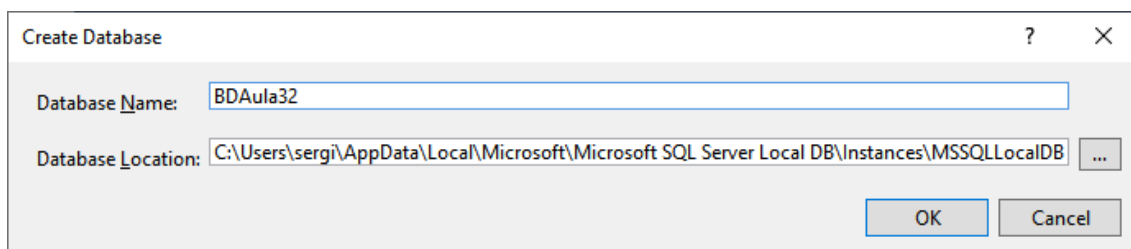
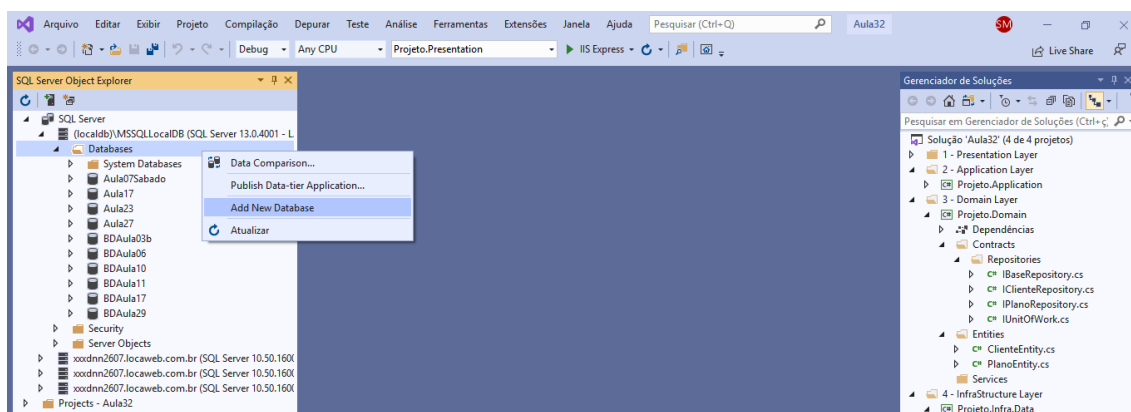
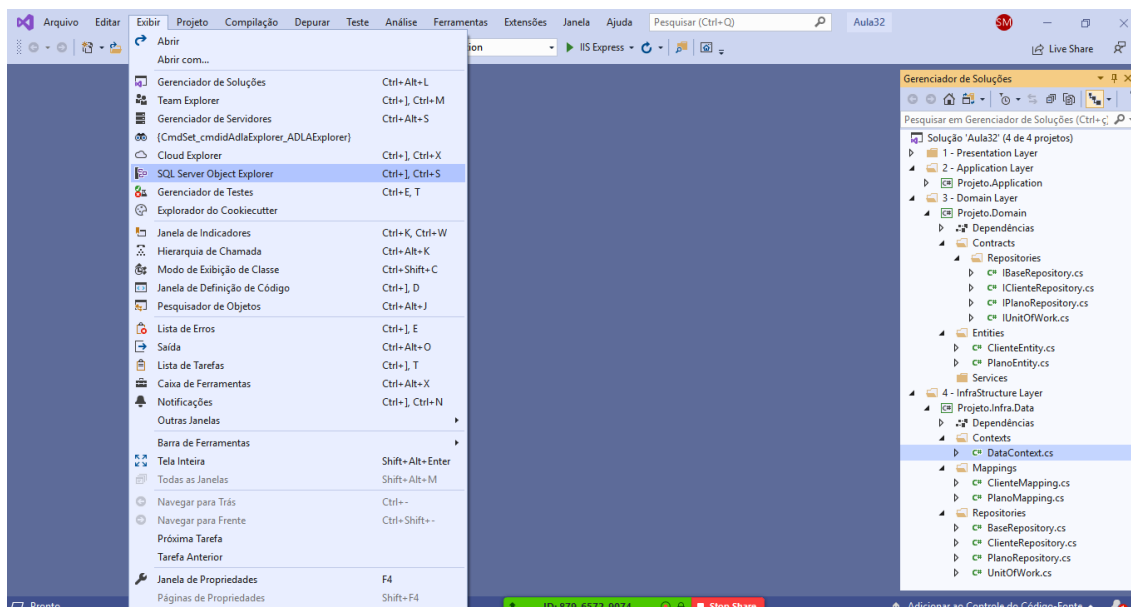
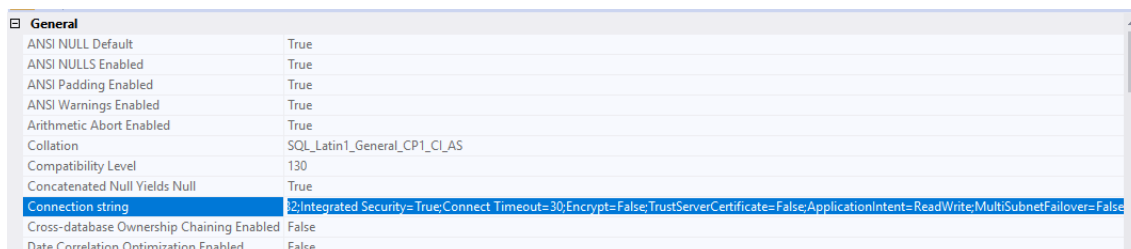
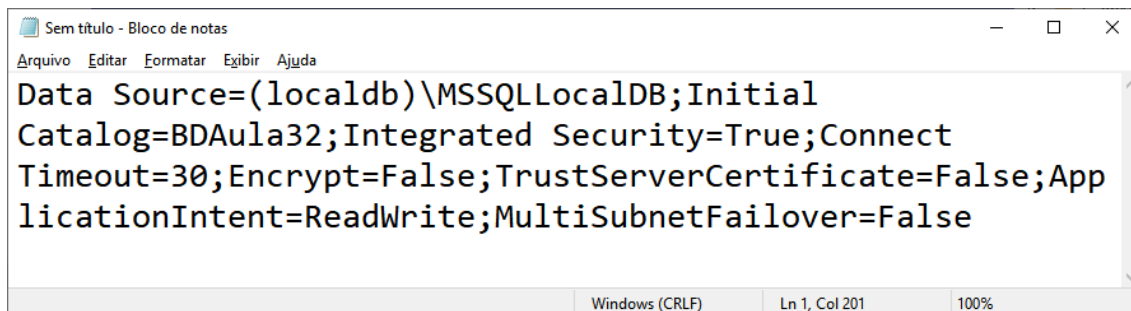


