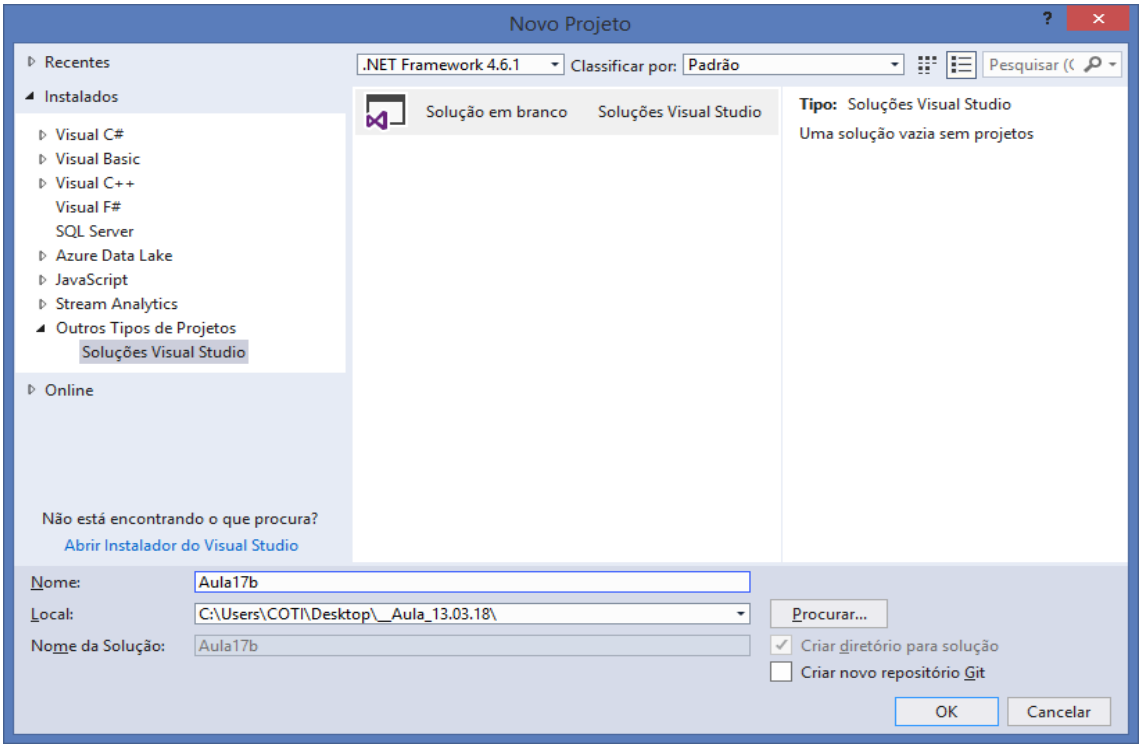
Nova solution em branco:



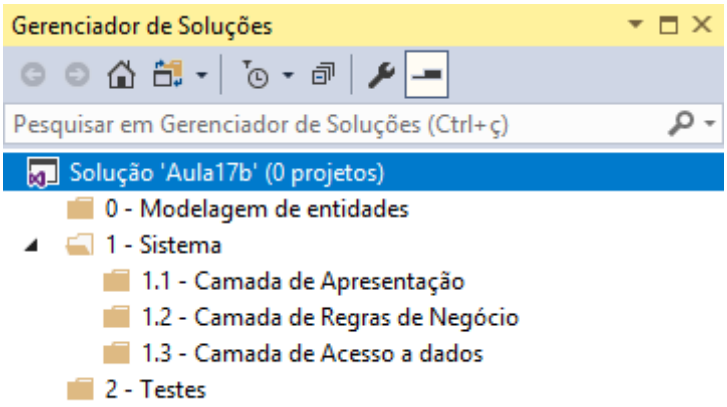
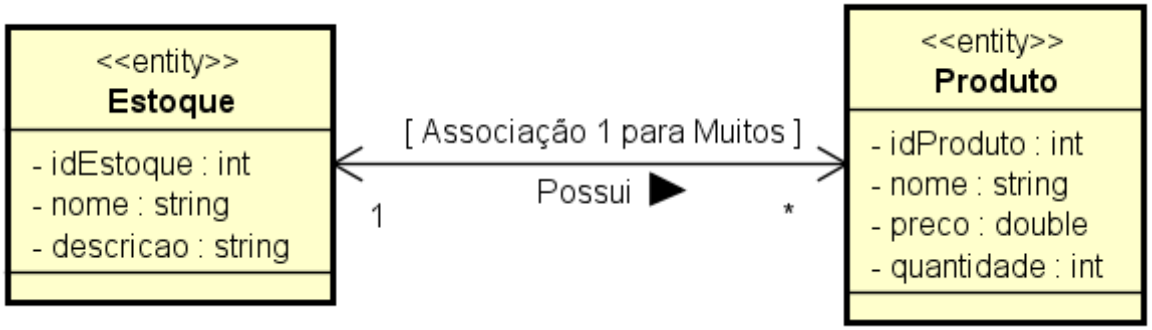


