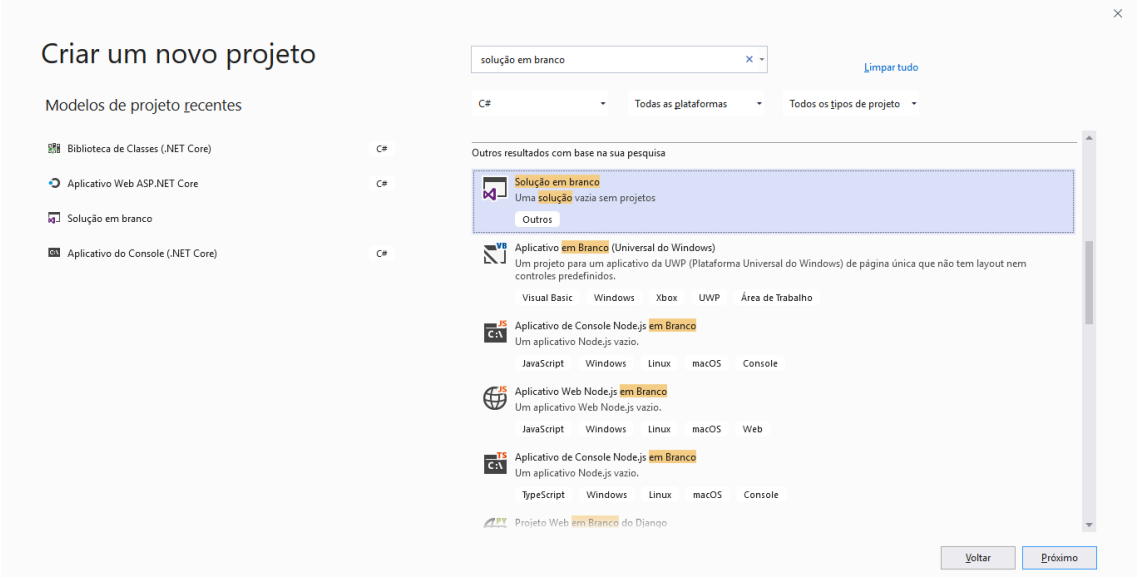
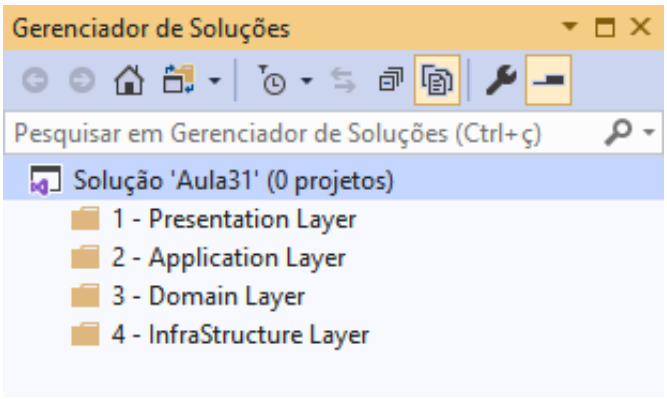
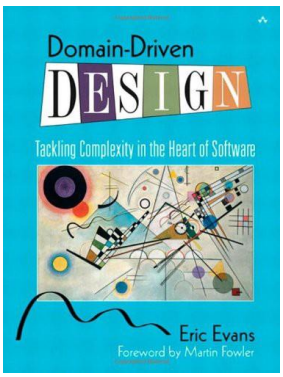
Criando uma solução em branco:
Organizando o sistema:





DDD – Domain Driven Design

Desenvolvimento Orientado a Domínio



O DDD é conjunto de boas práticas voltado para desenvolvimento de aplicações orientadas a Domínio. É um conceito criado por **Eric Evans** no livro "**Domain-Driven Design**" e que tem como objetivo definir um tipo de desenho de aplicação voltado para o domínio de conhecimento do projeto.

O que é o Domínio?

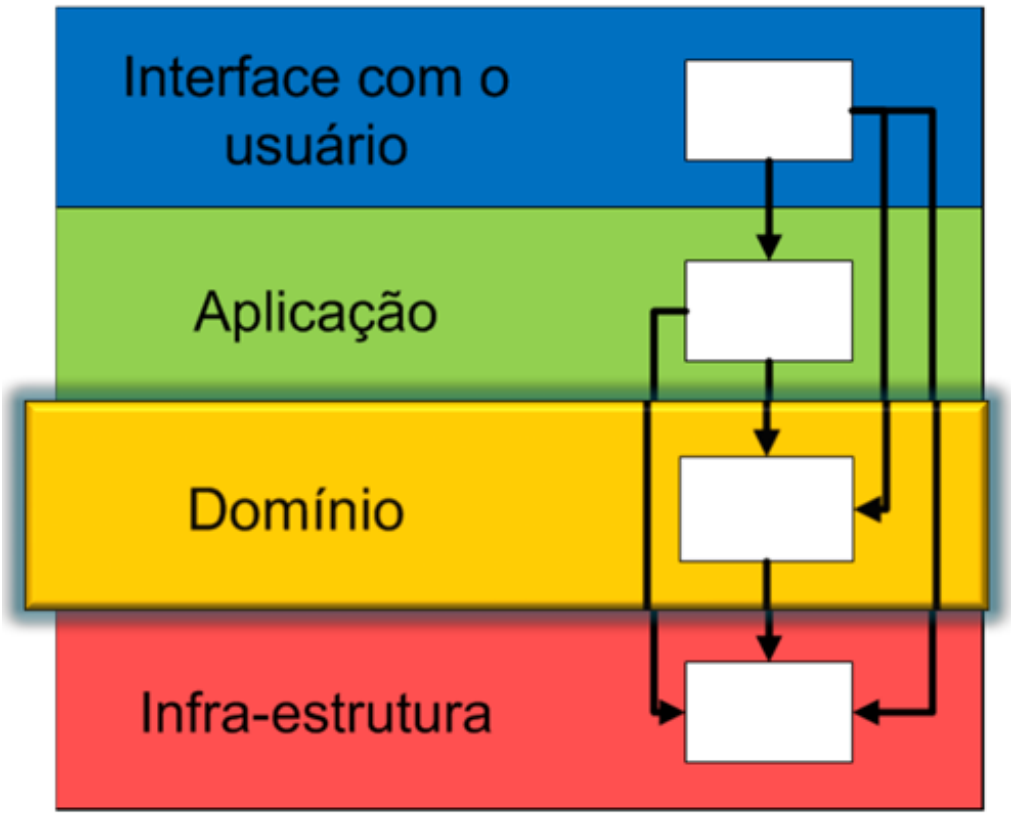
Composto por tudo aquilo que se refere às regras de negócio e conhecimento do projeto. Por exemplo:

- **Modelagem de entidades**
- **Regras de negócio**

O objetivo principal do DDD é desenvolver uma arquitetura de sistemas onde possamos ISOLAR o Domínio de todas as demais camadas do projeto.

Na **camada de domínio**, não iremos utilizar nenhum tipo de framework, biblioteca, etc... Nesta camada faremos uso puro e simples da **linguagem C# e dos princípios do SOLID**.