Criando uma base de dados:



Obtendo a string de conexão do banco de dados:





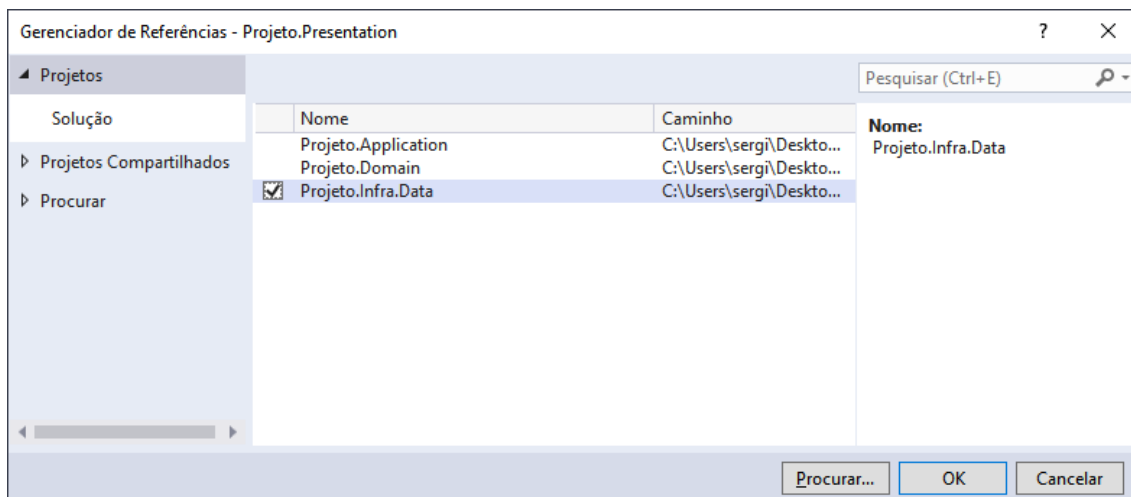
```
Sem título - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=BDAula32;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;App
licationIntent=ReadWrite;MultiSubnetFailover=False
Windows (CRLF)  Ln 1, Col 201  100%
```

appsettings.json

Mapeamento da string de conexão

```
{
  "ConnectionStrings": {
    "ProjetoDDD": "Data Source=(localdb)\MSSQLLocalDB;Initial
                  Catalog=BDAula32;Integrated Security=True;Connect
                  Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIn
                  tent=ReadWrite;MultiSubnetFailover=False"
  }
}
```

** Adicionando referência no projeto **Presentation** para o projeto **InfraStructure.Data**



Startup.cs

Configurando o EntityFramework e fazer a injeção de dependência da connectionstring.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Projeto.Infra.Data.Contexts;

namespace Projeto.Presentation
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime.
        // Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(
                CompatibilityVersion.Version_2_1);

            #region EntityFramework

            //configurar o uso do EntityFramework na aplicação
            //injeção de dependencia na classe DataContext de forma a enviar
            //o caminho da string de conexão do banco de dados