Diagrama de Classes:

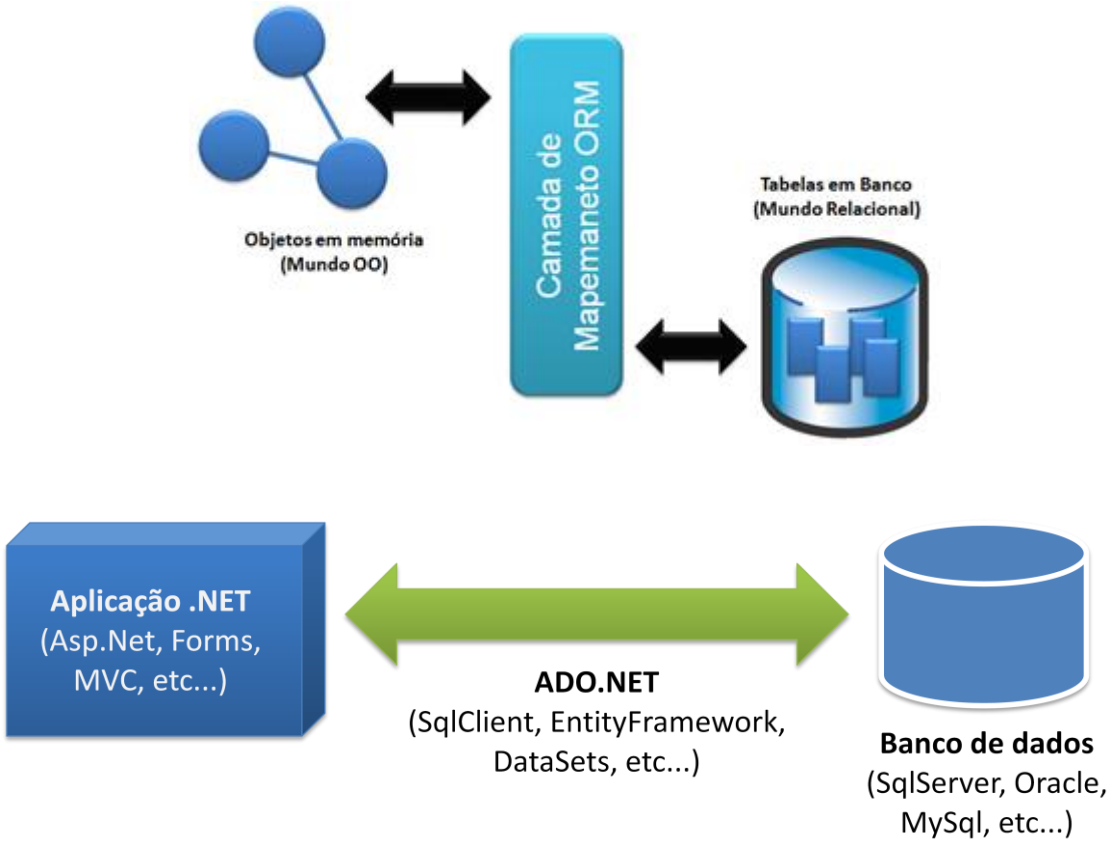


ADO.NET Entity Framework

ADO.NET (ActiveX Data Objects) consiste em um conjunto de bibliotecas definidas pelo .NET framework (localizadas no namespace **System.Data**) que pode ser utilizado para manipular e persistir informações armazenadas numa base de dados remota.