Um projeto desenvolvido em DDD, deve seguir a seguinte organização:



Presentation Layer

Camada WEB do projeto, contendo tecnologias MVC ou API.

Application Layer

Camada fica entre o domínio e a apresentação, é a camada que faz com que os serviços do domínio (regras de negócio) cheguem até a camada de apresentação.

Domain Layer

Camada principal do projeto. Nesta estão contidos o modelo de entidades do sistema, DTOs, regras de negócio (serviços de domínio), etc. Nesta camada não iremos utilizar nenhum tipo de framework.

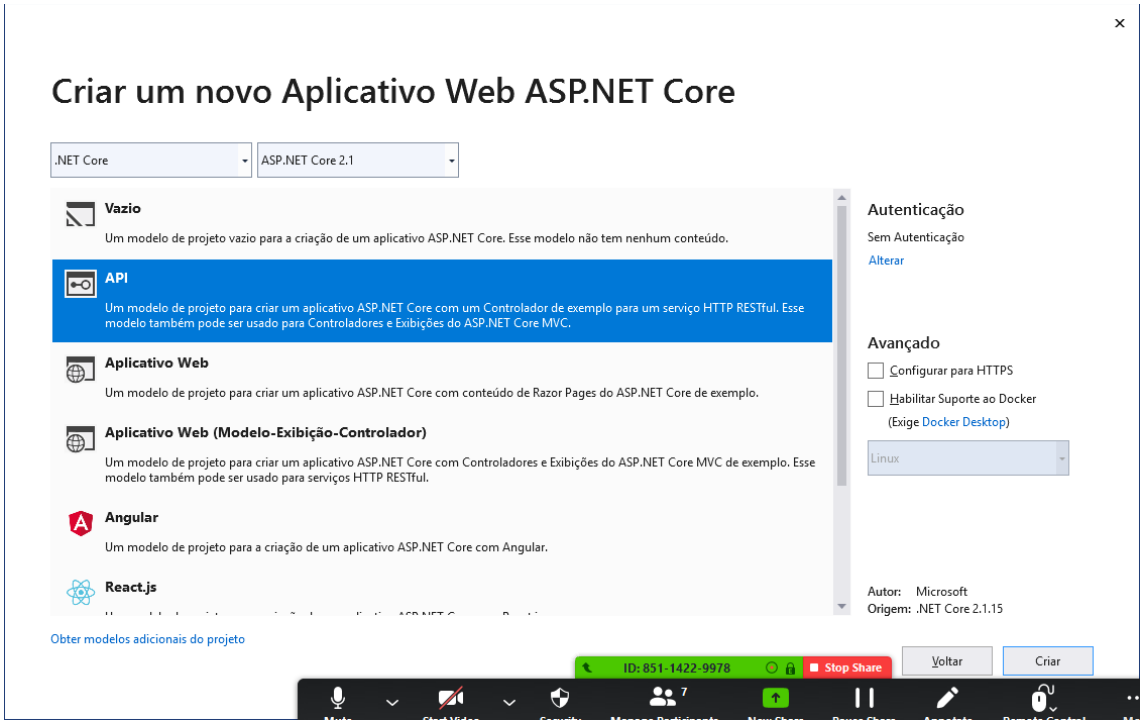
InfraStructure Layer

Camada de infra estrutura do projeto, onde estarão contidos o Repositório (acesso ao banco de dados), demais rotinas que irão dar suporte ao domínio.

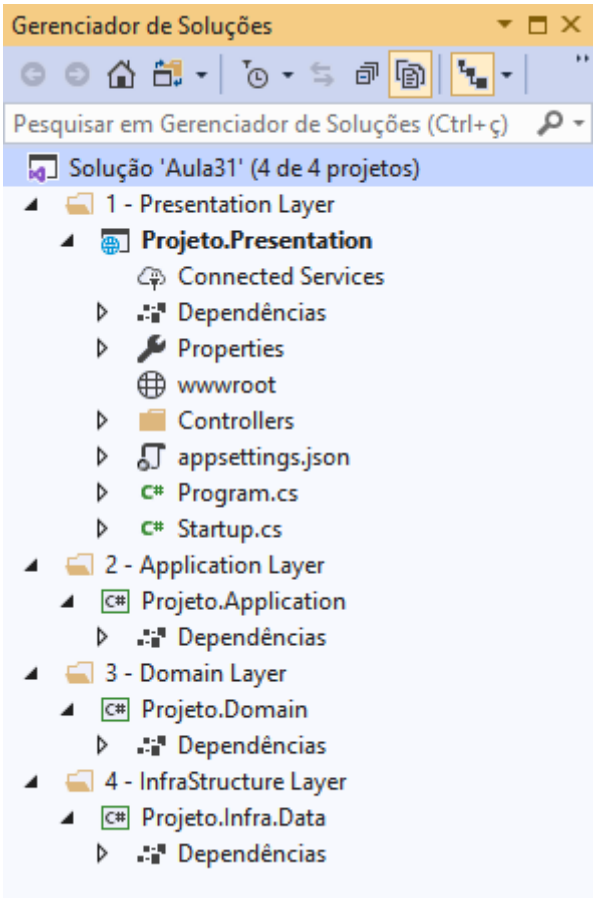
Criando os projetos na solution:

1 – Presentation Layer

Projeto .NET CORE API



Os demais projetos serão criados como **Bibliotecas de Classes**:

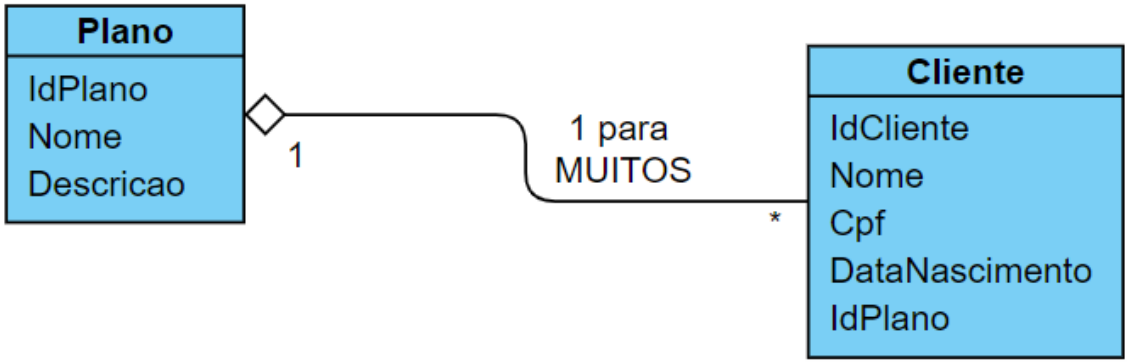


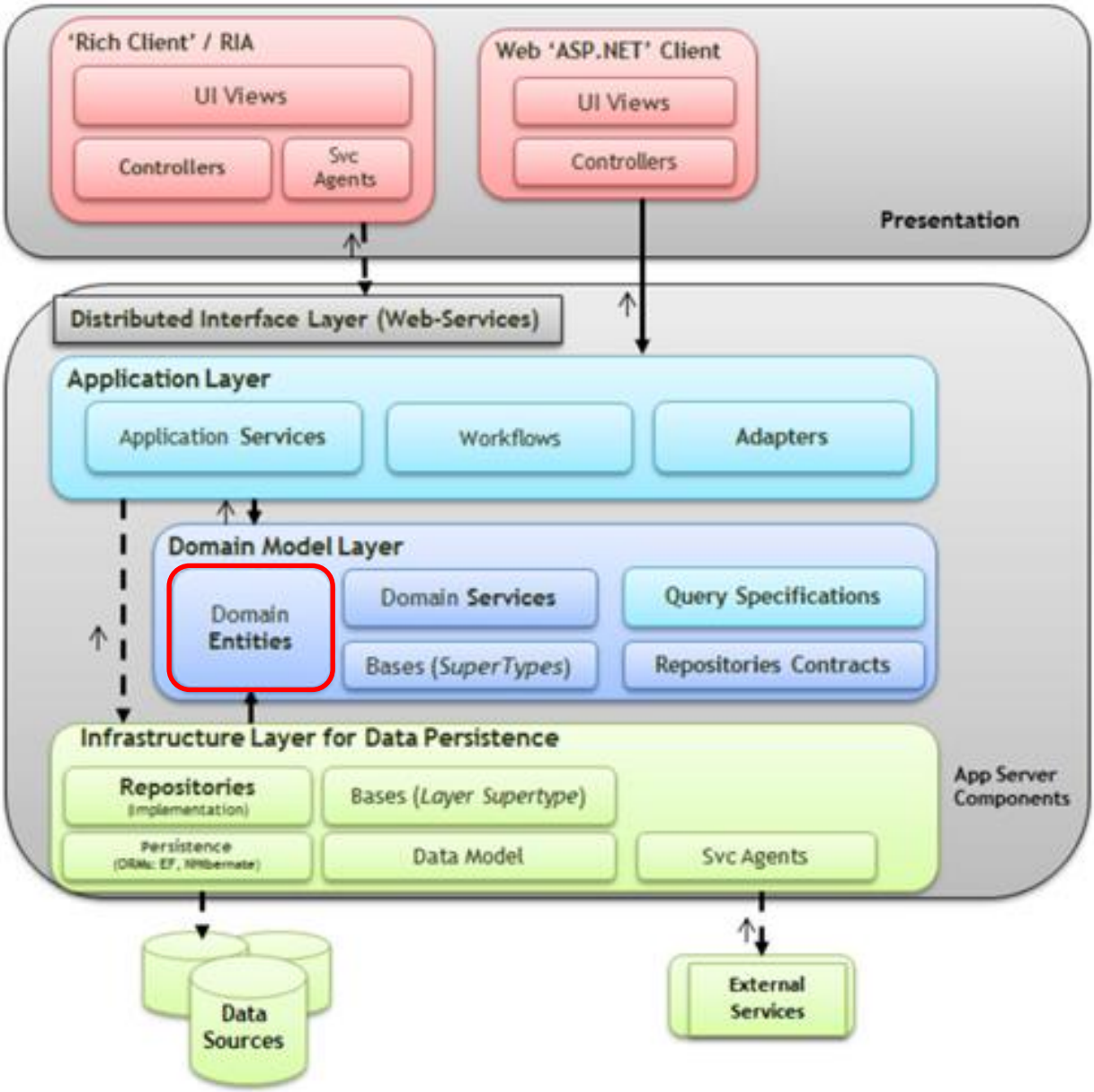
Domain Layer

Camada de domínio, consiste na principal camada do projeto DDD.

Nela estarão contidos o modelo de entidades, regras de negócio do sistema e o uso dos princípios SOLID para desenvolvimento.

Modelagem de entidades





Domain Entities

Entidades do domínio. Classes simples que irão modelar as entidades a partir do qual o domínio será construído.

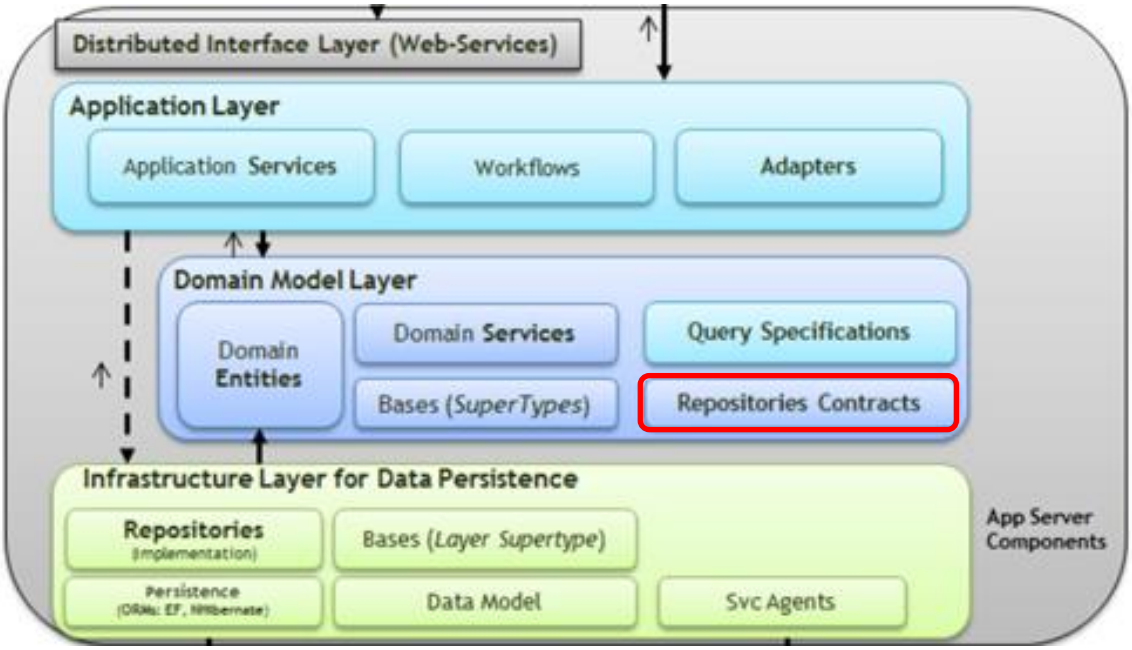
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Entities
{
    public class PlanoEntity
    {
        public int IdPlano { get; set; }
        public string Nome { get; set; }
        public string Descricao { get; set; }
```

```
        #region Relacionamentos

        public List<ClienteEntity> Clientes { get; set; }

        #endregion
    }
}
```

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

namespace Projeto.Domain.Entities
{
    public class ClienteEntity
    {
        public int IdCliente { get; set; }
        public string Nome { get; set; }
        public string Cpf { get; set; }
        public DateTime DataNascimento { get; set; }
        public int IdPlano { get; set; }

        #region Relacionamentos

        public PlanoEntity Plano { get; set; }

        #endregion
    }
}
```

Repositories Contracts

Serão criadas as interfaces que irão definir os métodos a serem implementados na camada de infraestrutura de banco de dados.

