IBaseRepository.cs

Interface genérica para definir os métodos base do repositório.
Exemplo: Inserir, Alterar, Excluir, Consultar, ObterPorId, etc...

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contracts
{
    public interface IBaseRepository<T>
        where T : class
    {
        void Inserir(T entity);
        void Alterar(T entity);
        void Excluir(T entity);
        List<T> Consultar();
        T ObterPorId(int id);
    }
}
```

Expressões LAMBDA para execução de consultas no banco de dados

Em EntityFramework podemos utilizar o recurso do LAMBDA para executar consultas na base de dados, substituindo o uso de SQL.
Iremos criar na interface IBaseRepository métodos que possam receber uma expressão lambda e então executar uma consulta na base de dados.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contracts
{
    public interface IBaseRepository<T>
        where T : class
    {
        void Inserir(T entity);
        void Alterar(T entity);
        void Excluir(T entity);
        List<T> Consultar();
        List<T> Consultar(Func<T, bool> where);
        T Obter(Func<T, bool> where);
        T ObterPorId(int id);
    }
}
```

Demais repositórios herdando o repositório Base:

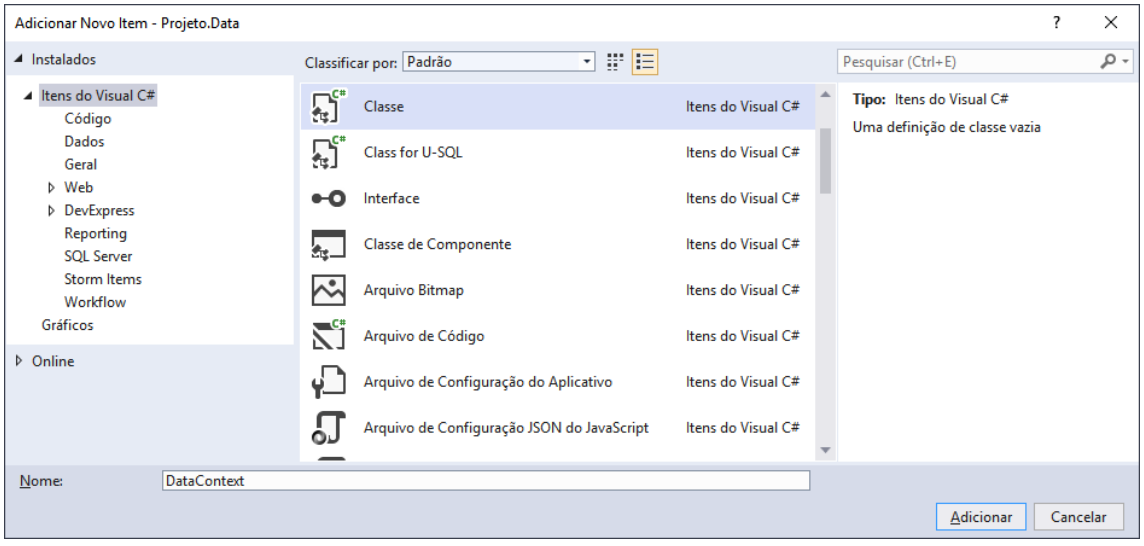
```
using Projeto.Data.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contracts
{
    public interface IEstoqueRepository : IBaseRepository<Estoque>
    {

    }
}
```

```
using Projeto.Data.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contracts
{
    public interface IProdutoRepository : IBaseRepository<Produto>
    {

    }
}
```

Classe de Contexto com o banco de dados

Trata-se de uma classe responsável por realizar a conexão com o banco de dados através do EntityFramework.

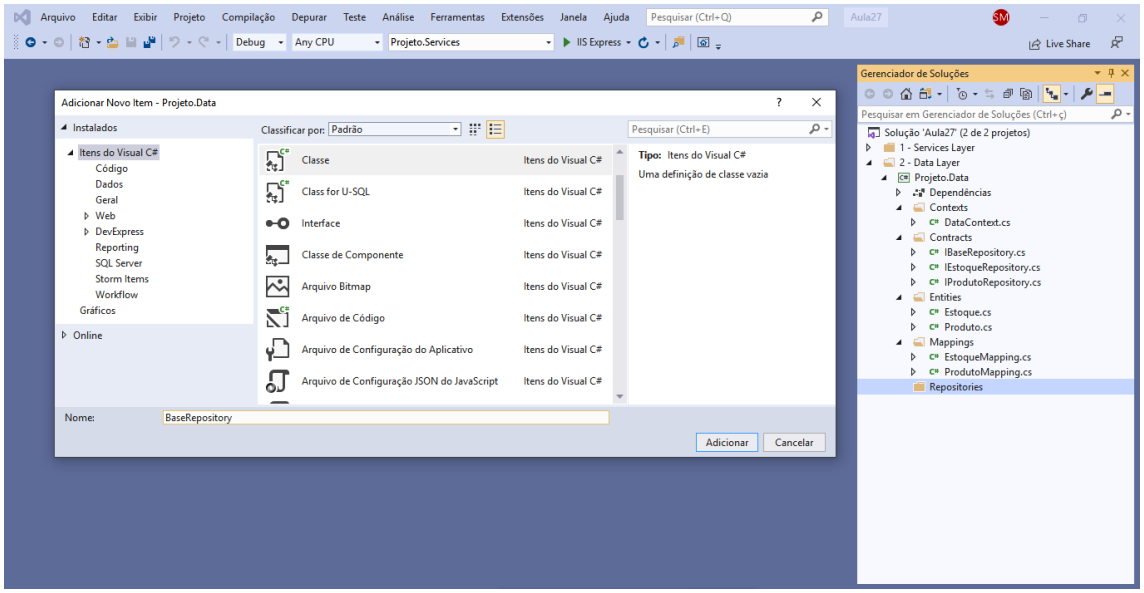
Esta classe é chamada comumente de: **DataContext**