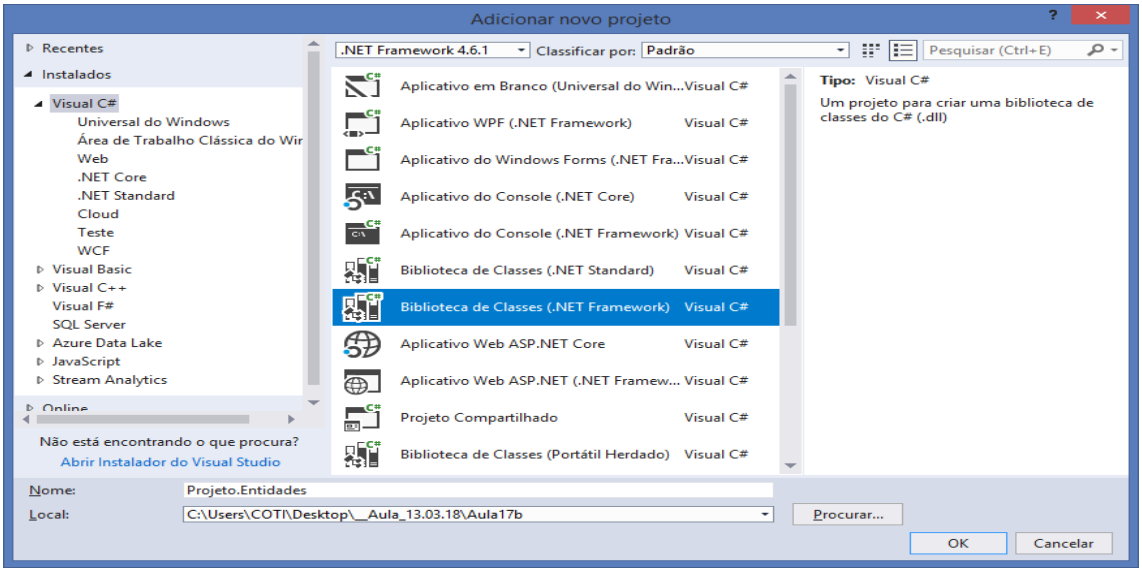
O Microsoft® **ADO.NET Entity Framework** é um framework do tipo ORM (Object/Relational Mapping) que permite aos desenvolvedores trabalhar com dados relacionais como objetos de domínio específico, eliminando a necessidade de maior parte dos códigos de acesso de dados que os desenvolvedores geralmente precisam escrever. Com o Entity Framework, os desenvolvedores podem lançar consultas usando expressões LAMBDA, e depois recuperar e manipular dados como objetos fortemente tipificados.

A implementação do ORM do Entity Framework fornece serviços como rastreamento de alterações, resolução de identidades, lazy loading e tradução de consultas para que os desenvolvedores possam se concentrar na lógica de negócios de seus aplicativos em vez dos princípios básicos de acesso a dados.