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

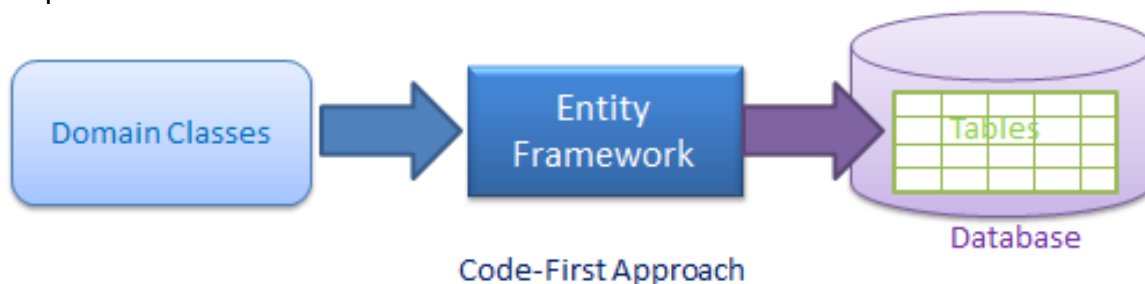
            #endregion
        }

        // This method gets called by the runtime.
        // Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseMvc();
        }
    }
}
```

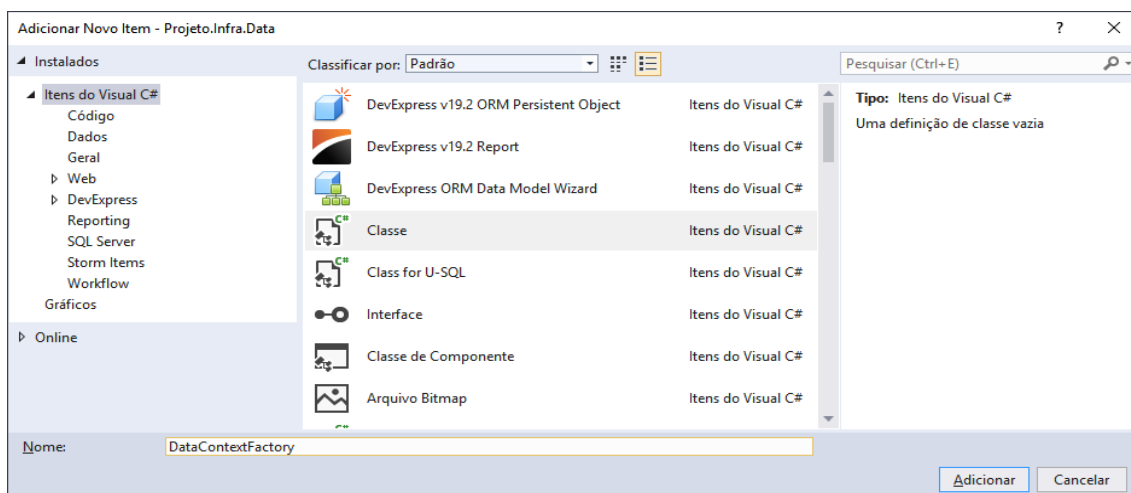
CodeFirst Migrations

Recurso do EntityFramework para interpretar os mapeamentos das entidades e atualizar a estrutura de tabelas do banco de dados conforme o mapeamento.



Para executarmos o Migrations é necessário criarmos uma classe que "rode" o **CodeFirst** e atualize a estrutura do banco de dados.

/Projeto.Infra.Data/Contexts



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

namespace Projeto.Infra.Data.Contexts
{
    public class DataContextFactory : IDesignTimeDbContextFactory<DataContext>
    {
        public DataContext CreateDbContext(string[] args)
        {
            var builder = new DbContextOptionsBuilder<DataContext>();
            builder.UseSqlServer(@"");

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

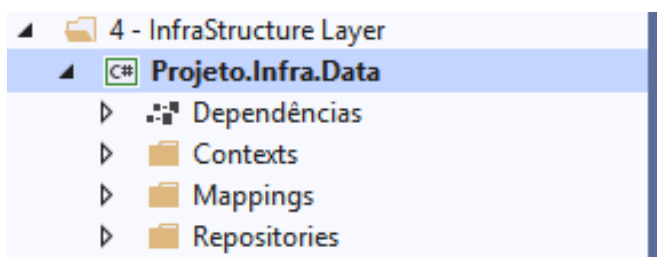
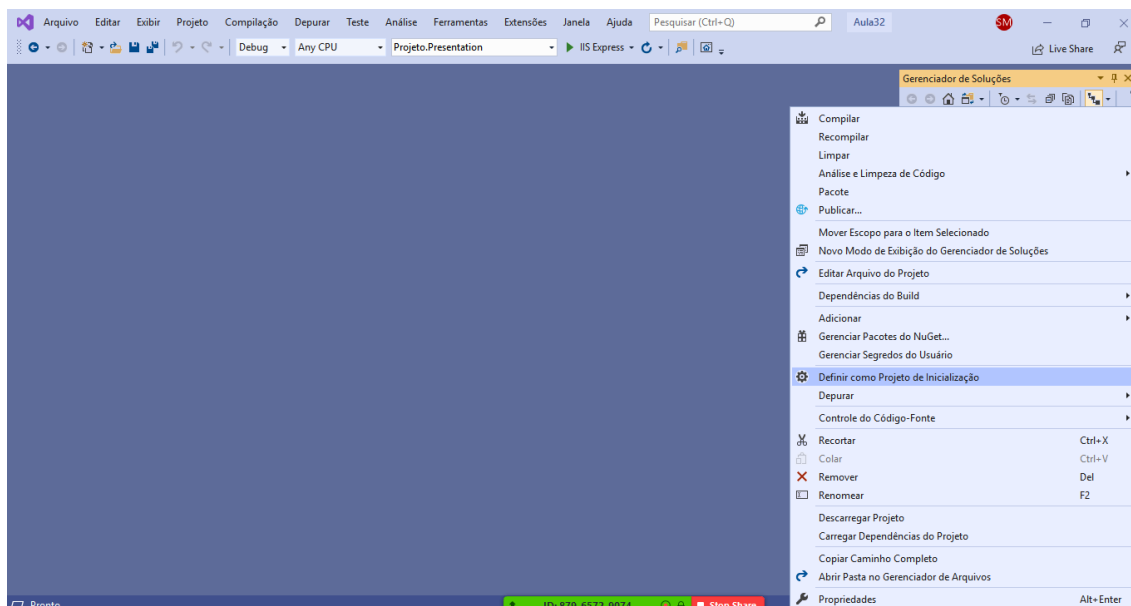
Adicionando a string de conexão:

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

namespace Projeto.Infra.Data.Contexts
{
    public class DataContextFactory : IDesignTimeDbContextFactory<DataContext>
    {
        public DataContext CreateDbContext(string[] args)
        {
            var builder = new DbContextOptionsBuilder<DataContext>();
            builder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial
                                Catalog=BDAula32;Integrated Security=True;Connect
                                Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIn
                                tent=ReadWrite;MultiSubnetFailover=False");

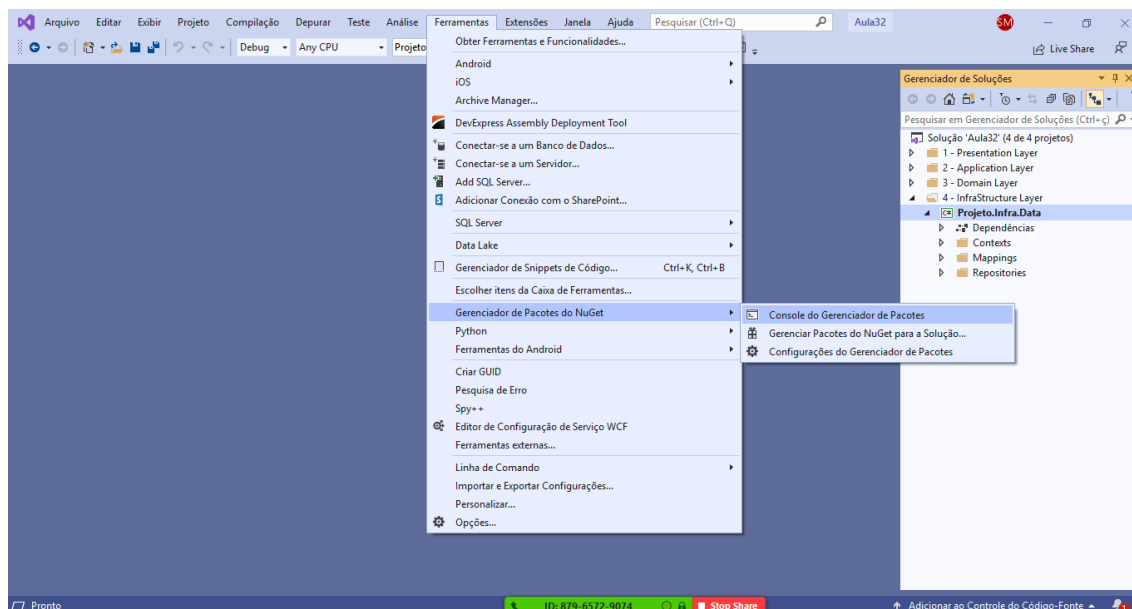
            return new DataContext(builder.Options);
        }
    }
}
```