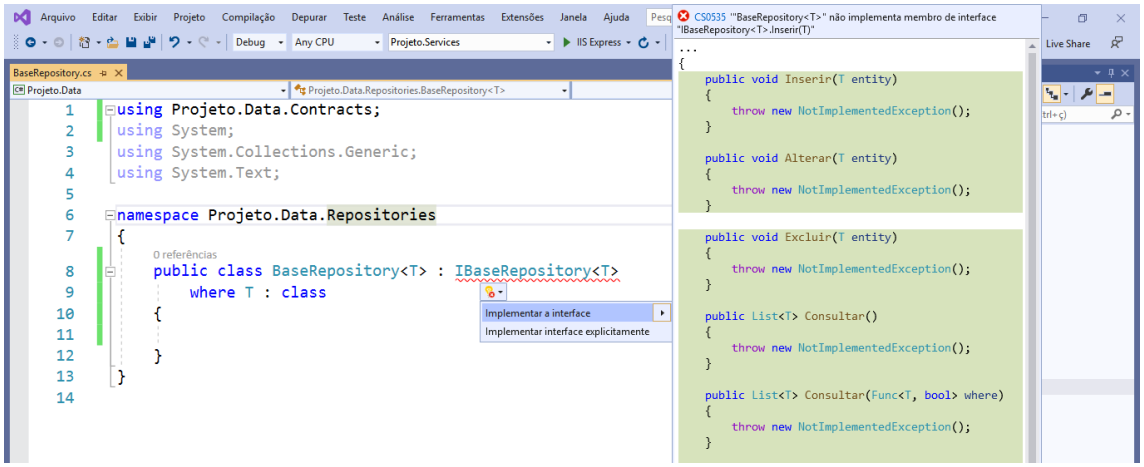
```
using Microsoft.EntityFrameworkCore;
using Projeto.Data.Entities;
using Projeto.Data.Mappings;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contexts
{
    //REGRA 1) Deverá HERDAR DbContext
    public class DataContext : DbContext
    {
        //REGRA 2) Criando um construtor para injeção de dependência
        //este construtor irá receber configurações definidas na
        //classe Startup.cs do projeto API
        public DataContext(DbContextOptions<DataContext> options)
            : base(options) //construtor da superclasse
        {

        }

        //REGRA 3) Sobrescrita (OVERRIDE) do método OnModelCreating
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            //adicionar cada classe de mapeamento (Mapping) feito no projeto
            modelBuilder.ApplyConfiguration(new EstoqueMapping());
            modelBuilder.ApplyConfiguration(new ProdutoMapping());
        }

        //REGRA 4) Declarar um set/get utilizando a classe DbSet do EF
        //para cada entidade do projeto. Este DbSet irá permitir o uso
        //de expressões LAMBDA para executar consultas com qualquer
        //uma das entidades
        public DbSet<Estoque> Estoque { get; set; } //LAMBDA Functions
        public DbSet<Produto> Produto { get; set; } //LAMBDA Functions
    }

}
```

Repositório Genérico

Em EF, podemos implementar os métodos Inserir, Excluir, Alterar, Consultar etc de forma genérica, ou seja, de forma que sirvam para qualquer entidade do projeto.




```
using Projeto.Data.Contracts;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Repositories
{
    public class BaseRepository<T> : IBaseRepository<T>
        where T : class
    {

    }
}
```



```
using Projeto.Data.Contracts;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Repositories
{
    public class BaseRepository<T> : IBaseRepository<T>
        where T : class
    {
        public void Inserir(T entity)
        {
            throw new NotImplementedException();
        }

        public void Alterar(T entity)
        {
            throw new NotImplementedException();
        }

        public void Excluir(T entity)
        {
            throw new NotImplementedException();
        }

        public List<T> Consultar()
        {
            throw new NotImplementedException();
        }

        public List<T> Consultar(Func<T, bool> where)
        {
            throw new NotImplementedException();
        }

        public T Obter(Func<T, bool> where)
        {
            throw new NotImplementedException();
        }

        public T ObterPorId(int id)
        {
            throw new NotImplementedException();
        }

    }

}
```