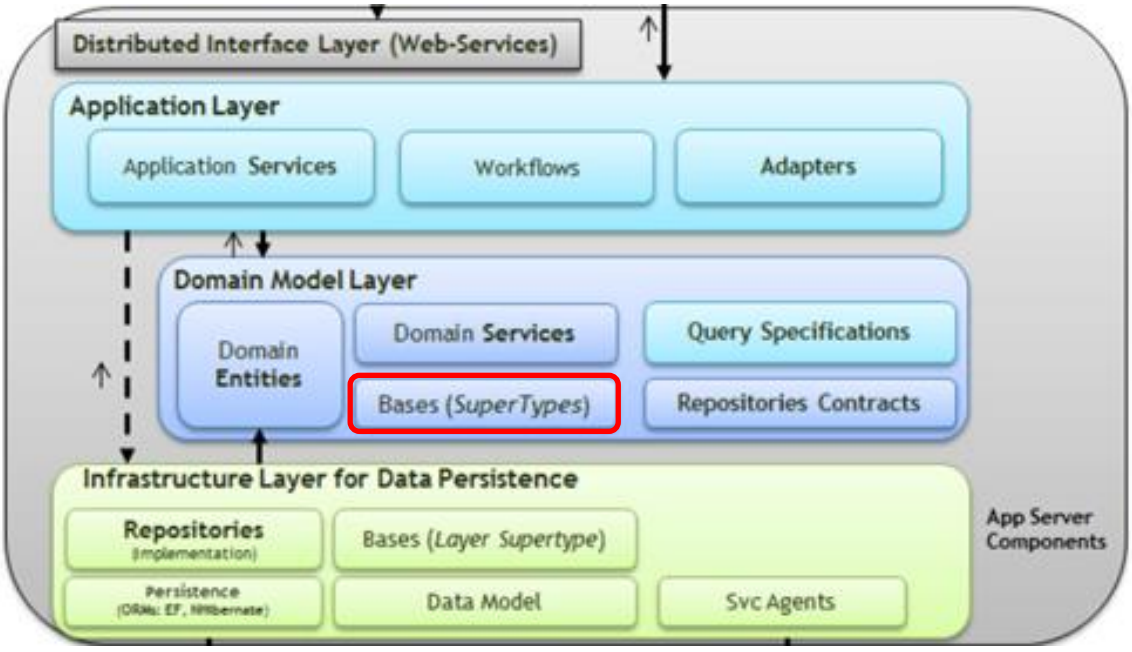


```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IClienteRepository
    {
    }
}
```

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

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IPlanoRepository
    {
    }
}
```

Bases (Super Types)

É uma boa prática em DDD sempre criarmos interfaces ou classes genéricas para depois então especializarmos estas interfaces ou classes.




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

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IBaseRepository
    {
    }
}
```

Bases (Super Types)

Criando a interface genérica:

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

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IBaseRepository<TEntity>
        where TEntity : class
    {
        void Create(TEntity obj);
        void Update(TEntity obj);
        void Delete(TEntity obj);
        List<TEntity> GetAll();
        List<TEntity> GetAll(Func<TEntity, bool> where);
        TEntity Get(Func<TEntity, bool> where);
        TEntity GetById(int id);
    }
}
```

Repositories Contracts

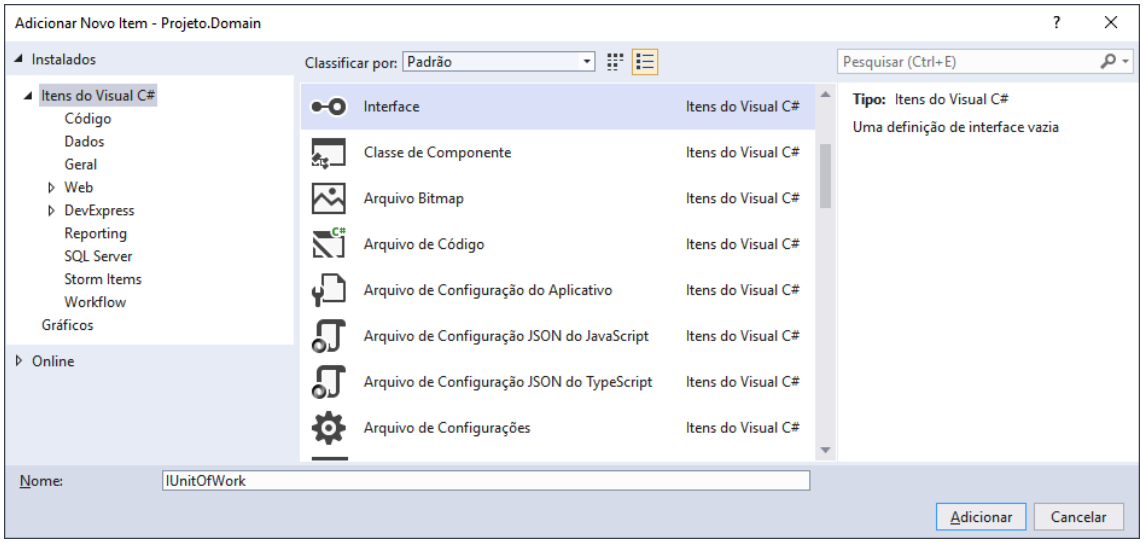
Desenvolvendo as interfaces especificas do repositório:

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

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IClienteRepository : IBaseRepository<ClienteEntity>
    {


    }
}
```

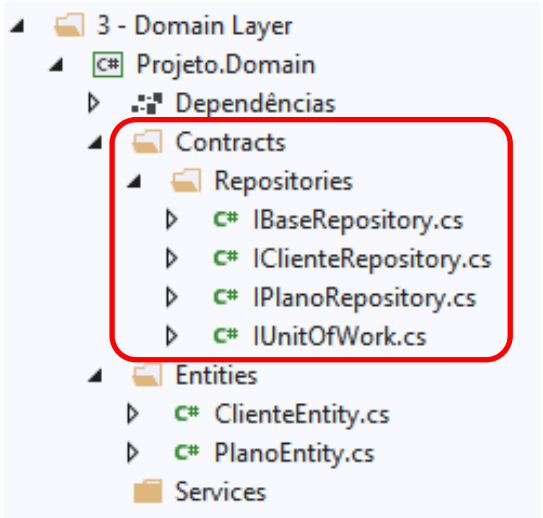
```
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IPlanoRepository : IBaseRepository<PlanoEntity>
    {


    }
}
```

UnitOfWork

Iremos criar uma interface para definir como deverá ser implementado no repositório o padrão UnitOfWork (gerenciamento de transações para todos os repositórios).