** Para executar o **Migrations**, coloque o **Projeto.Infra.Data** provisoriamente como **projeto principal da solution**.

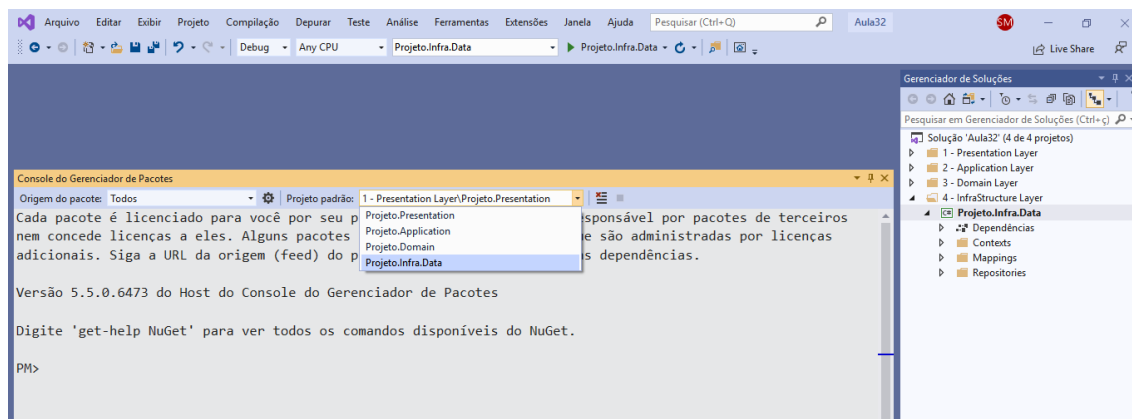


Executando o Migrations

/ Ferramentas / Gerenciador de pacotes do NuGet
/ Console do gerenciador de pacotes



Selecione o projeto Infra.Data



PM> Add-Migration ProjetoDDD

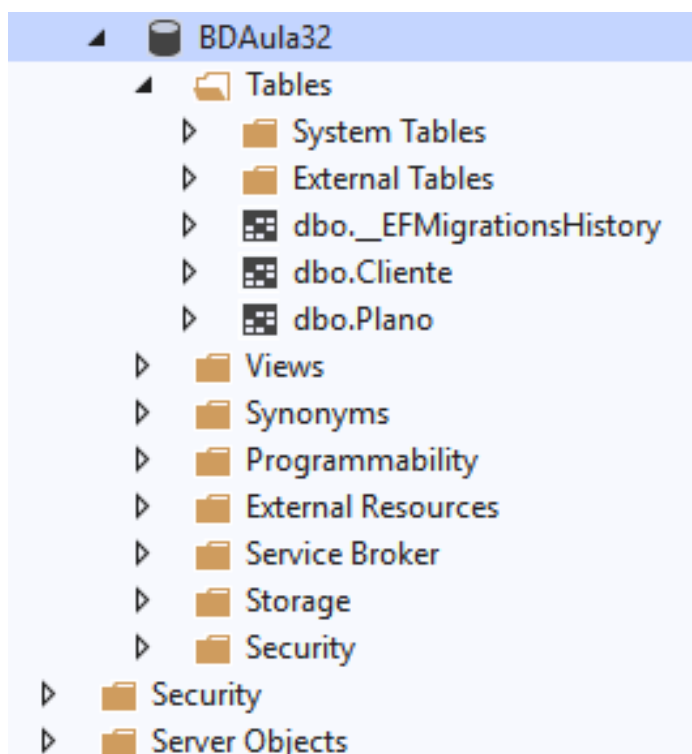
To undo this action, use Remove-Migration.

PM> Update-Database

Applying migration '20200506231249_ProjetoDDD'.