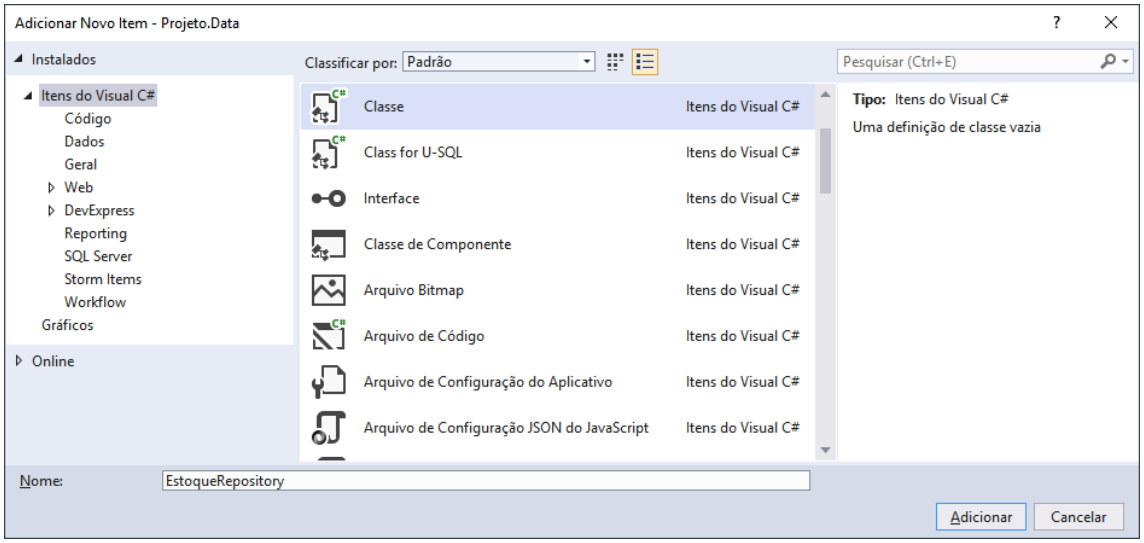
Implementando o repositório genérico:

```
using Microsoft.EntityFrameworkCore;
using Projeto.Data.Contexts;
using Projeto.Data.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Projeto.Data.Repositories
{
    public class BaseRepository<T> : IBaseRepository<T>
        where T : class
    {
        //atributo para armazenar o contexto do EF
        private readonly DataContext dataContext;

        //construtor para injeção de dependência ]
        //(construtor com entrada de argumentos)
        public BaseRepository(DataContext dataContext)
        {
            this.dataContext = dataContext;
        }

        public void Inserir(T entity)
        {
            dataContext.Entry(entity).State = EntityState.Added; //inserção
            dataContext.SaveChanges(); //executando
        }

        public void Alterar(T entity)
        {
            dataContext.Entry(entity).State = EntityState.Modified; //edição
            dataContext.SaveChanges(); //executando
        }

        public void Excluir(T entity)
        {
            dataContext.Entry(entity).State = EntityState.Deleted; //exclusão
            dataContext.SaveChanges(); //executando
        }

        public List<T> Consultar()
        {
            return dataContext.Set<T>().ToList();
        }

        public List<T> Consultar(Func<T, bool> where)
        {
            return dataContext.Set<T>()
                .Where(where)
                .ToList();
        }

        public T Obter(Func<T, bool> where)
        {
            return dataContext.Set<T>()
                .FirstOrDefault(where);
        }
```

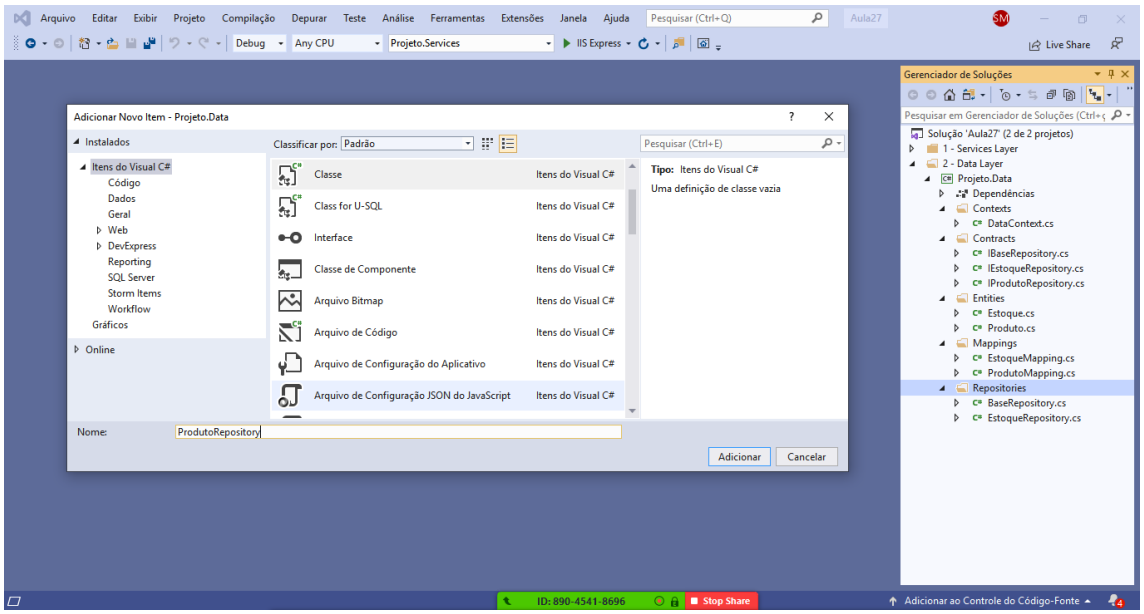
```
        public T ObterPorId(int id)
        {
            return dataContext.Set<T>()
                .Find(id); //buscar pelo id..
        }
    }
}
```

Criando as demais subclasses herdando o repositório genérico:

- EstoqueRepository
- ProdutoRepository

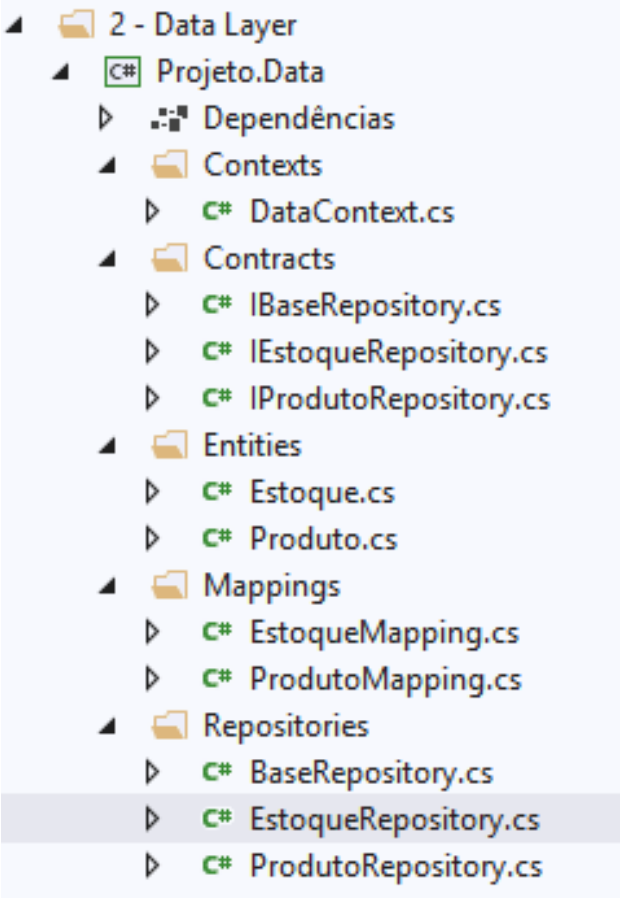



```
using Projeto.Data.Contexts;
using Projeto.Data.Contracts;
using Projeto.Data.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Repositories
{
    public class EstoqueRepository : BaseRepository<Estoque>, IEstoqueRepository
    {
        //atributo
        private readonly DataContext dataContext;

        //construtor para injeção de dependência
        public EstoqueRepository(DataContext dataContext)
            : base(dataContext) //construtor da classe pai..
        {
            this.dataContext = dataContext;
        }
    }
}
```

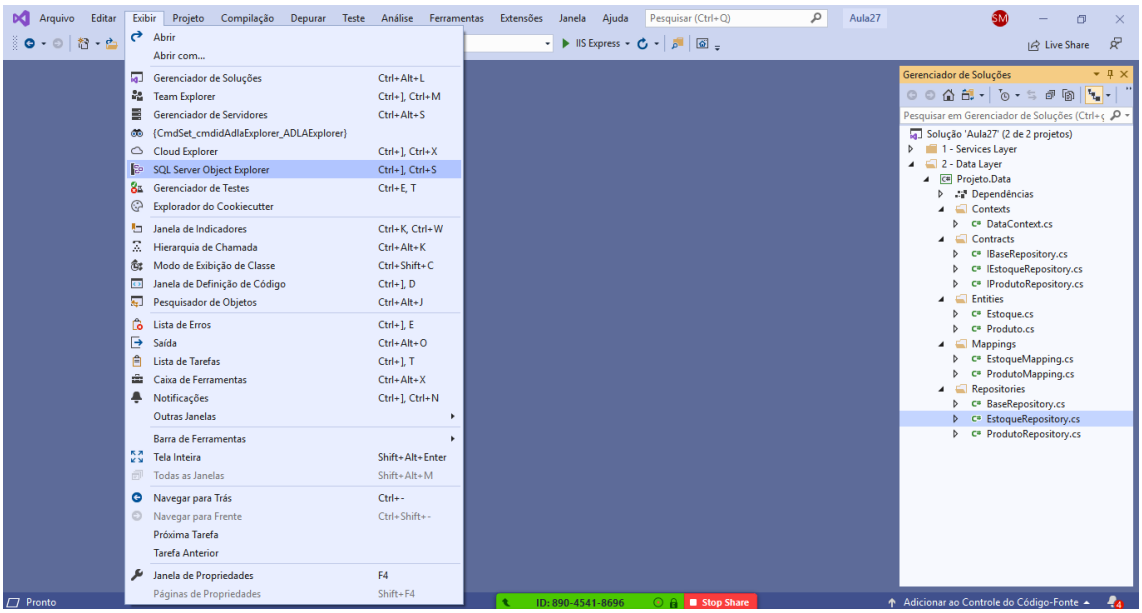


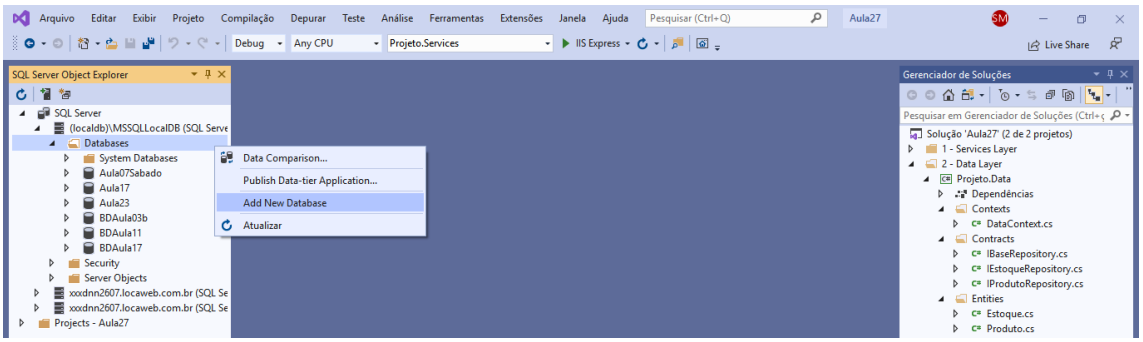

```
using Projeto.Data.Contexts;

using Projeto.Data.Contracts;

using Projeto.Data.Entities;

using System;

using System.Collections.Generic;

using System.Text;


namespace Projeto.Data.Repositories
{
    public class ProdutoRepository : BaseRepository<Produto>, IProdutoRepository
    {
        private readonly DataContext dataContext;

        public ProdutoRepository(DataContext dataContext)
            : base(dataContext)
        {
            this.dataContext = dataContext;
        }
    }
}
```

