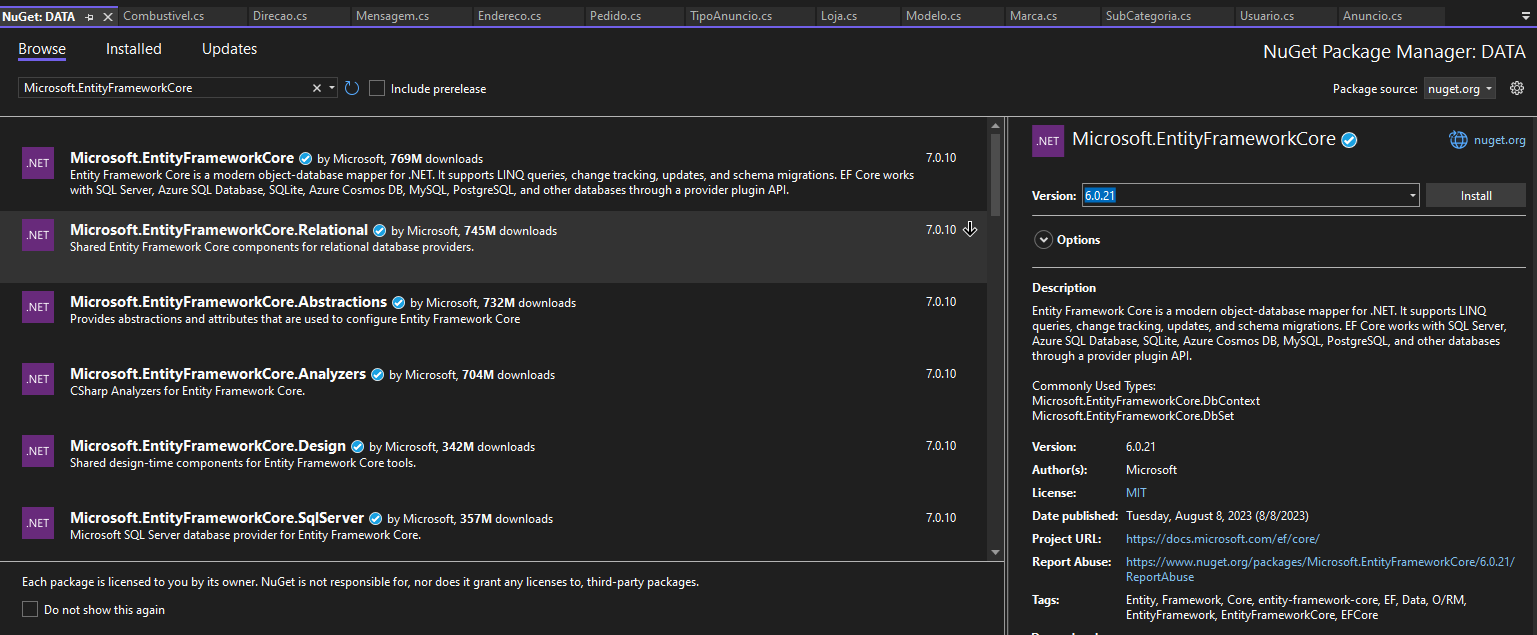
1.8 – Criar as pastas (Context, Mappings, Repository) dentro da camada (Data)

****

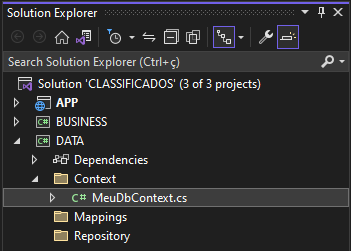
## **– Criação e Mapeando das Entidades**

* 1. – Criação das classes dentro (Business\Models)
  2. - Executar o comando Install-Package Microsoft.EntityFrameworkCore na camada “Date”

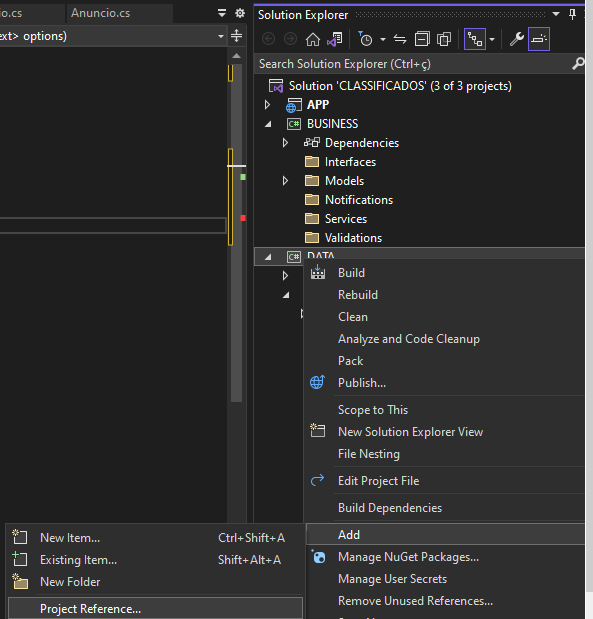


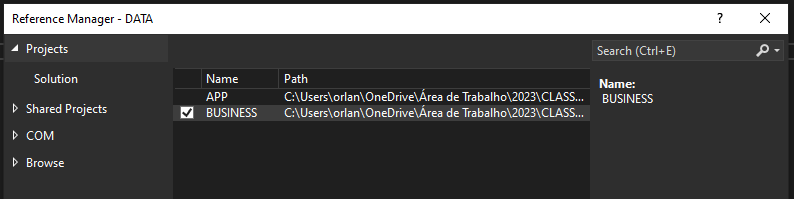


* 1. - Criar a classe “MeuDbContext” na pasta Context na camada Date

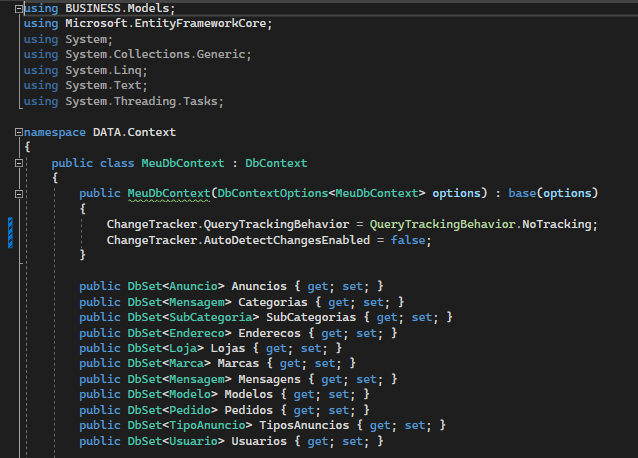


2.4 - Add na camada “DATA” as da camada de “Business”