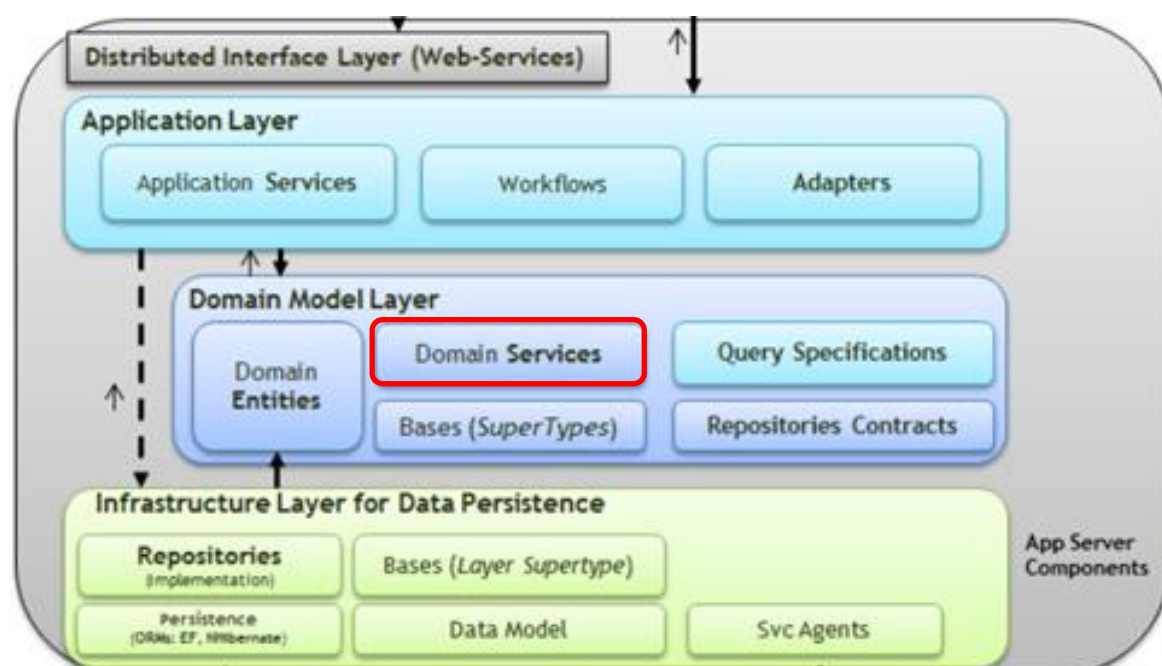
Done.

Base de dados:



Domain Services

Classes que serão desenvolvidas na camada de domínio com o objetivo de implementar as regras de negócio do projeto.

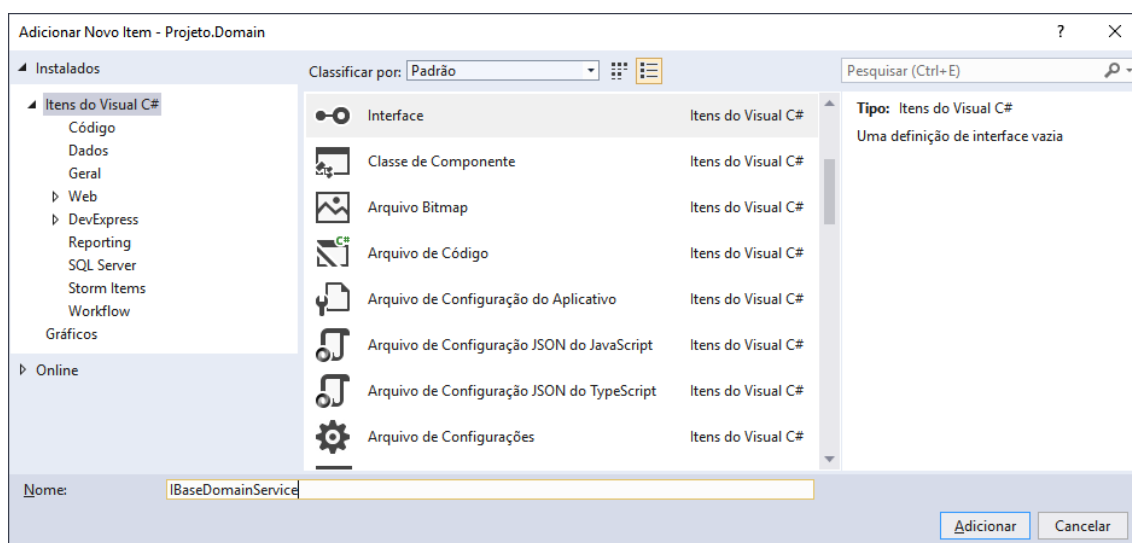


Linguagem Ubíqua

Em DDD, os nomes dos métodos, serviços destinados a implementar as regras de negócio do sistema devem ser os mesmos usados pelos usuários / stakeholders do projeto. Ou seja, é importante evitarmos aqui o uso de nomenclaturas que não sejam as utilizadas no mundo real do usuário.

Primeiro passo:

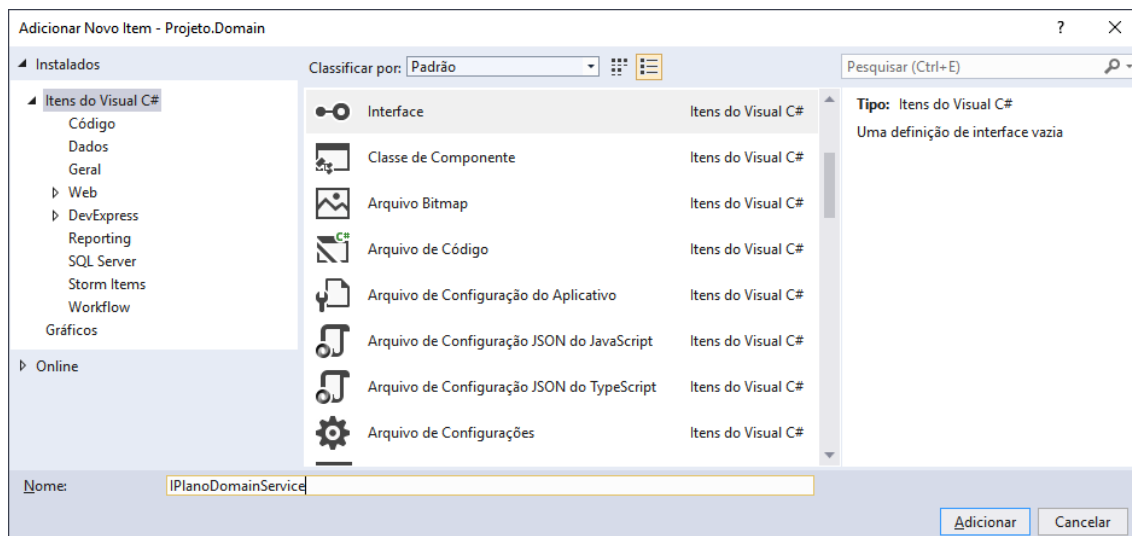
Criando interfaces para definir os contratos das regras de negócio:



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

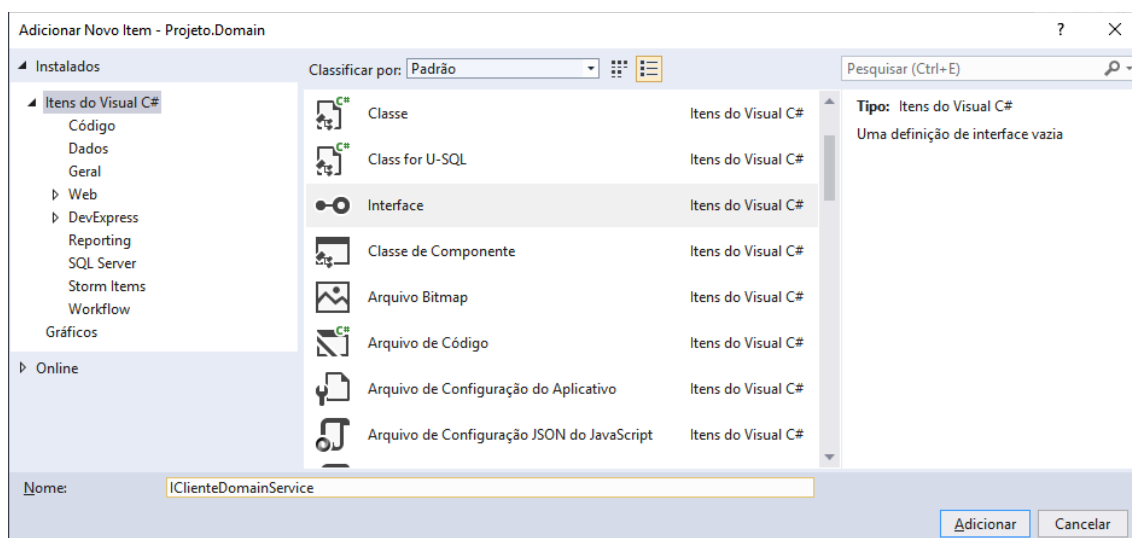
namespace Projeto.Domain.Contracts.Services
{
    public interface IBaseDomainService<TEntity>
        where TEntity : class
    {
        void Cadastrar(TEntity obj);
        void Atualizar(TEntity obj);
        void Excluir(TEntity obj);
        List<TEntity> Consultar();
        TEntity ObterPorId(int id);
    }
}
```


Criando interfaces de negócio específicas para cada entidade de domínio:



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

namespace Projeto.Domain.Contracts.Services
{
    public interface IPlanoDomainService : IBaseDomainService<PlanoEntity>
    {
    }
}
```



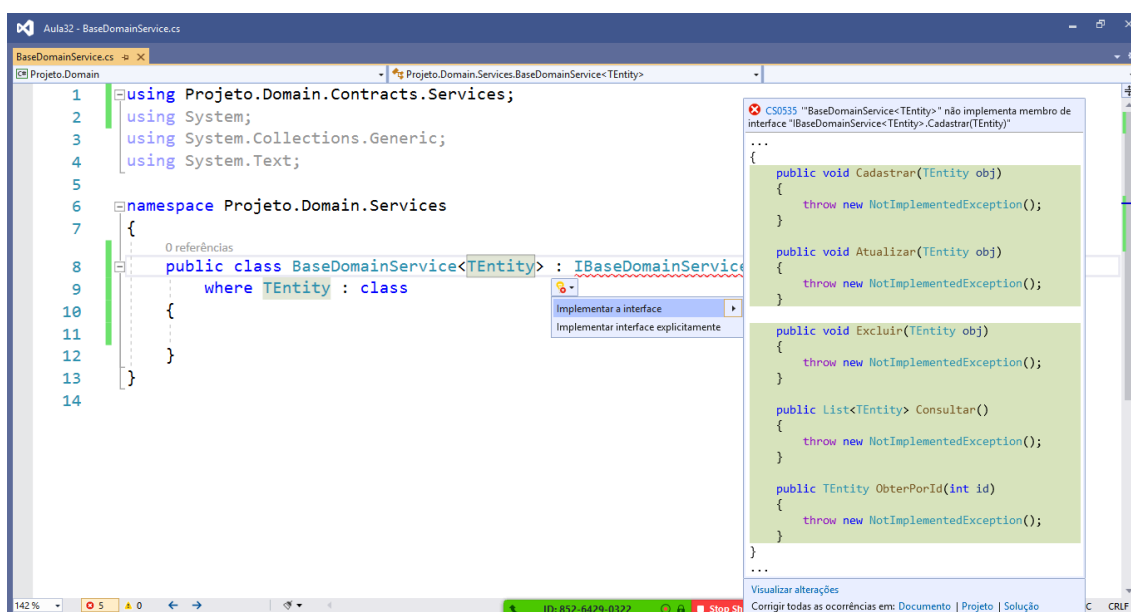
```
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace Projeto.Domain.Contracts.Services
{
    public interface IClienteDomainService : IBaseDomainService<ClienteEntity>
    {
    }
}
```

Implementando as interfaces:

/Services/BaseDomainService.cs

Programando as regras de negócio do sistema de forma genérica.