0 - Modelagem de entidades

Class Library .NET Framework




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto.Entidades
{
    public class Produto
    {
        public int IdProduto { get; set; }
        public string Nome { get; set; }
        public decimal Preco { get; set; }
        public int Quantidade { get; set; }

        public Estoque Estoque { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto.Entidades
{
    public class Estoque
    {
        public int IdEstoque { get; set; }
        public string Nome { get; set; }
        public string Descricao { get; set; }

        public List<Produto> Produtos { get; set; }
    }
}
```

Para que o EntityFramework possa mapear as classes de entidade, é necessário que as propriedades set e get sejam declaradas com operador **virtual**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto.Entidades
{
    public class Estoque
    {
        public virtual int IdEstoque { get; set; }
        public virtual string Nome { get; set; }
        public virtual string Descricao { get; set; }

        public virtual List<Produto> Produtos { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto.Entidades
{
    public class Produto
    {
        public virtual int IdProduto { get; set; }
        public virtual string Nome { get; set; }
        public virtual decimal Preco { get; set; }
        public virtual int Quantidade { get; set; }

        public virtual Estoque Estoque { get; set; }
    }
}
```

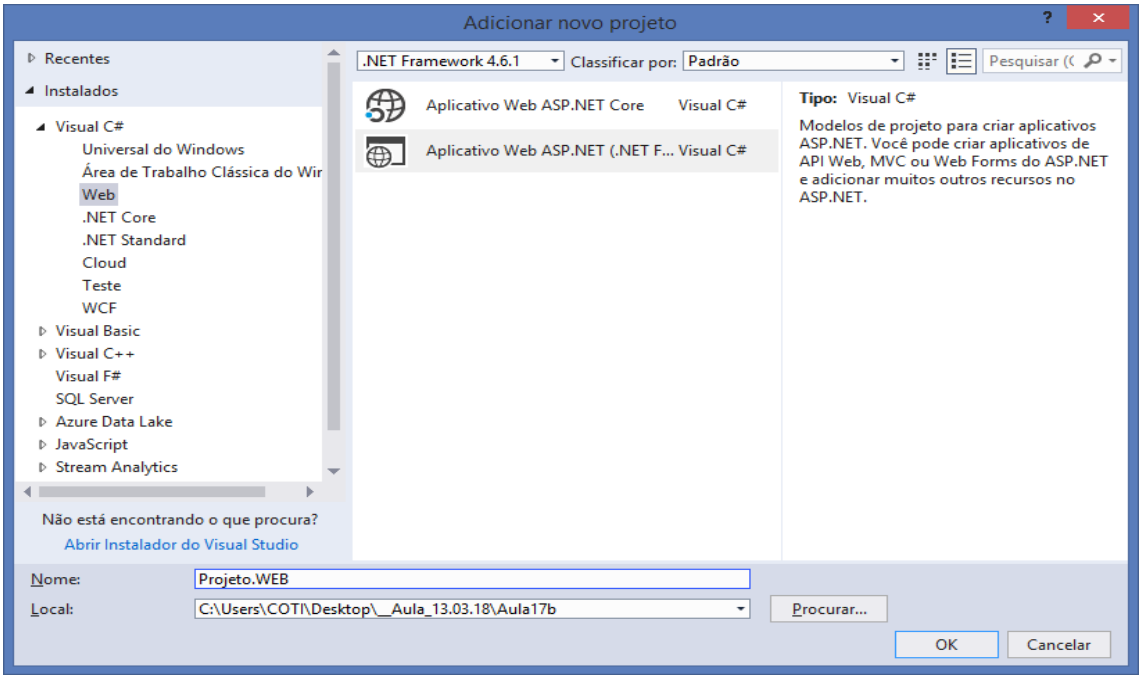
Regra: Se uma entidade no banco de dados conter foreign key, será necessário declarar na classe uma propriedade para esta foreign key