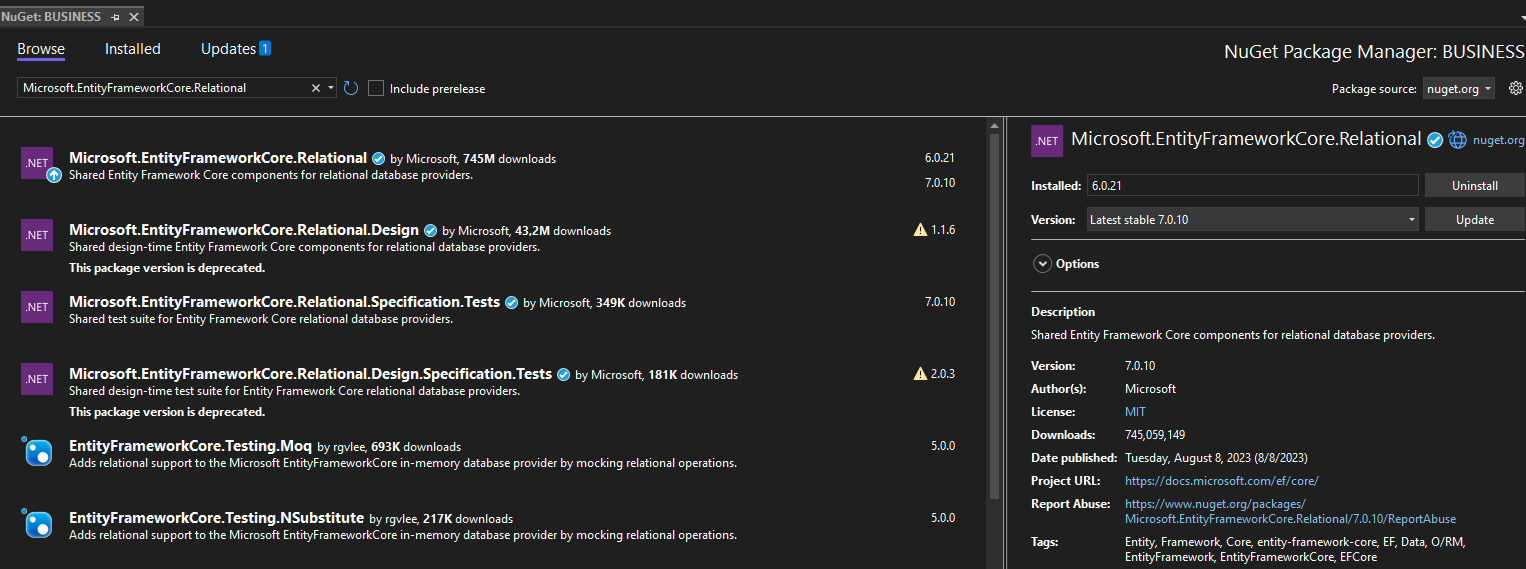
2.5 - Após a instalação é possível relaciona a classe MeuDebContext “: DbContext” e criar um .construtor (Public MeuDbContext(DbContextOptions options) : base(options)).

Nas versões 2.0 e 3.0 não era necessario adcionar os **ChangeTracker**.

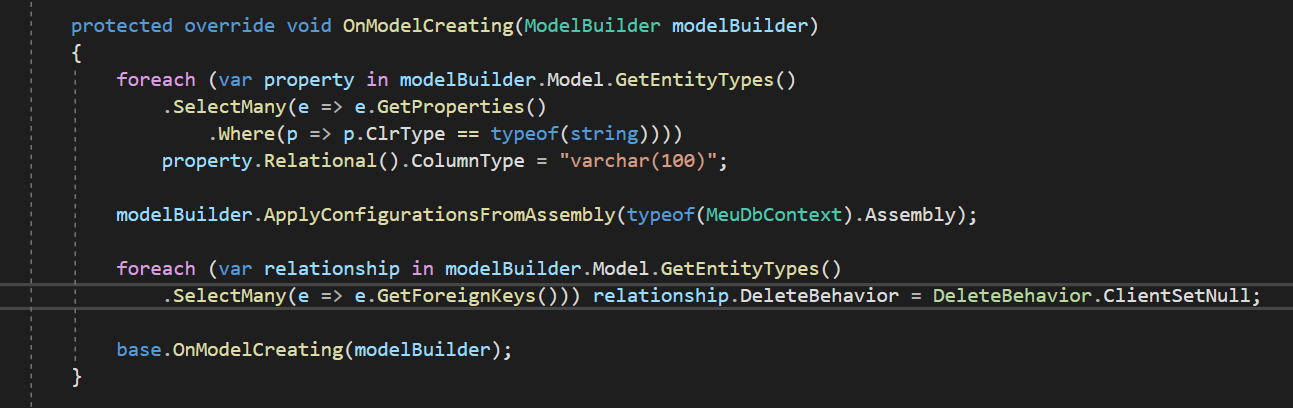
2.6 - Criar a classe “**Model**Mapping” e herda de : IEntityTypeConfiguration<**Model**>.

Dentro da classe é necessário implementar a interfece “public void Configure(EntityTypeBuilder<Produto> builder)” e dentro é feito o mapeamento dos campos e seus relacionamentos

**Obs.: é importante instalar o Microsoft.EntityFrameworkCore.Relational**

****

2.7 – Adicionando na class MeuDbContext



modelBuilder.ApplyConfigurationsFromAssembly(typeof(MeuDbContext).Assembly);

2.8 – Desabilitação do cascade delete (Isso impe que ao excluir, exclua seus filhos juntos)

foreach (var relationship in modelBuilder.Model.GetEntityTypes()

.SelectMany(e => e.GetForeignKeys())) relationship.DeleteBehavior = DeleteBehavior.ClientSetNull;

2.9 – Garantir que as colunas do tipo string esquecidas de ser mapeadas fique varchar(100) e não varchar(MAX)

foreach (var property in modelBuilder.Model.GetEntityTypes()

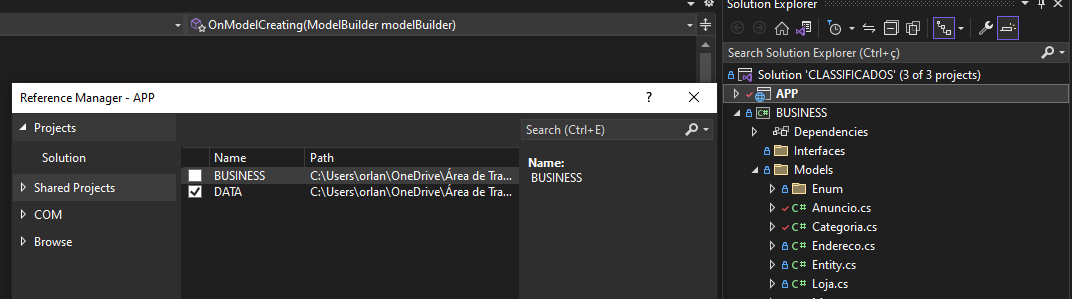
.SelectMany(e => e.GetProperties()

.Where(p => p.ClrType == typeof(string))))

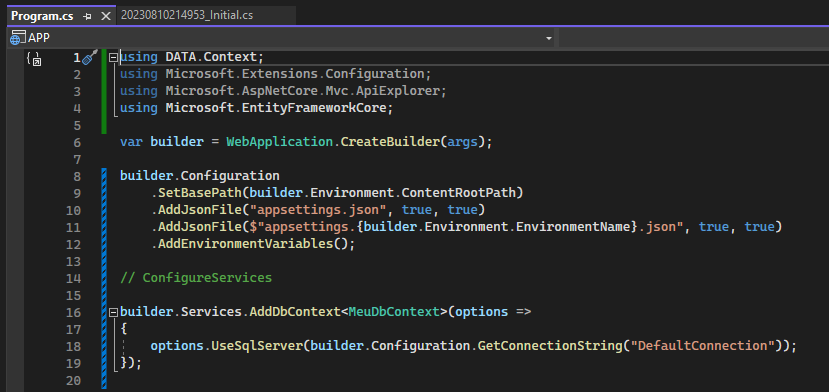
property.Relational().ColumnType="varchar(100)";

2.10 – Configuração do DbContext na aplicação

Ir nas dependencies no App e add a reference Data.