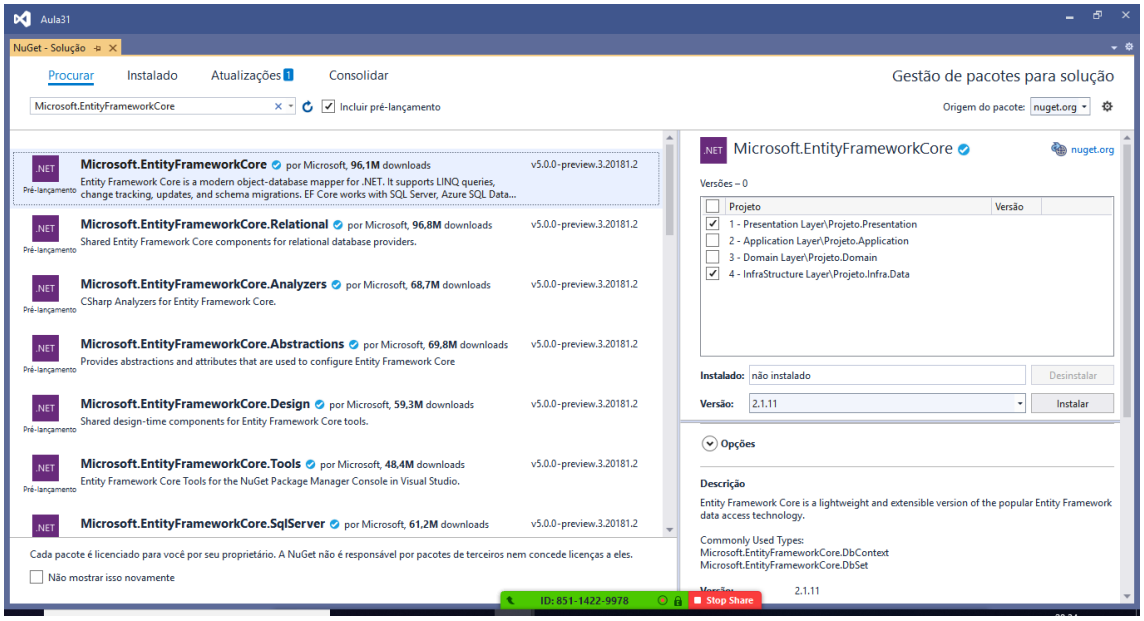

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Domain.Contracts.Repositories
{
    public interface IUnitOfWork
    {
        void BeginTransaction();
        void Commit();
        void Rollback();