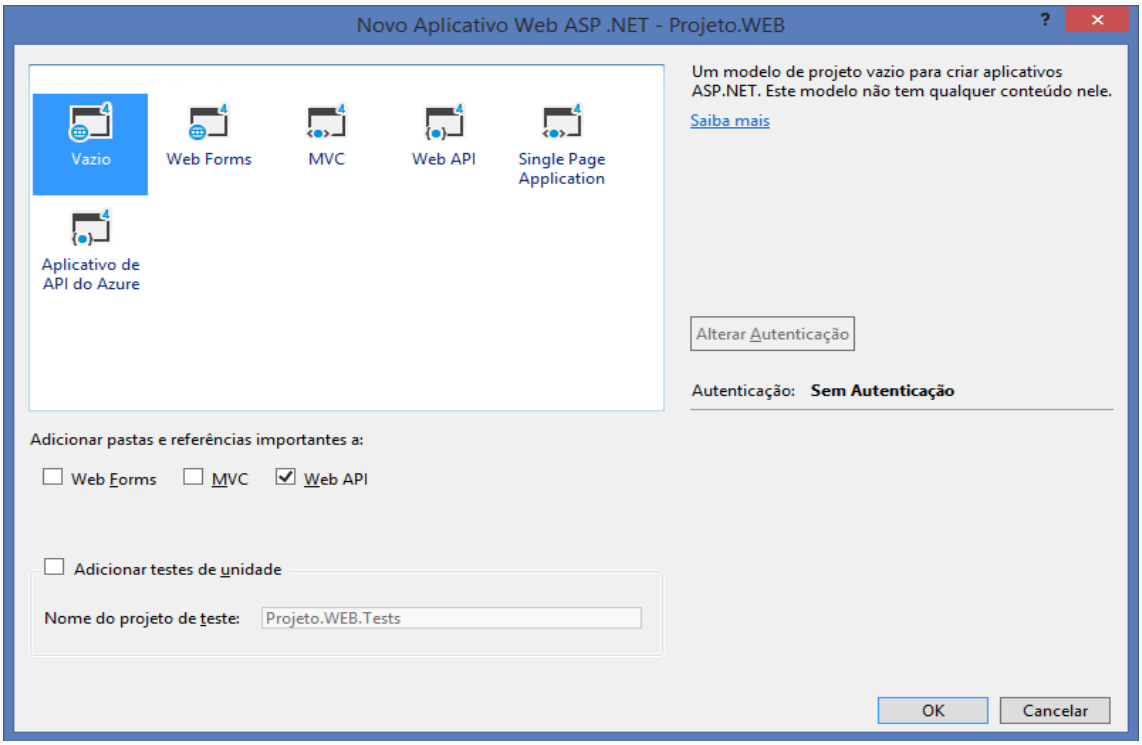
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto.Entidades
{
    public class Produto
    {
        public virtual int IdProduto { get; set; }
        public virtual string Nome { get; set; }
```

```
        public virtual decimal Preco { get; set; }
        public virtual int Quantidade { get; set; }
        public virtual int IdEstoque { get; set; }