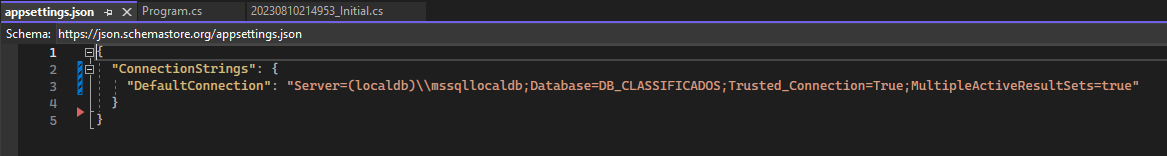
2.11 – Depois ir no Program.cs e add um novo **services.AddDbContext**



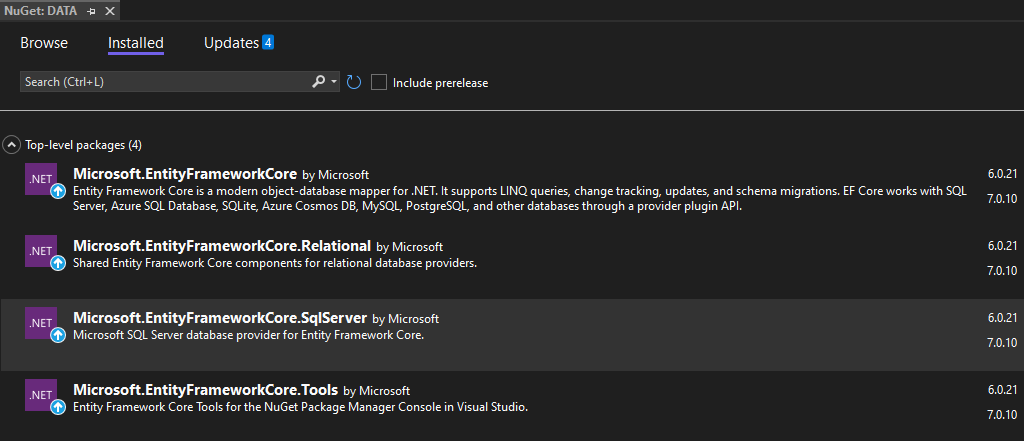
services.AddDbContext<MeuDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

2.12 – Alterar o nome do Database para DB\_CLASSIFICADOS no arquivo appsettings.json da camada **App**

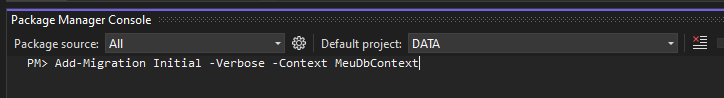


2.13 – É importante intalar na camanda de **Data NuGets**



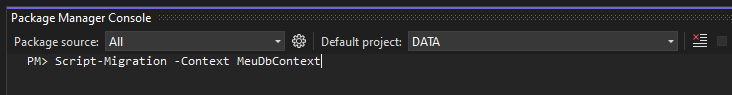
2.14 – Criar uma Migration: (É necessário selecionar a camada (DATA) e só do ser executada em apenas um contexto)

PM> Add-Migration Initial -Verbose -Context MeuDbContext



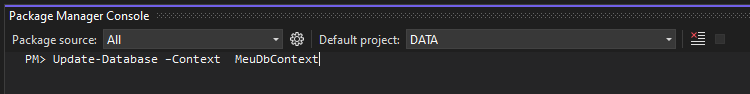
2.15 – Criação dos scripts das tabelas

PM> Script-Migration -Context MeuDbContext



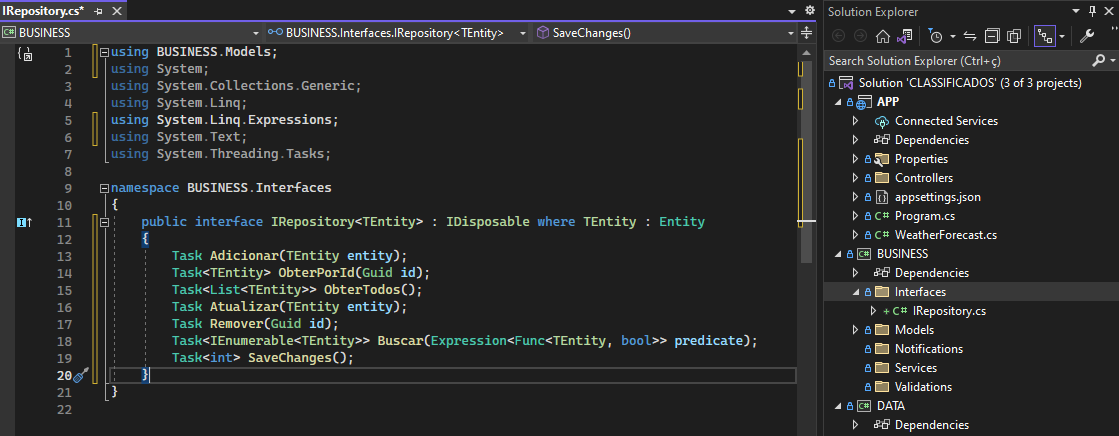
2.16 – Criação da base da aplicação

PM> Update-Database -Context MeuDbContext



## **3 - Acessando o banco via repositórios**

3.1 – Criação da interface **IRepository** dentro Business.Interfaces



public interface IRepository<TEntity> : IDisposable where TEntity : Entity

<TEntity> - Repositorio Genérico

IDisposable – Implementa para liberar a memoria

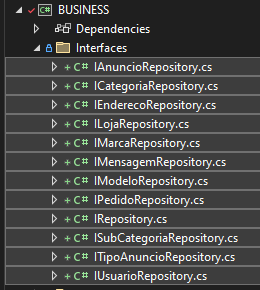
where TEntity : Entity – É especifico par classes filha de Entity

Sempre é bom trabalhar com métodos assíncrono (Task) para garantir a melhor performance e a saúde do servido.

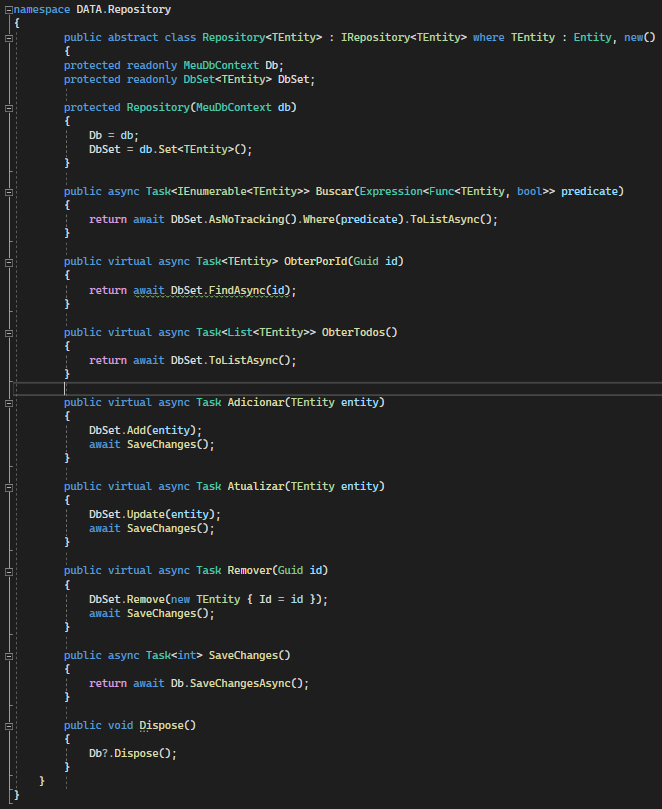
* Criar o método Adicionar
* Criar o método ObterPorId
* Criar o método ObterTodos
* Criar o método Atualizar
* Criar o método REMOVER
* Criar o método Buscar
* Criar o método SaveChanges

O repositório genérico serve para qualquer entidade, ele esta na camada de negocio pq a camada de negócio não vai conhecer a camada de acesso a dados “MeuDbContext”, quem conhece a camada de negocio e a camada de acesso a dados, a única forma da camada de negócio conversa é através dessa interface “**IRepository**” que será injetada por dependência fazendo referência a uma classe de repositório que está na camada de dados.