Estrutura da camada de repositório:

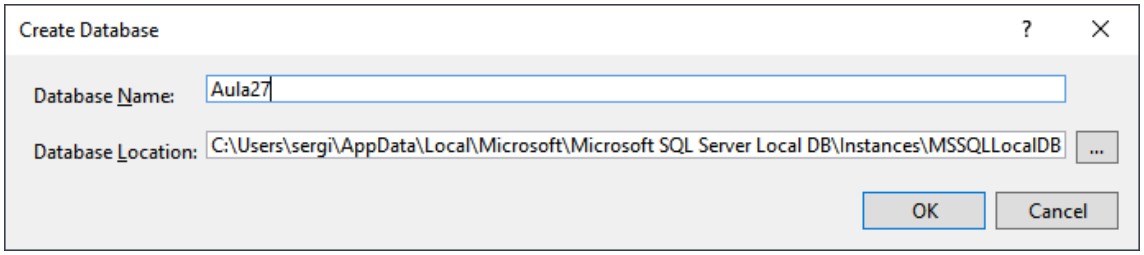


Criando o banco de dados:
Pesquisador de objetos do SQLServer

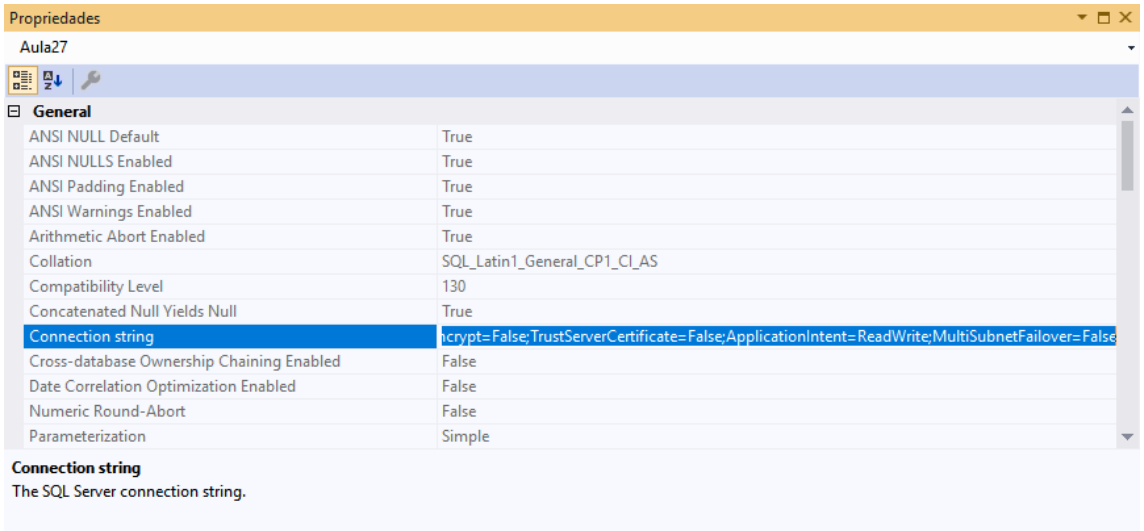


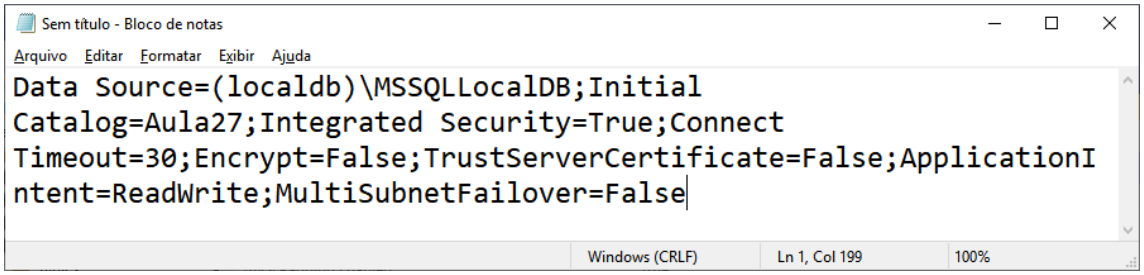


Novo banco de dados:



Obtendo a connectionstring:



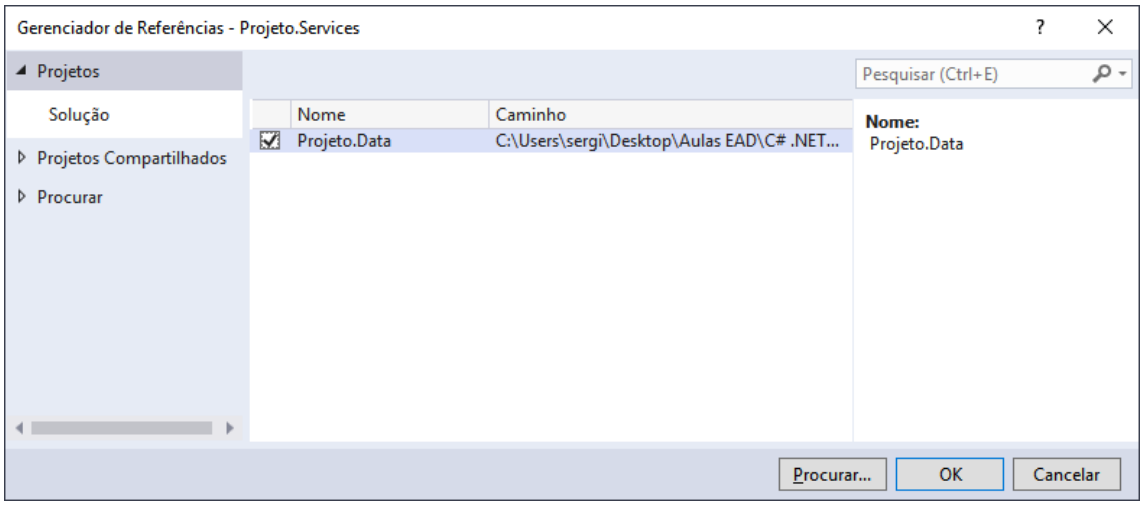


appsettings.json

Mapeamento da string de conexão do banco de dados.

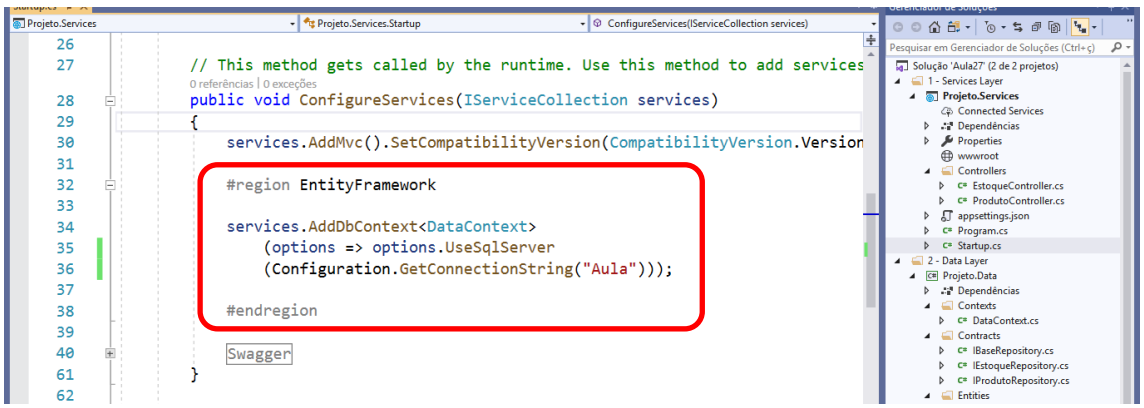
```
{
  "ConnectionStrings": {
      "Aula": "Data Source=(localdb)\\MSSQLLocalDB;Initial
      Catalog=Aula27;Integrated Security=True;Connect
      Timeout=30;Encrypt=False;TrustServerCertificate=False;Applicatio
      nIntent=ReadWrite;MultiSubnetFailover=False"
  }
}
```

Adicionando referência no projeto **Services** para o projeto **Data**



Startup.cs

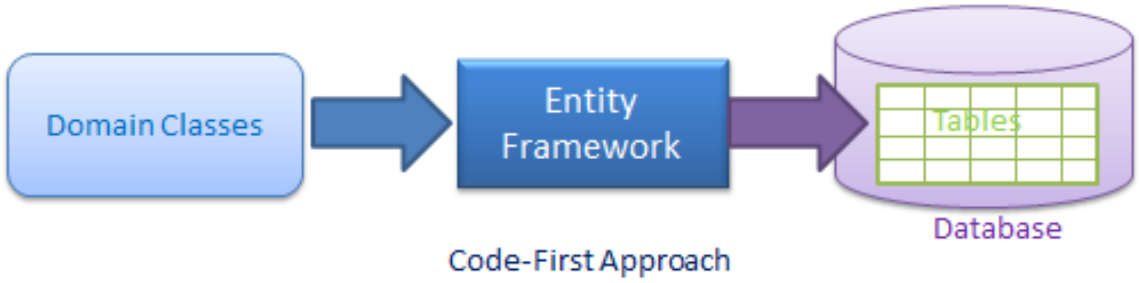
```
#region EntityFramework

services.AddDbContext<DataContext>
      (options => options.UseSqlServer
            (Configuration.GetConnectionString("Aula")));

#endregion
```