** Sempre em classes genéricas (Base), podemos usar uma boa prática que é criarmos os métodos com a palavra reservada **virtual** (permitir a sobrescrita do método pelas suas subclasses)

```
using Projeto.Domain.Contracts.Repositories;
using Projeto.Domain.Contracts.Services;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Services
{
    public class BaseDomainService<TEntity> : IBaseDomainService<TEntity>
        where TEntity : class
    {
        //atributo
        private readonly IBaseRepository<TEntity> repository;

        //construtor para injeção de dependência (inicialização)
        public BaseDomainService(IBaseRepository<TEntity> repository)
        {
            this.repository = repository;
        }
    }
}
```

```
public virtual void Cadastrar(TEntity obj)
{
    repository.Create(obj);
}

public virtual void Atualizar(TEntity obj)
{
    repository.Update(obj);
}

public virtual void Excluir(TEntity obj)
{
    repository.Delete(obj);
}

public virtual List<TEntity> Consultar()
{
    return repository.GetAll();
}

public virtual TEntity ObterPorId(int id)
{
    return repository.GetById(id);
}
}
```

Criando classes de regras de negócio (DomainServices) específicas para Plano e para Cliente.

/Services/PlanoDomainService.cs

Programando as regras de negócio do sistema voltadas para PlanoEntity.

```
using Projeto.Domain.Contracts.Repositories;
using Projeto.Domain.Contracts.Services;
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Services
{
    public class PlanoDomainService
        : BaseDomainService<PlanoEntity>, IPlanoDomainService
    {
        //atributo..
        private readonly IPlanoRepository repository;

        //construtor para injeção de dependência (inicialização)
        public PlanoDomainService(IPlanoRepository repository)
            : base(repository) //construtor da superclasse
        {
            this.repository = repository;
        }
    }
}
```

```
using Projeto.Domain.Contracts.Repositories;
using Projeto.Domain.Contracts.Services;
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Services
{
    public class ClienteDomainService
        : BaseDomainService<ClienteEntity>, IClienteDomainService
    {
        //atributo..
        private readonly IClienteRepository repository;

        //construtor para injeção de dependência (inicialização)
        public ClienteDomainService(IClienteRepository repository)
            : base(repository) //construtor da superclasse
        {
            this.repository = repository;
        }
    }
}
```

Regra de Negócio:

- O sistema só deverá permitir o cadastro de clientes maiores de 18 anos.

```
using Projeto.Domain.Contracts.Repositories;
using Projeto.Domain.Contracts.Services;
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Text;

namespace Projeto.Domain.Services
{
    public class ClienteDomainService
        : BaseDomainService<ClienteEntity>, IClienteDomainService
    {
        //atributo..
        private readonly IClienteRepository repository;

        //construtor para injeção de dependência (inicialização)
        public ClienteDomainService(IClienteRepository repository)
            : base(repository) //construtor da superclasse
        {
            this.repository = repository;
        }

        //sobrescrita do método de cadastro
        public override void Cadastrar(ClienteEntity obj)
        {
            //verificando se o cliente é maior de idade
            if (ObterIdade(obj.DataNascimento) >= 18)
            {
            }
        }
    }
}
```

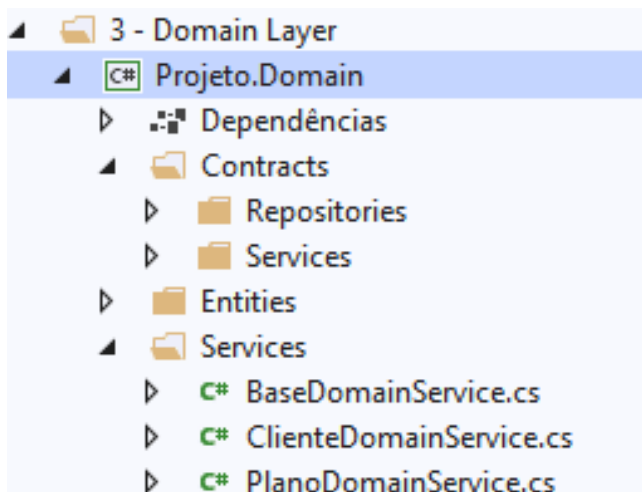
```
        repository.Create(obj);
    }
    else
    {
        throw new Exception("Erro ao cadastrar:
                               O Cliente deve ser maior de idade.");
    }
}

//sobrescrita do método de edição
public override void Atualizar(ClienteEntity obj)
{
    //verificando se o cliente é maior de idade
    if (ObterIdade(obj.DataNascimento) >= 18)
    {
        repository.Update(obj);
    }
    else
    {
        throw new Exception("Erro ao atualizar:
                               O Cliente deve ser maior de idade.");
    }
}

//método private (somente da classe)
private int ObterIdade(DateTime dataNascimento)
{
    var idade = DateTime.Now.Year - dataNascimento.Year;

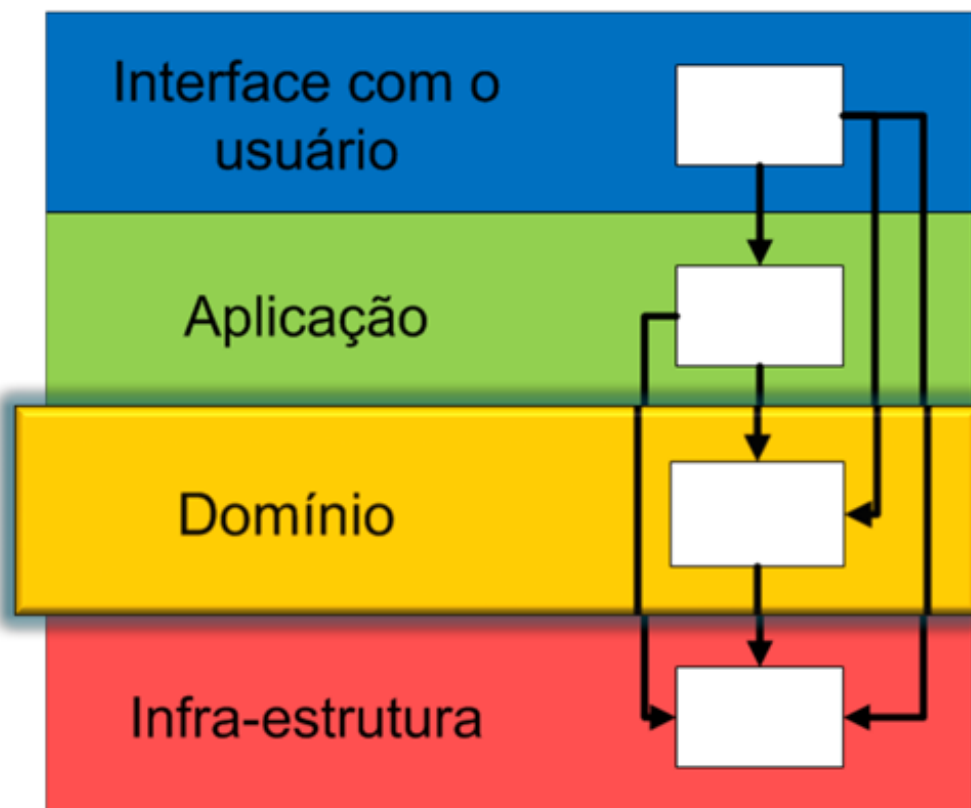
    //verificar se o cliente não fez aniversario
    if(DateTime.Now.DayOfYear < dataNascimento.DayOfYear)
    {
        //descontar 1 ano na idade
        idade = idade - 1;
    }

    return idade;
}
}
```