3.2 – Criação da interface responsável pela classe de Models



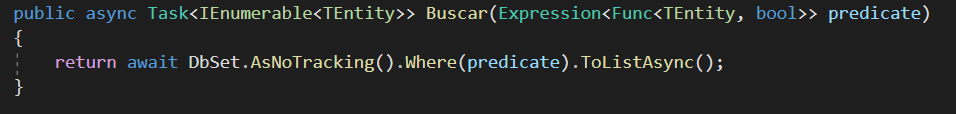
3.3 – Implementar as Interfaces como classes completas, a primeira a ser implementada é Data.Repository é a Repository



Ela é uma classe abstracta só pode ser herdada não pode ser instanciada.

E é necessário implementa a interface IRepository de <TEntity> mas essa é uma filha de **Entity**

É obrigatório implementar todos os métodos desses contratos.

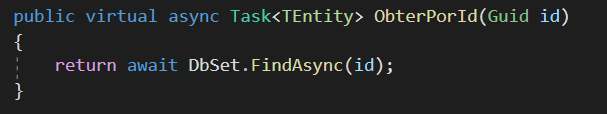


Vamos ter que devolver uma consulta utilizando o predicado.

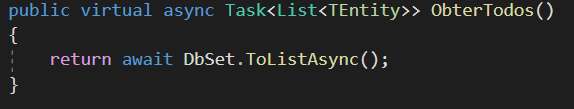
Sempre será um **await** por ser métodos assíncronos

É importante usar o **AsNotracking** pois ele fica acompanhado as mudanças de estados, retorna a resposta com mais performance.

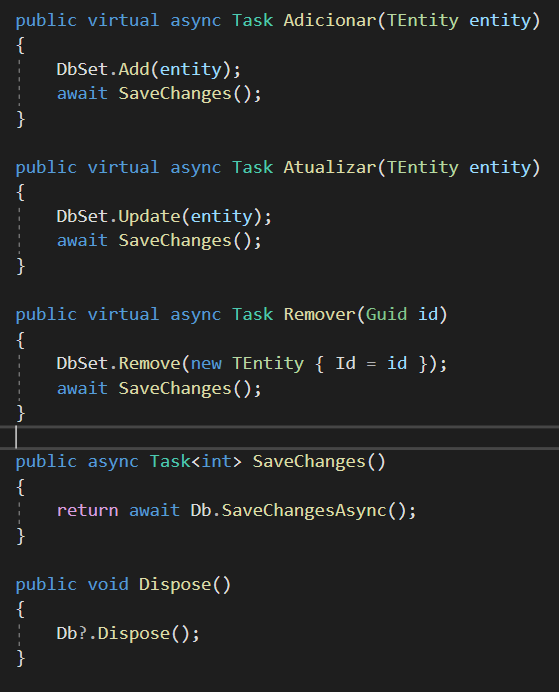
Obs: Vá até o banco de dados busque na entidade onde a expressão que vc passar retorne uma lista de forma assíncrona.



Obter por Id.



Obter todos.

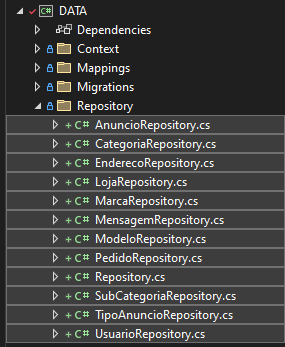


**Virtual** é usada quando você quer em algum momento sobrescrever o método.

3.4 – Implementar as Interfaces **ModelsRepository** na Data.Repository

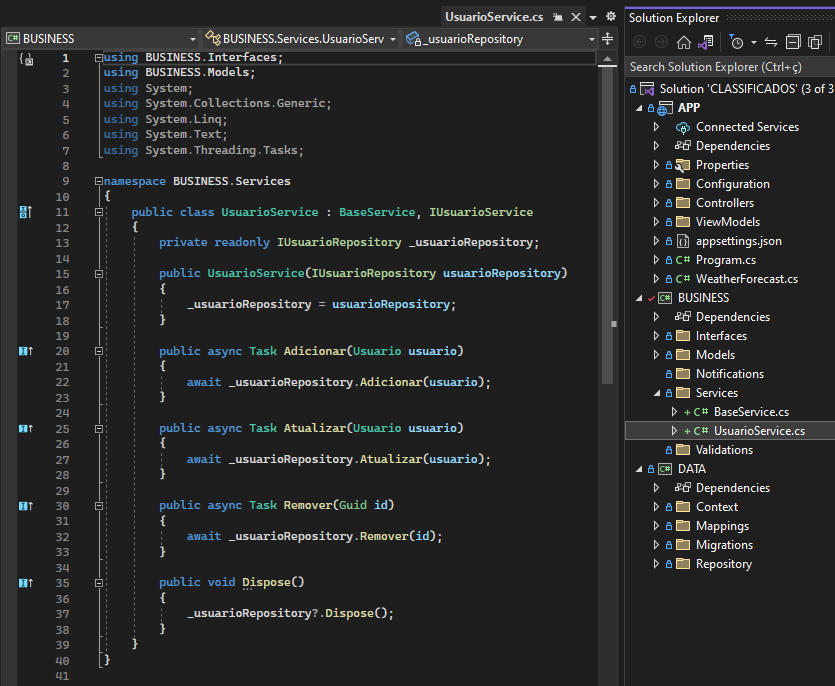
public class **Models**Repository : Repository<**Models**>, I**Models**Repository

É obrigatório implementar todos os métodos desses contratos (I**Models**Repository), mas usamos as **Repository** apenas para utilizar os metodos de consulta (**BuscarTodos**, **BuscarID**)



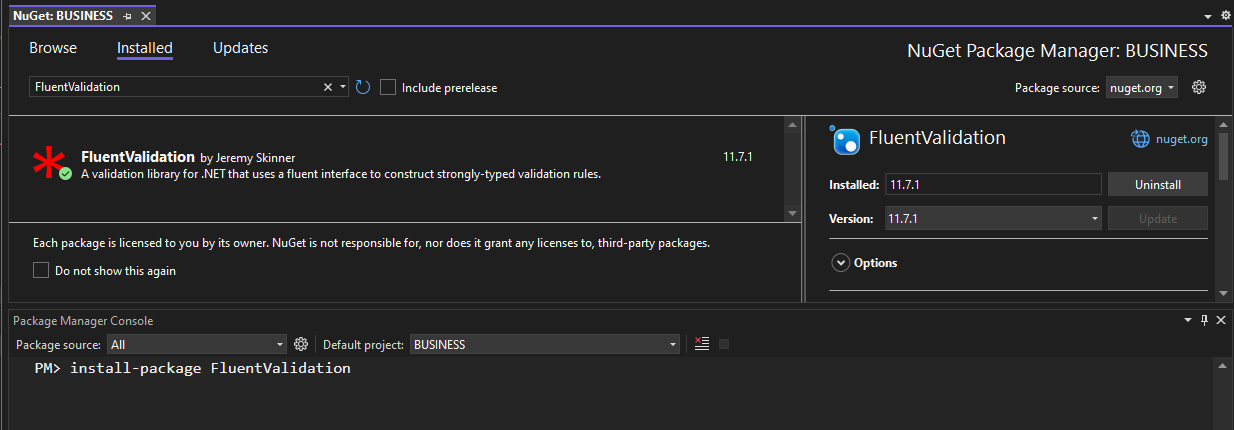
3.5 – Implementar as Interfaces **ModelsServices** na Data.Services