        IClienteRepository ClienteRepository { get; }
        IPlanoRepository PlanoRepository { get; }
    }
}
```



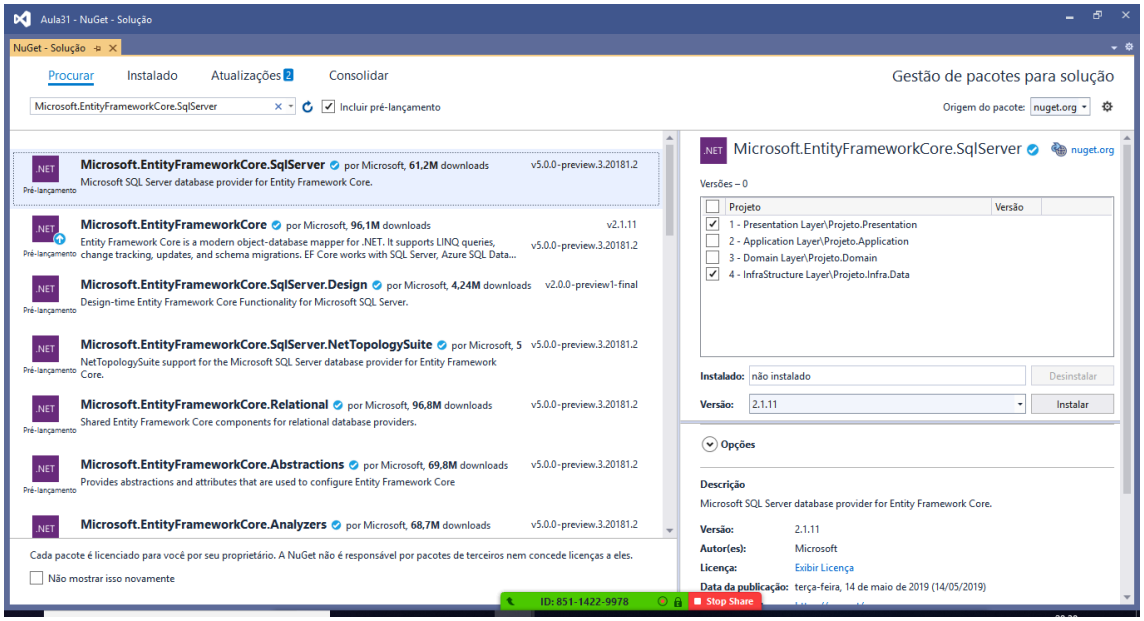
Instalando o EntityFramework

Gerenciador de pacotes do Nuget

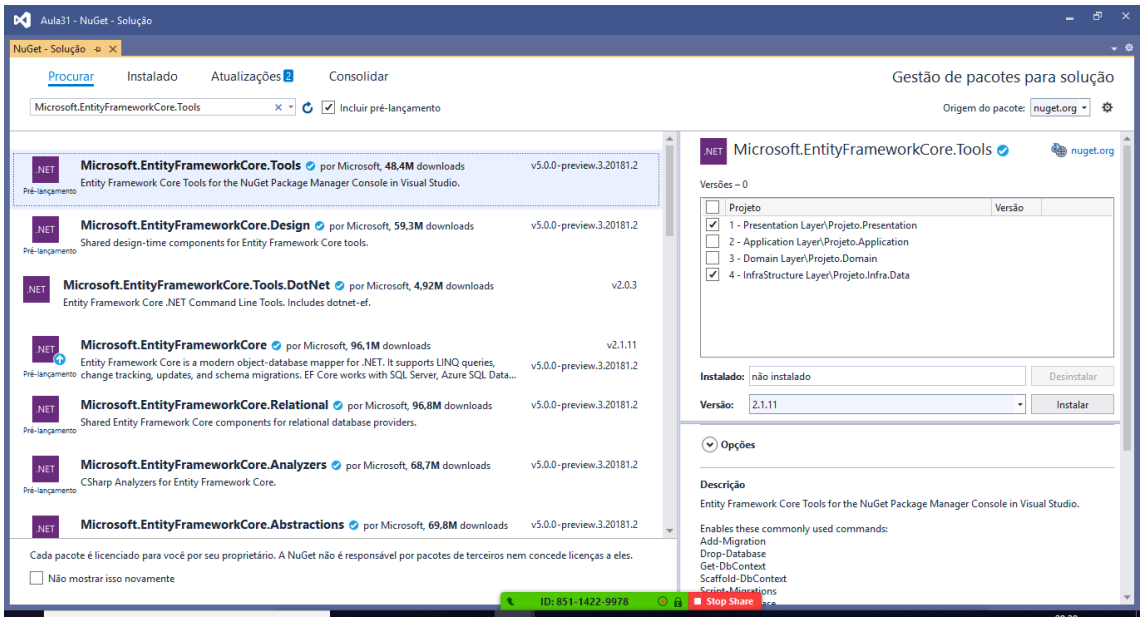
Microsoft.EntityFrameworkCore v.2.1.11



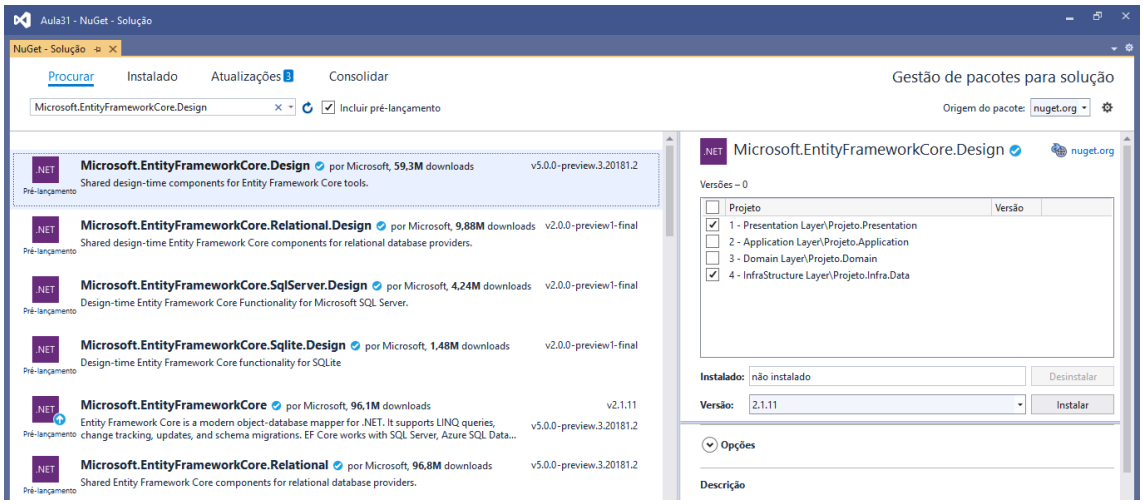
Microsoft.EntityFrameworkCore.SqlServer v.2.1.11



Microsoft.EntityFrameworkCore.Tools v.2.1.11

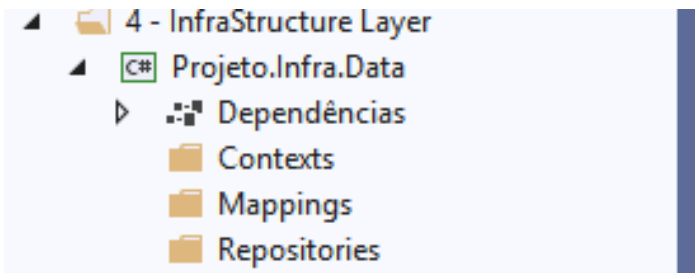


Microsoft.EntityFrameworkCore.Design v.2.1.11



InfraStructure Layer

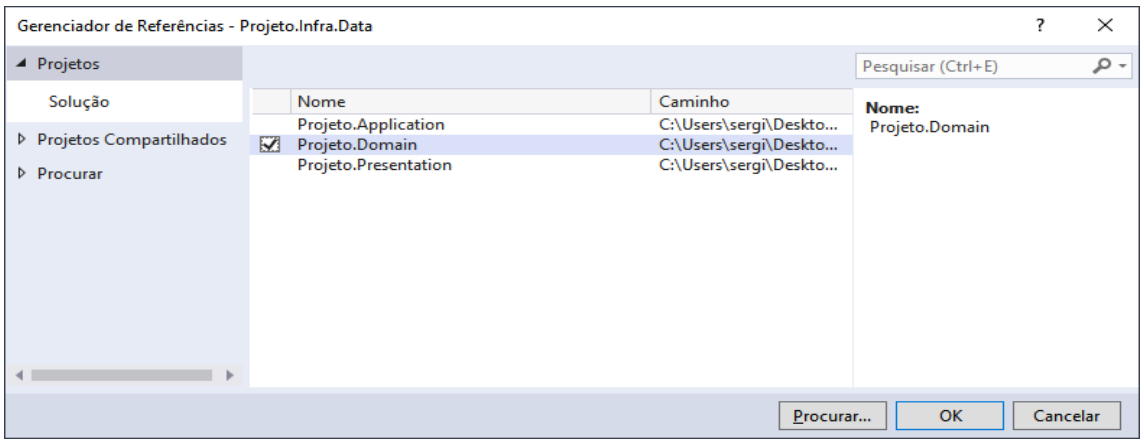
Criando a camada de acesso a banco de dados implementando os contratos de repositório definidos pela camada de domínio.

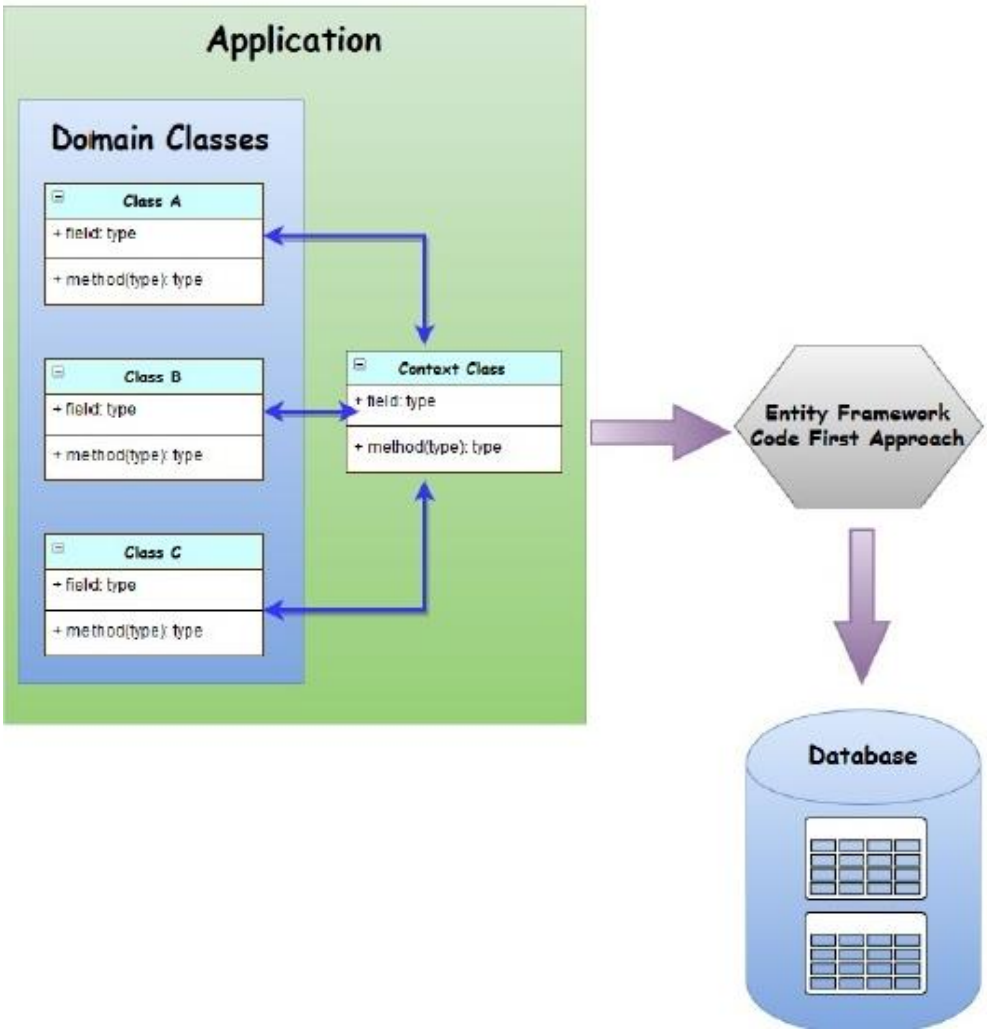


ORM – Object Relational Mapping
Mapeamento Objeto Relacional

Mapear cada classe **Domain Entity** para que possa ser criado
e interpretado como uma tabela do banco de dados (Migrations)

** Adicionar referência no projeto **Infra.Data** para o projeto **Domain**





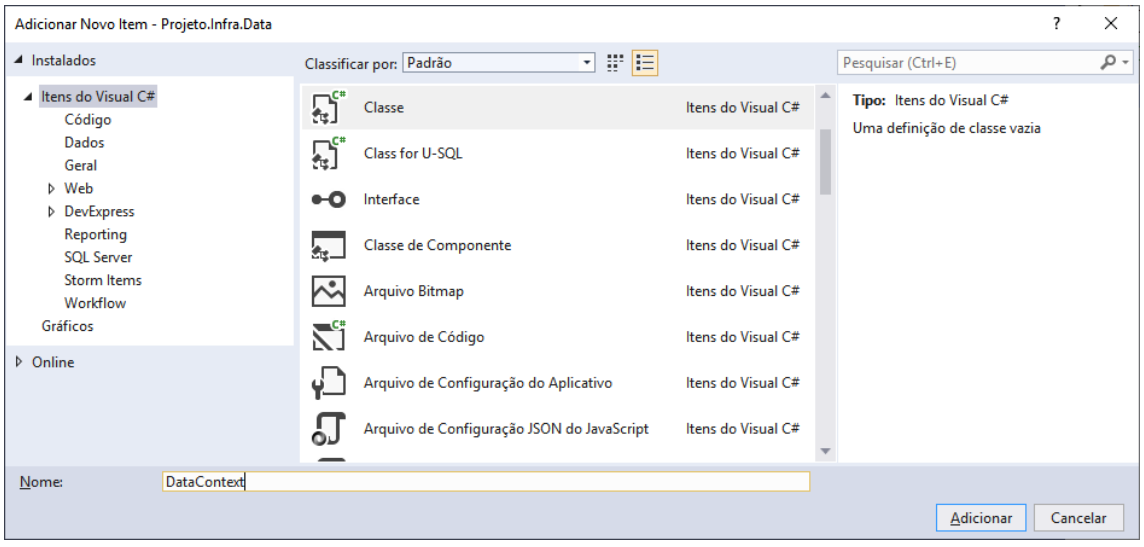
```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Infra.Data.Mappings
{
    public class PlanoMapping : IEntityTypeConfiguration<PlanoEntity>
    {
        public void Configure(EntityTypeBuilder<PlanoEntity> builder)
        {
            //nome da tabela no BD
            builder.ToTable("Plano");