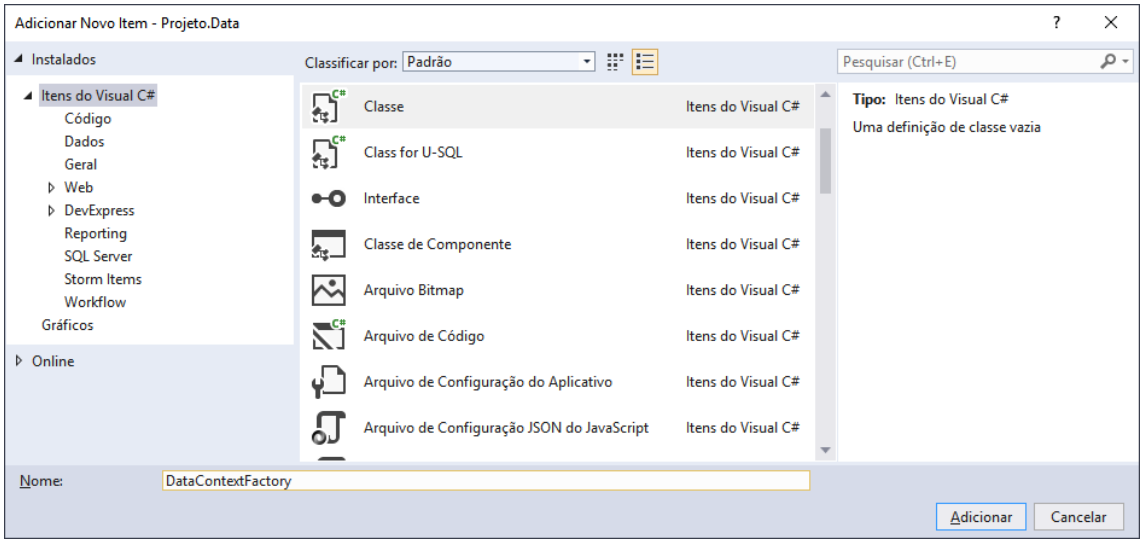
Migrations (CodeFirst)

Recurso do EF capaz de modificar a estrutura da base de dados conforme o mapeamento das entidades do projeto (criar tabelas, adicionar campos, modificar tabelas ou campos, etc)

CodeFirst → Define a ideia de que primeiro iremos criar classes de entidade, mapeamentos e depois então o EF gera / cria essas entidades ou campos no banco de dados de forma automática.



Será necessário criarmos uma classe que ficará responsável por executar o serviço de migration. Esta classe irá rodar como um programa local (Main)



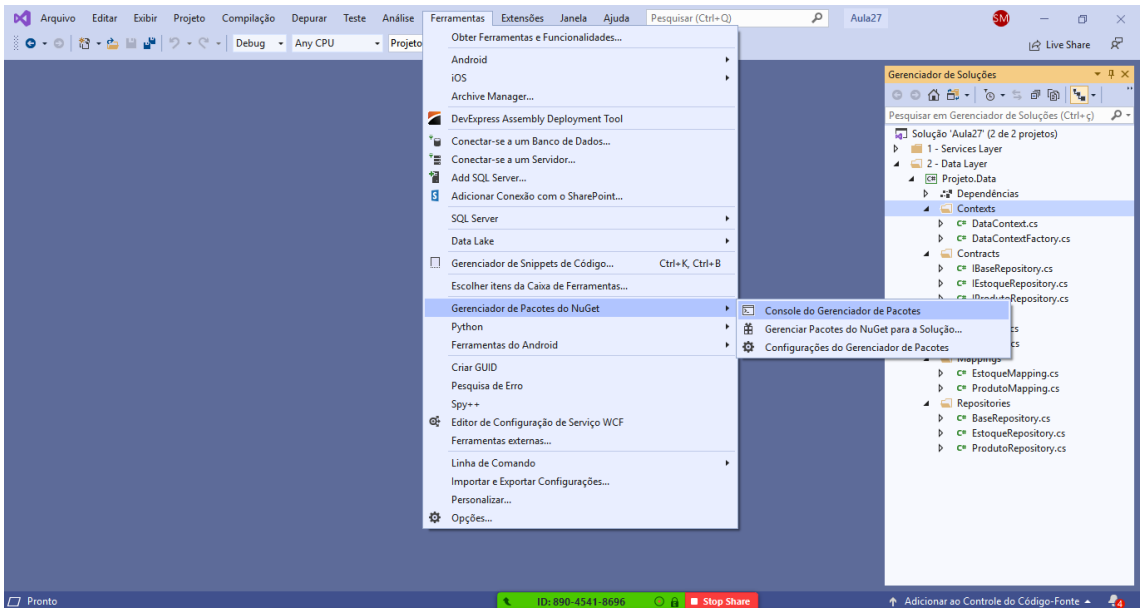

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Data.Contexts
{
    //classe destinada somente à execução do Migrations (CodeFirst)
    public class DataContextFactory : IDesignTimeDbContextFactory<DataContext>
    {
        //método para executar o migrations no banco de dados
        public DataContext CreateDbContext(string[] args)
        {
```