As **ModelsServices.cs** são responsaveis pelas metodos de **Adição, Edição e Deleção, porem a Services deve conter as validações antes de fazer alterações na base.**

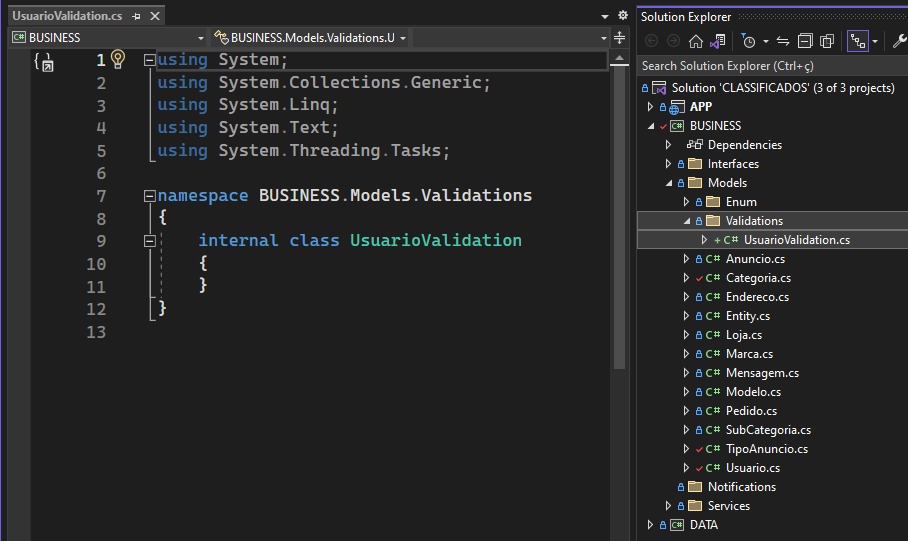


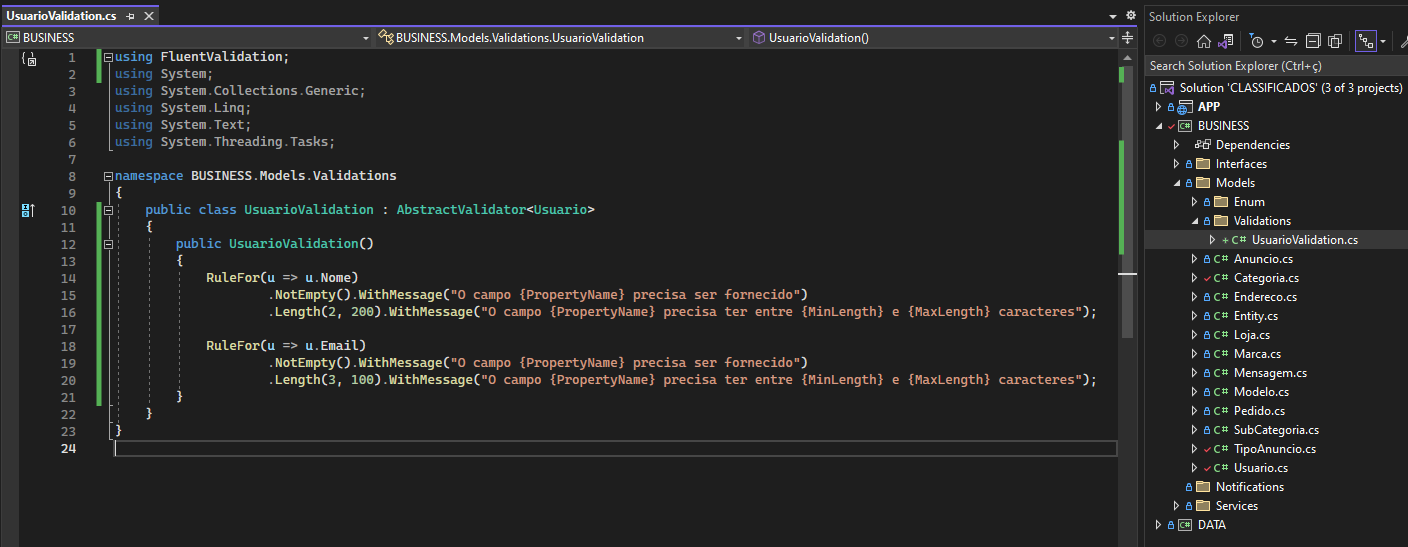
3.6 – Criar as validações das Models.Services.

Instalar o FluentValidation.

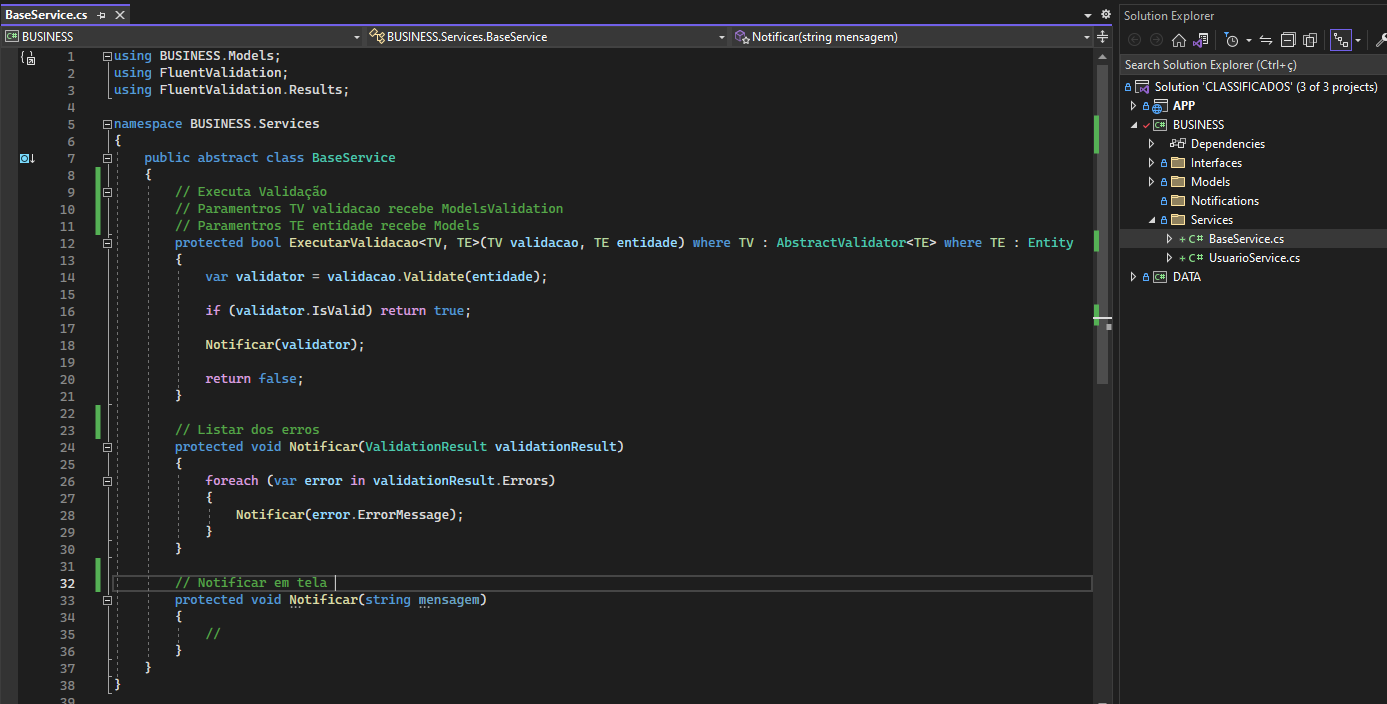


3.7 – Criar a pasta Validation dentro da Business.Data.Validations e suas ModelsValidation.cs com suas validações.