            //chave primária
            builder.HasKey(p => p.IdPlano);

            //mapear todos os campos da tabela
            builder.Property(p => p.IdPlano)
                .HasColumnName("IdPlano");

            builder.Property(p => p.Nome)
                .HasColumnName("Nome")
                .HasMaxLength(150)
                .IsRequired();

            builder.Property(p => p.Descricao)
                .HasColumnName("Descricao")
                .HasMaxLength(1000)
                .IsRequired();
        }
    }
}
```

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Projeto.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Infra.Data.Mappings
{
    public class ClienteMapping : IEntityTypeConfiguration<ClienteEntity>
    {
        public void Configure(EntityTypeBuilder<ClienteEntity> builder)
        {
            builder.ToTable("Cliente");

            builder.HasKey(c => c.IdCliente);

            builder.Property(c => c.IdCliente)
                .HasColumnName("IdCliente");

            builder.Property(c => c.Nome)
                .HasColumnName("Nome")
                .HasMaxLength(150)
                .IsRequired();
```

```
            builder.Property(c => c.Cpf)
                .HasColumnName("Cpf")
                .HasMaxLength(11)
                .IsRequired();

            builder.Property(c => c.DataNascimento)
                .HasColumnName("DataNascimento")
                .HasColumnType("date")
                .IsRequired();

            builder.Property(c => c.IdPlano)
                .HasColumnName("IdPlano")
                .IsRequired();

            #region Relacionamentos

            builder.HasOne(c => c.Plano) //Cliente TEM 1 Plano
                .WithMany(p => p.Clientes) //Plano TEM MUITOS Clientes
                .HasForeignKey(c => c.IdPlano); //Chave estrangeira

            #endregion
        }
    }
}
```

DataContext

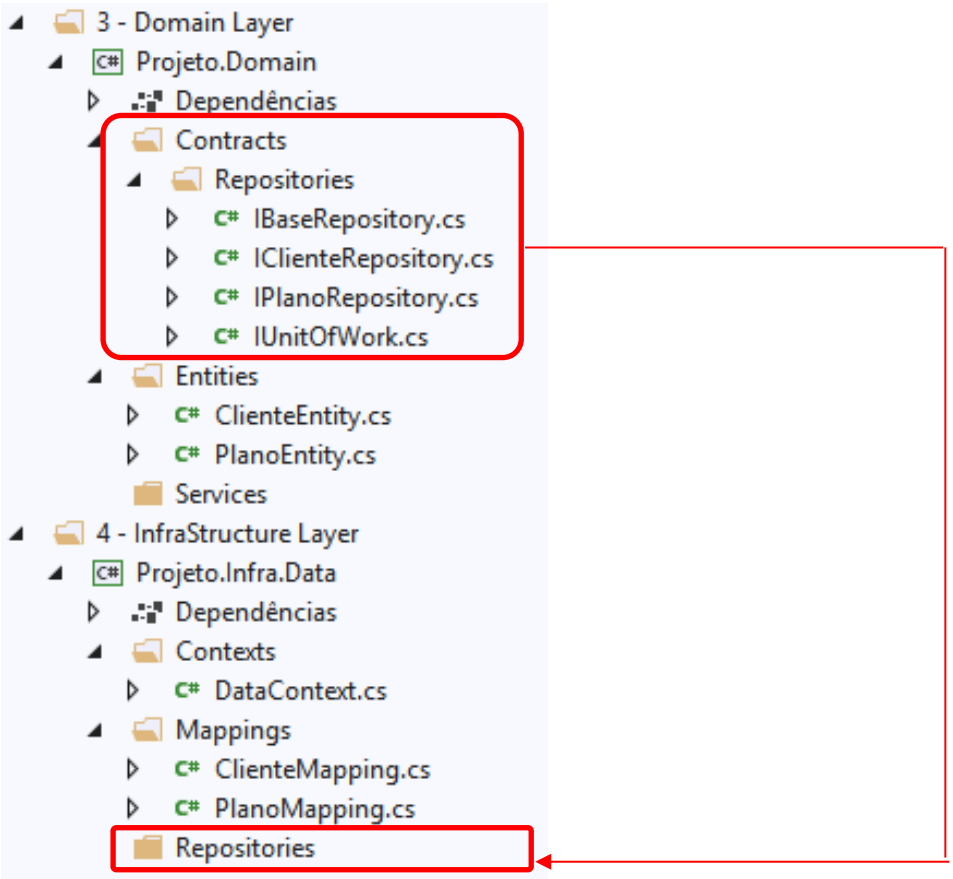
Classe para configurar o acesso ao banco de dados através do EntityFramework.