```
            var builder = new DbContextOptionsBuilder<DataContext>();

            builder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial
              Catalog=Aula27;Integrated Security=True;Connect
              Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIn
              tent=ReadWrite;MultiSubnetFailover=False");

            return new DataContext(builder.Options);
        }
    }
}
```

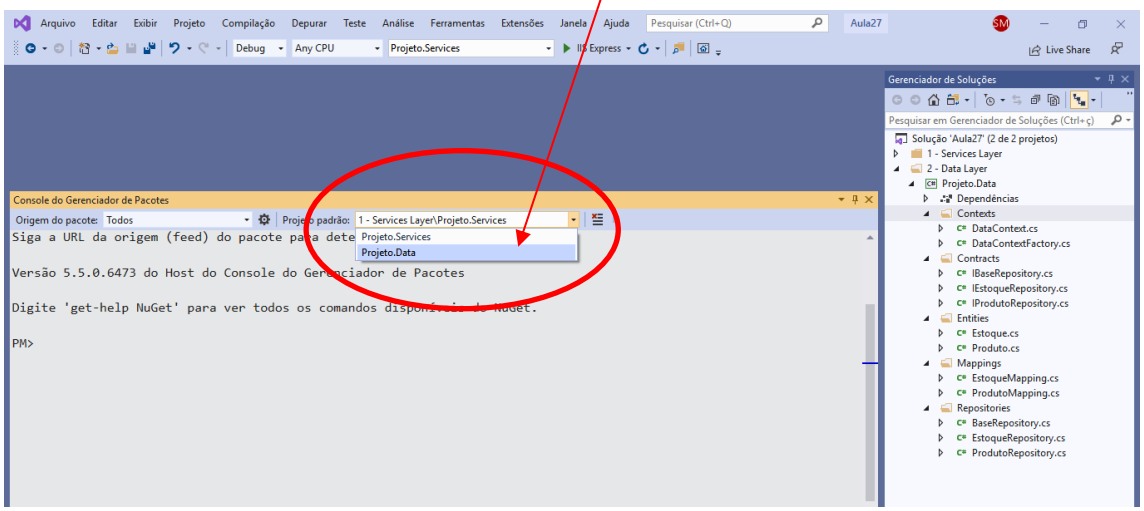
Executando o Migrations

Terminal de comandos do NuGet

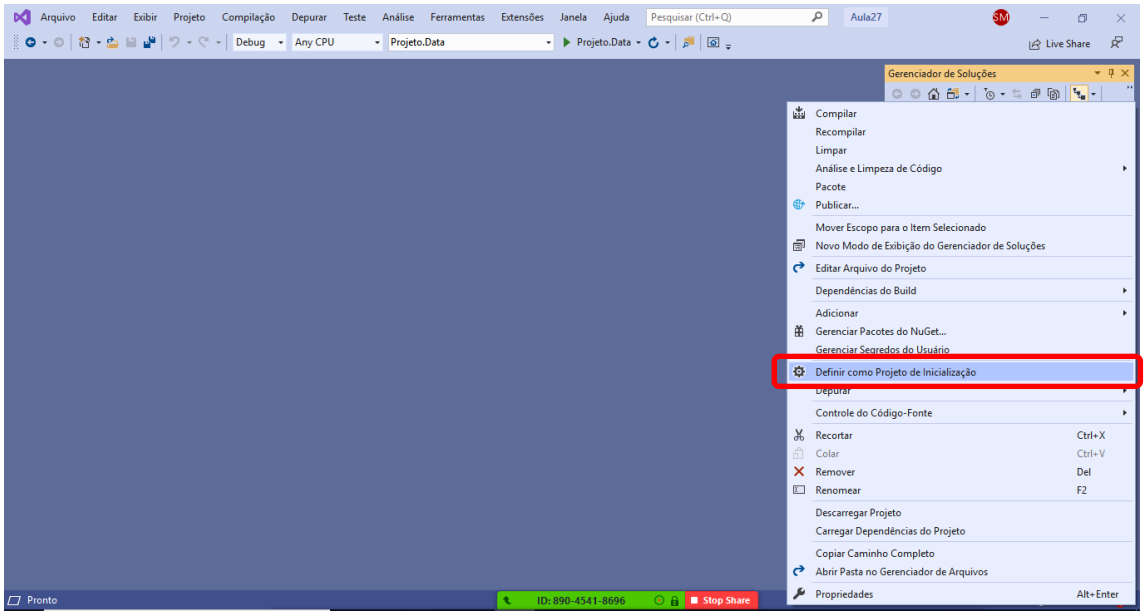
- Ferramentas / Gerenciador de Pacotes do NuGet / **Console do Gerenciador de pacotes**



Selecione como projeto padrão o **Projeto.Data**

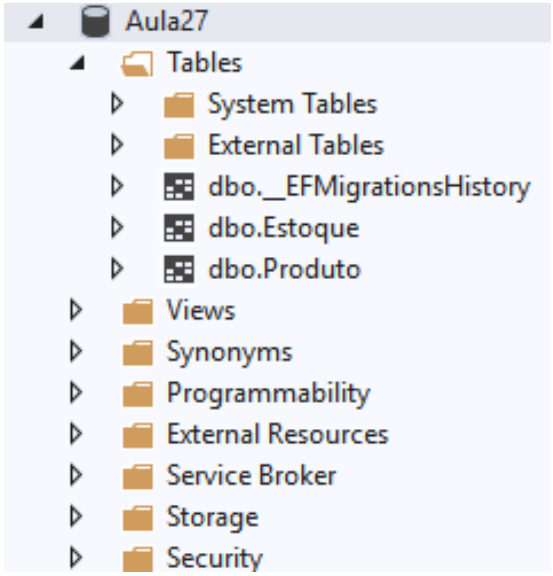


Para executar o Migrations também é necessário colocar o **Projeto.Data** como projeto de inicialização da solution (somente para executar o Migrations)




```
PM> Add-Migration ProjetoAula27
To undo this action, use Remove-Migration.
PM> Update-Database
Applying migration '20200423003332_ProjetoAula27'.
Done.
```

Tabelas criadas:



Continua...