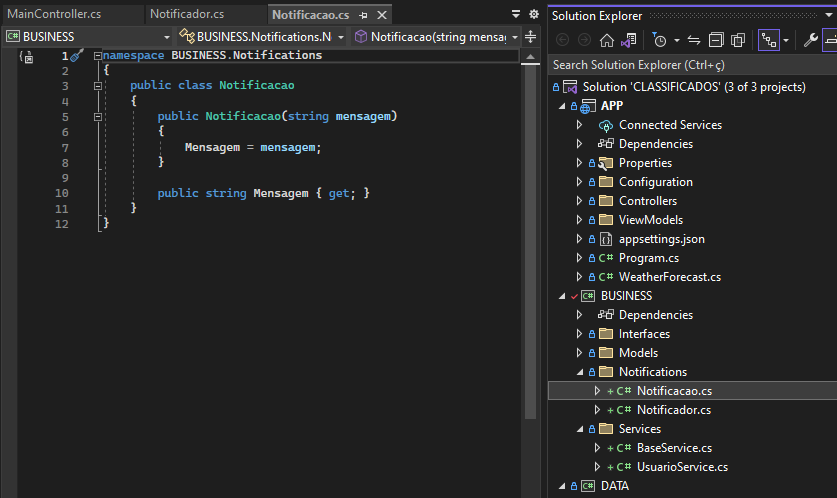
3.8 – Criação da validações dentro da classe **BaseService.cs**.



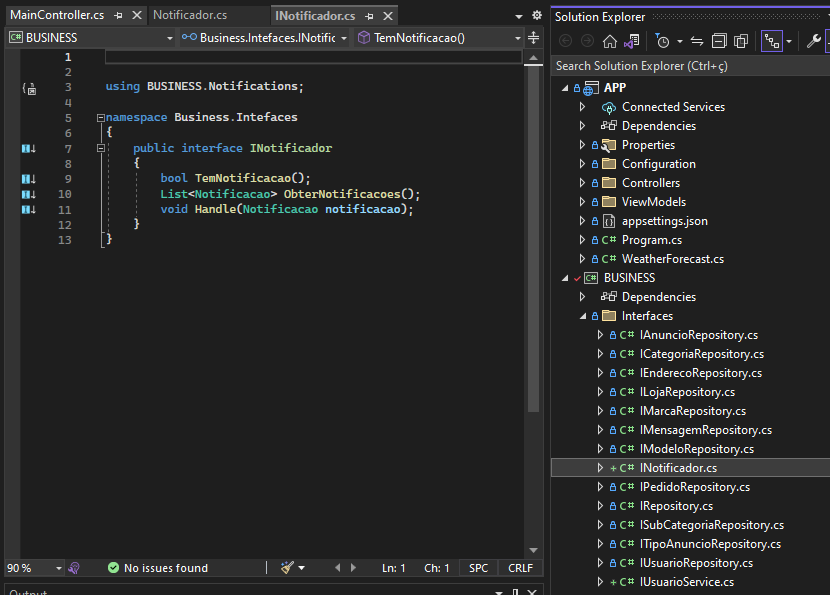
Nesse momento já estamos conseguindo validar as regras nabase de negocio e capturando as msn de erro, porem não estamos conseguindo propragalas, é o que sera feito a seguir.

3.9 – Criação do Notificador.

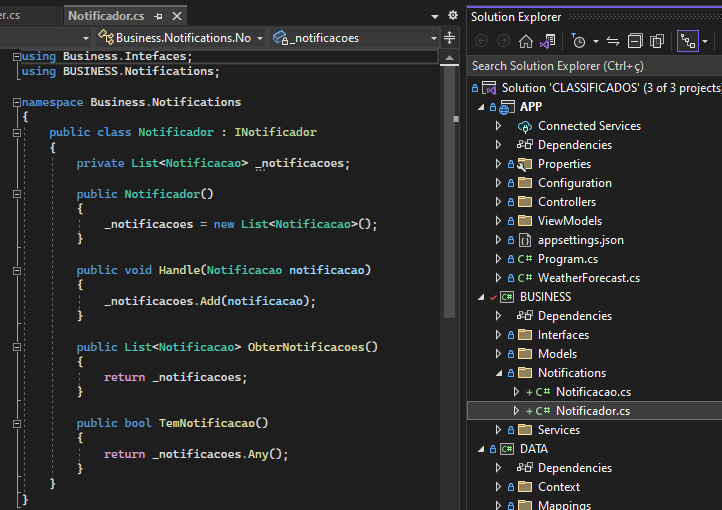
Criar classe Notificação:



Criar a interface Inotivicador.cs



Criar a Classe Notificador.cs

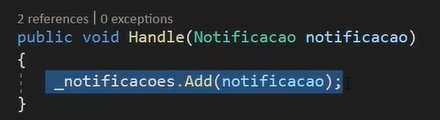


Endendendo:

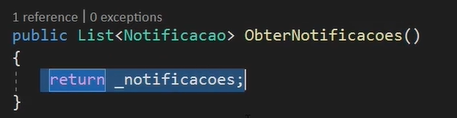
Lista de notificação



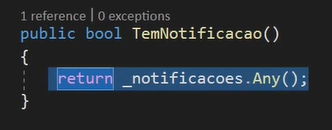
Quando é feito o Handle a adicionado as Notificações dentro da lista.



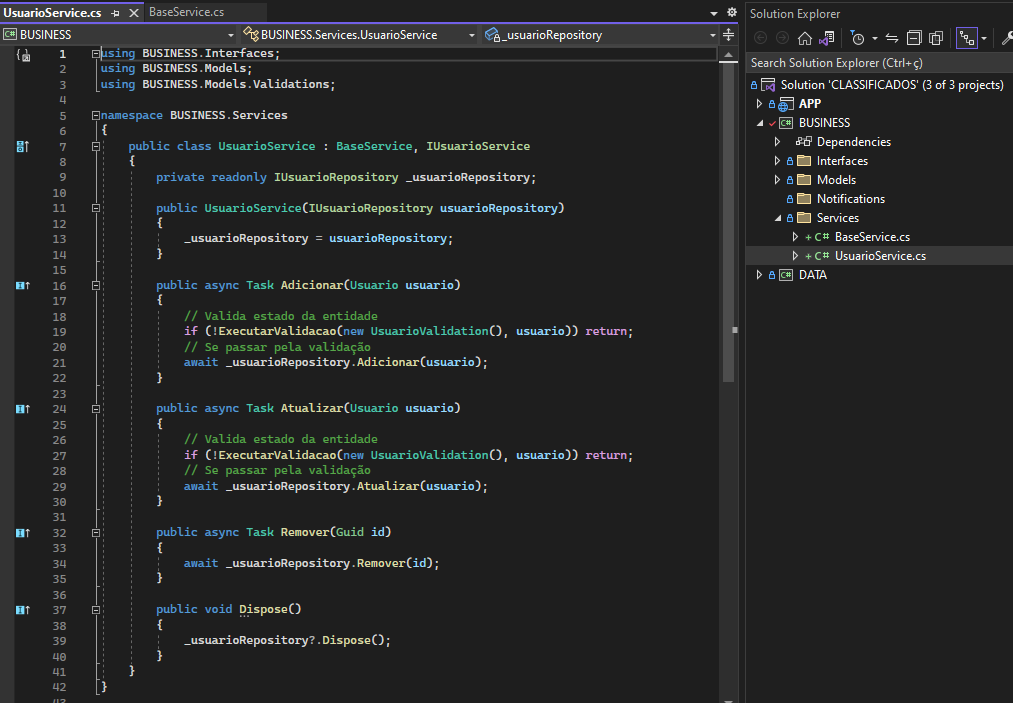
Metodo ObterNotificacoes.cs que retorna essa lista.