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

namespace Projeto.Infra.Data.Contexts
{
```

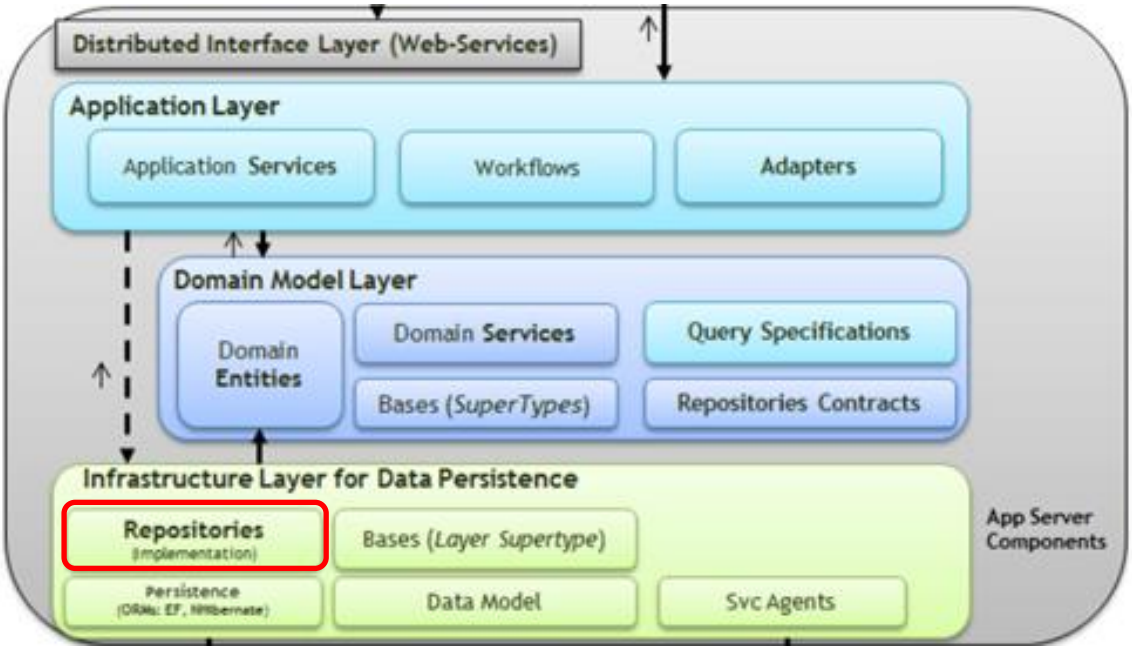
```
    //REGRA 1) HERDAR DbContext!
    public class DataContext : DbContext
    {
        //REGRA 2) Construtor para receber a connectionstring e envia-la
        //para o construtor da superclasse -> DbContext (Injeção de dependência)
        public DataContext(DbContextOptions<DataContext> options)
            : base(options) //construtor da superclasse (DbContext)
        {

        }

        //REGRA 3) Sobrescrever o método OnModelCreating (OVERRIDE)
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new PlanoMapping());
            modelBuilder.ApplyConfiguration(new ClienteMapping());
        }

        //REGRA 4) Declarar um DbSet para cada classe de entidade (LAMBDA)
        public DbSet<PlanoEntity> Plano { get; set; }
        public DbSet<ClienteEntity> Cliente { get; set; }
    }
}
```

Implementando os contratos de repositório definidos no domínio:



Repositories

Implementando as classes de repositório de banco de dados baseado nos contratos de interfaces criados na camada de domínio.



/Repositories/BaseRepository.cs

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

namespace Projeto.Infra.Data.Repositories
{
    public class BaseRepository<TEntity> : IBaseRepository<TEntity>
        where TEntity : class
    {
        //atributo
        private readonly DataContext context;

        //construtor para injeção de dependência
        public BaseRepository(DataContext context)
        {
            this.context = context;
        }

        public virtual void Create(TEntity obj)
        {
            context.Entry(obj).State = EntityState.Added;
            context.SaveChanges();
        }
```

```
        public virtual void Update(TEntity obj)
        {
            context.Entry(obj).State = EntityState.Modified;
            context.SaveChanges();
        }

        public virtual void Delete(TEntity obj)
        {
            context.Entry(obj).State = EntityState.Deleted;
            context.SaveChanges();
        }

        public virtual List<TEntity> GetAll()
        {
            return context.Set<TEntity>().ToList();
        }

        public virtual List<TEntity> GetAll(Func<TEntity, bool> where)
        {
            return context.Set<TEntity>()
                .Where(where)
                .ToList();
        }

        public virtual TEntity Get(Func<TEntity, bool> where)
        {
            return context.Set<TEntity>()
                .FirstOrDefault(where);
        }

        public virtual TEntity GetById(int id)
        {
            return context.Set<TEntity>()
                .Find(id);
        }
    }
}
```

/Repositories/ClienteRepository.cs

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

namespace Projeto.Infra.Data.Repositories
{
    public class ClienteRepository
              : BaseRepository<ClienteEntity>, IClienteRepository
    {
        //atributo
        private readonly DataContext context;

        //construtor para injeção de dependência
        public ClienteRepository(DataContext context)
            : base(context) //construtor da superclasse
        {
            this.context = context;
```

```
        }
    }
}
```

/Repositories/PlanoRepository.cs

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

namespace Projeto.Infra.Data.Repositories
{
    public class PlanoRepository : BaseRepository<PlanoEntity>, IPlanoRepository
    {
        private readonly DataContext context;

        public PlanoRepository(DataContext context)
            : base(context)
        {
            this.context = context;
        }
    }
}
```

/Repositories/UnitOfWork.cs

```
using Microsoft.EntityFrameworkCore.Storage;
using Projeto.Domain.Contracts.Repositories;
using Projeto.Infra.Data.Contexts;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Infra.Data.Repositories
{
    public class UnitOfWork : IUnitOfWork
    {
        //atributos..
        private readonly DataContext context;
        private IDbContextTransaction transaction;

        //construtor para injeção de dependência
        public UnitOfWork(DataContext context)
        {
            this.context = context;
        }

        public void BeginTransaction()
        {
            transaction = context.Database.BeginTransaction();
        }

        public void Commit()
        {
            transaction.Commit();
        }
```

```
        public void Rollback()
        {
            transaction.Rollback();
        }

        public IClienteRepository ClienteRepository
        {
            get { return new ClienteRepository(context); }
        }

        public IPlanoRepository PlanoRepository
        {
            get { return new PlanoRepository(context); }
        }
    }
}
```

Continua...