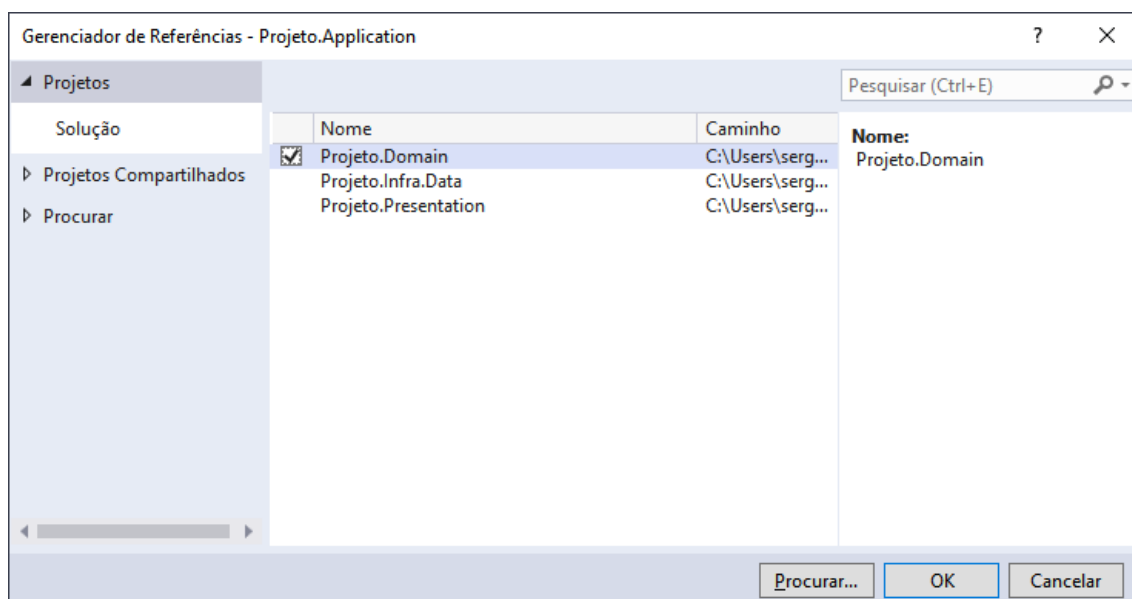


Application

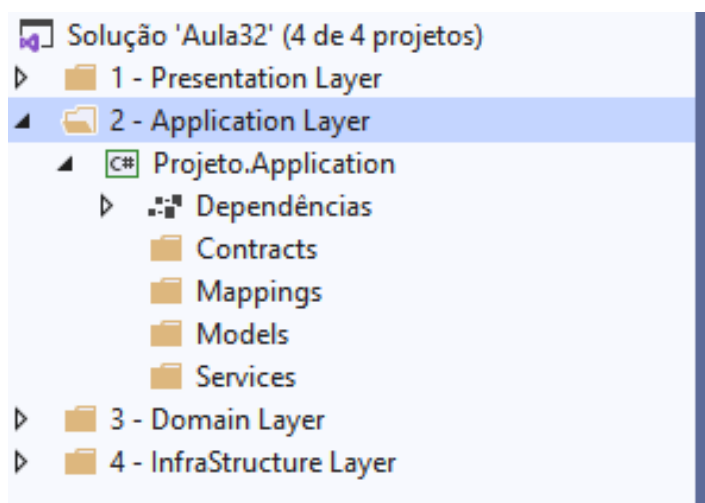
Trata-se de uma camada que irá fazer a junção do **Domínio** com a camada de **Apresentação**.



** Precisamos adicionar na **Application** referencia para a camada **Domain**



Diretórios da camada Application:



Continua...