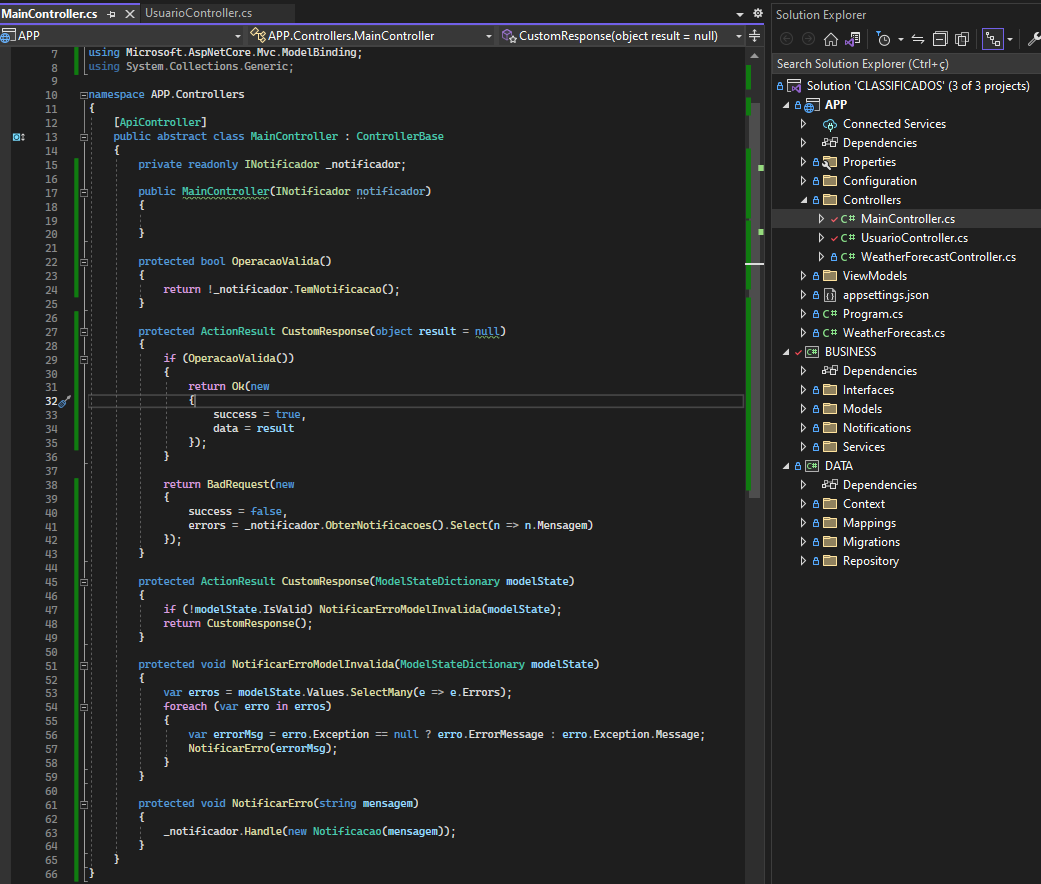
E o metodo TemNotificacao que retorna True ou False.



3.9 – Criar a ModelsServices.cs dentro da Busness/Services.



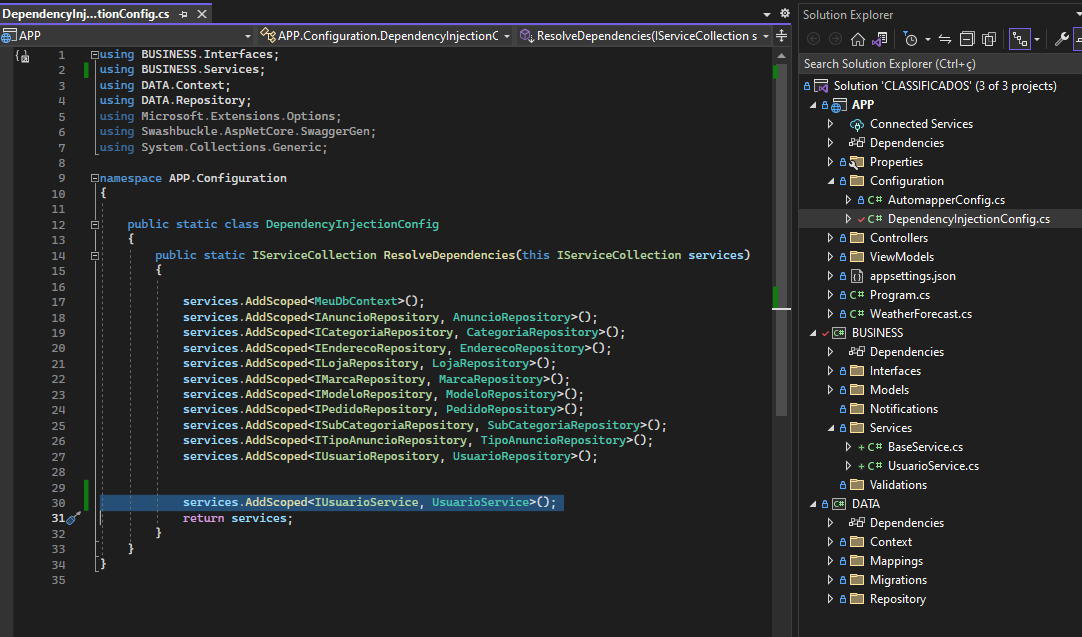
3.10 – Criando a MainController



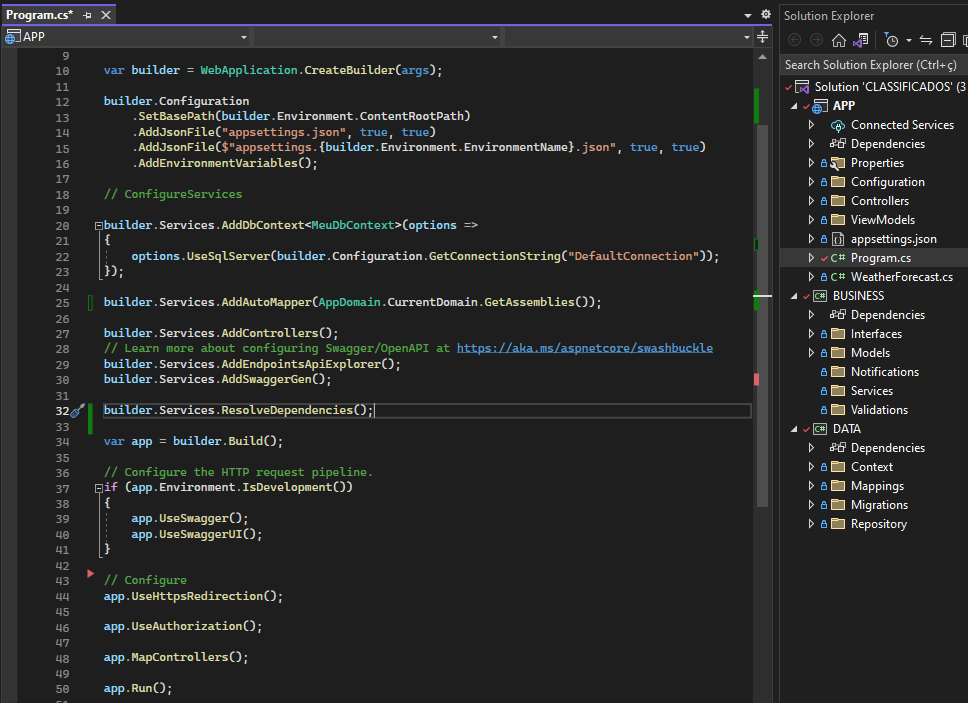
Depois é preciso fazer a injeção de dependencia



3.6 – Criar a classe responsavel pela **resolução da injeção de dependência,** criar a classe **DependencyInjectionConfig.cs** dentro da pasta APP.Configuration.



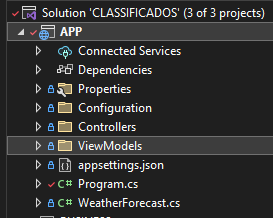
Colocar a classe **ResolveDependencies** no **Program.cs.**



**A camada ne negocios só pode acessar a camada de dados atravez da injeção de dependencias.**

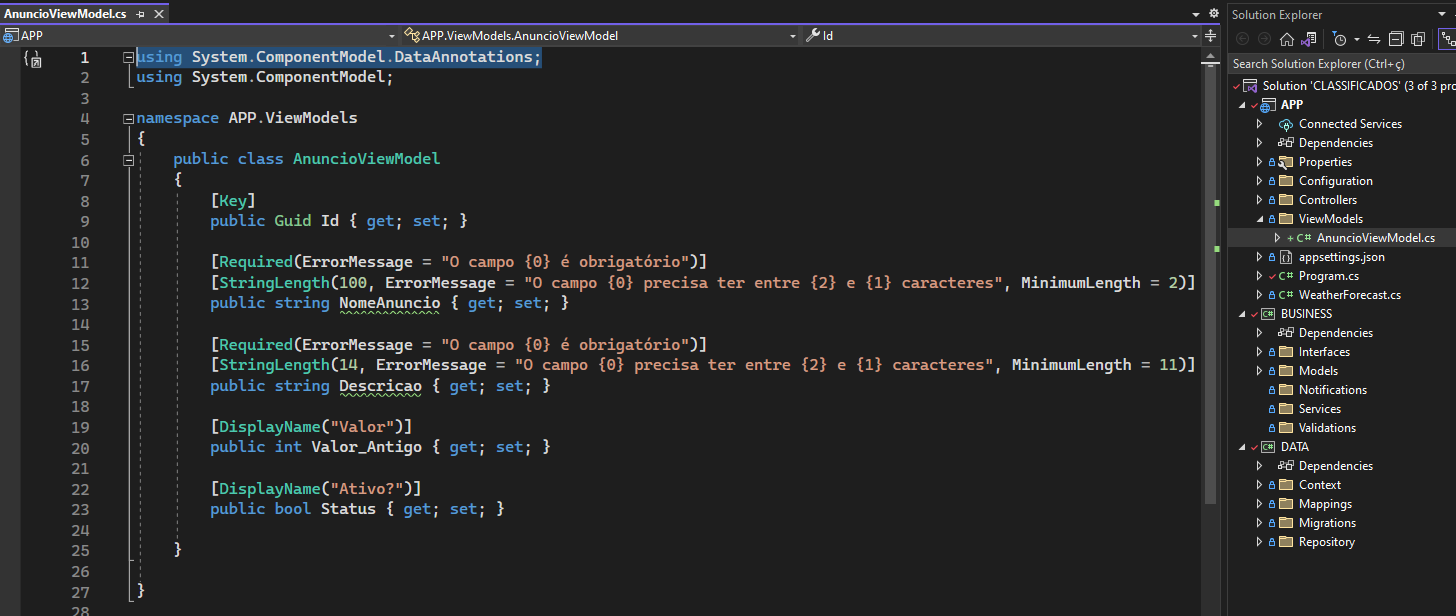
## **4 - Mapeando as entidades em ViewModels com Automapper**

4.1 – Criar a pasta **ViewModels** dentro do projeto APP.



4.2 – Para mapear as **ViewModels** é necessário acrescentar na as classes **Models...ViewModel.**

Depois devemos usar as devidas **ComponentModel.DataAnnotations** par cada situação especifica.

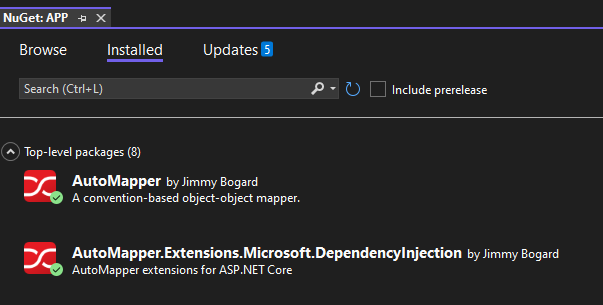
****

**4.3 –** Instalar o automapper via comando na solução **App**

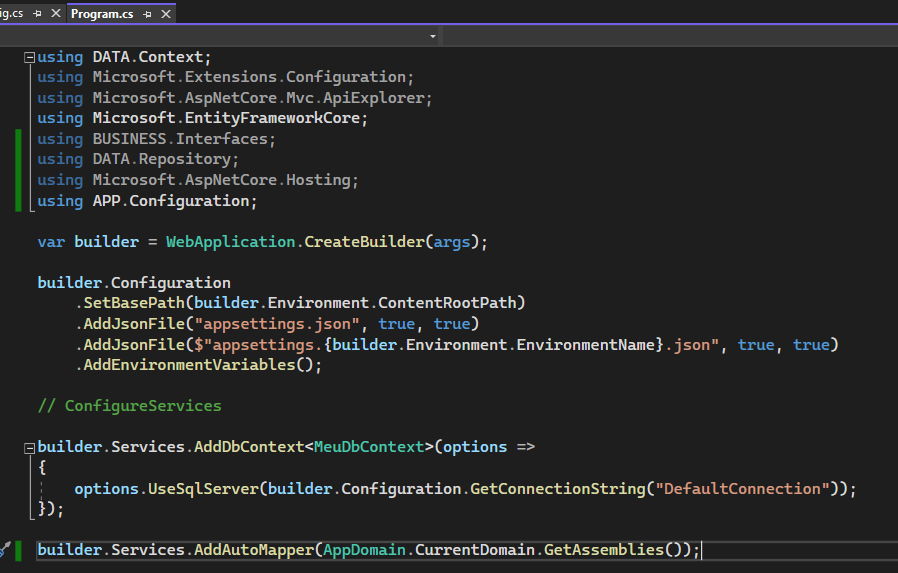
**PM>** Install-Package automapper

Ou

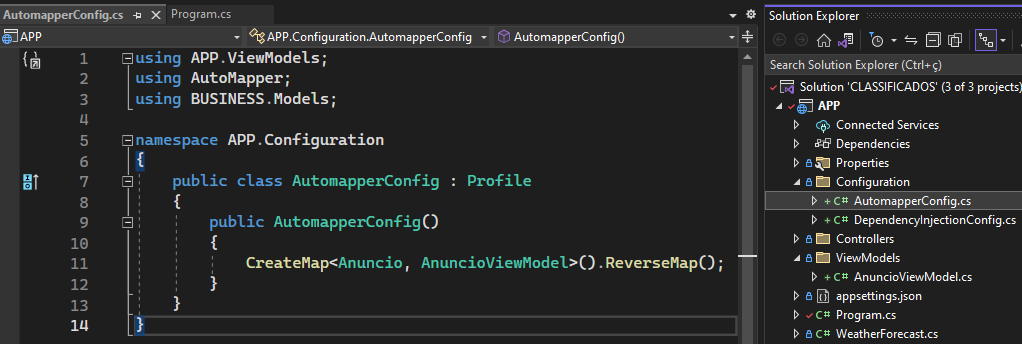
**PM>** Install-Package automapper.extensions.microsoft.dependencyinjection

****

4.4 – Realizar na **Program.cs**.



4.5 – Dentro da pasta **Configuration** criar a classeAutoMapperConfig

****

## **5 – Criação das Controles**