        public virtual Estoque Estoque { get; set; }
    }
}
```

1.1 - Camada de Apresentação

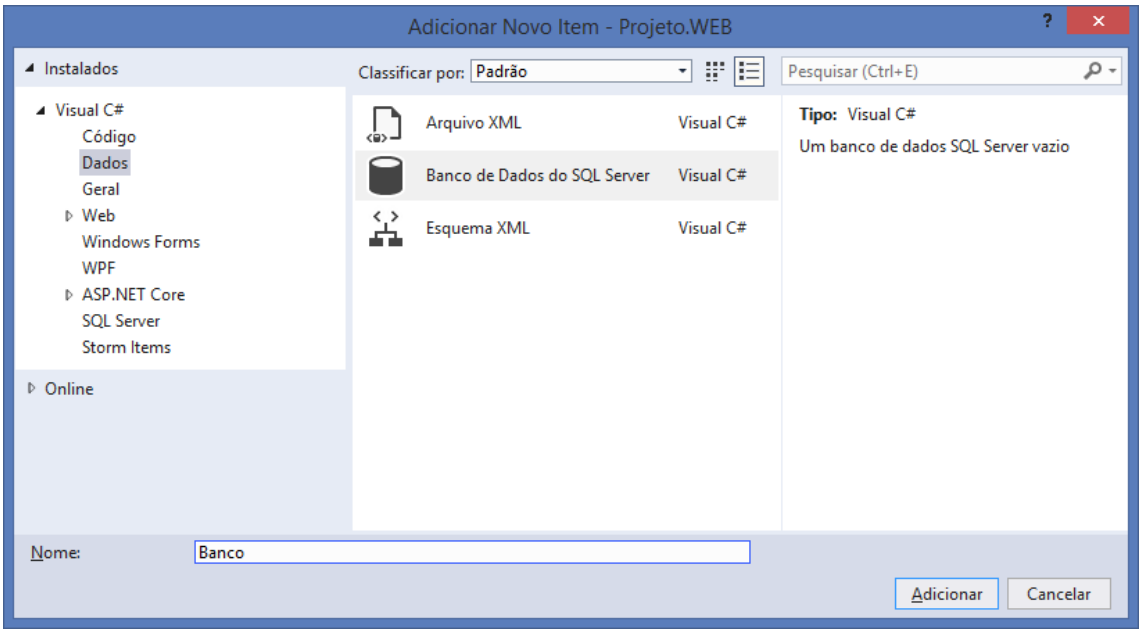
Asp.Net WebApi





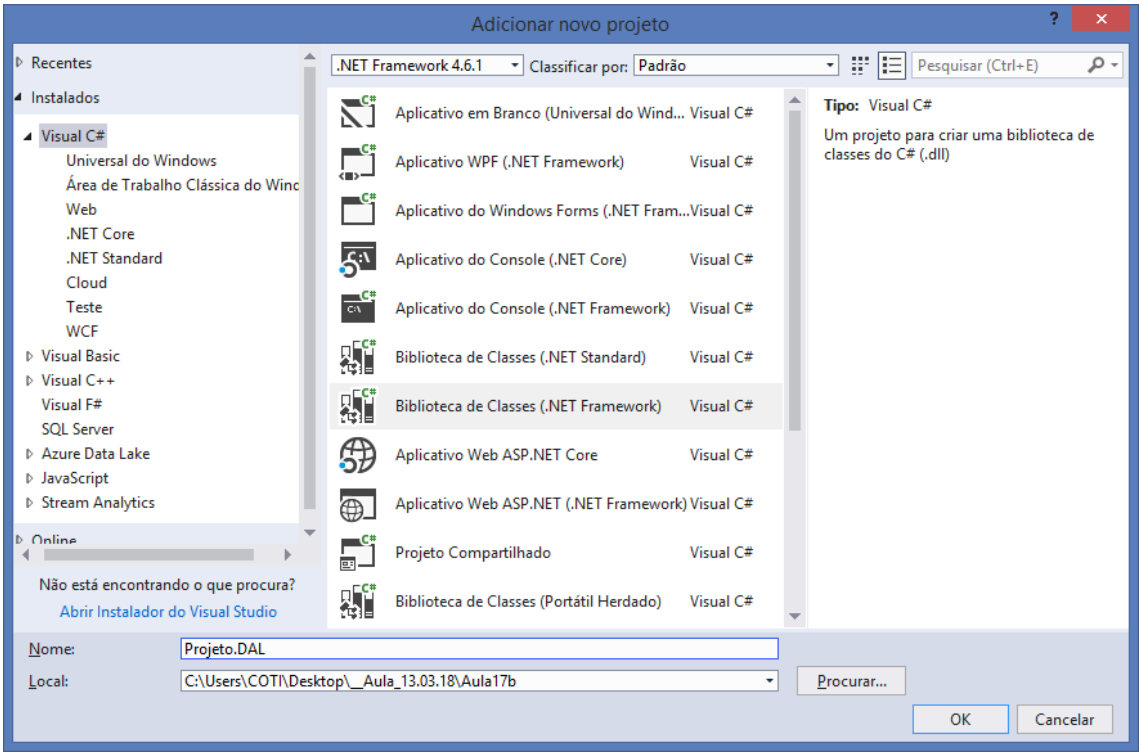
Criando a base de dados:
MDF - Master Database File

/App_Data/Banco.mdf



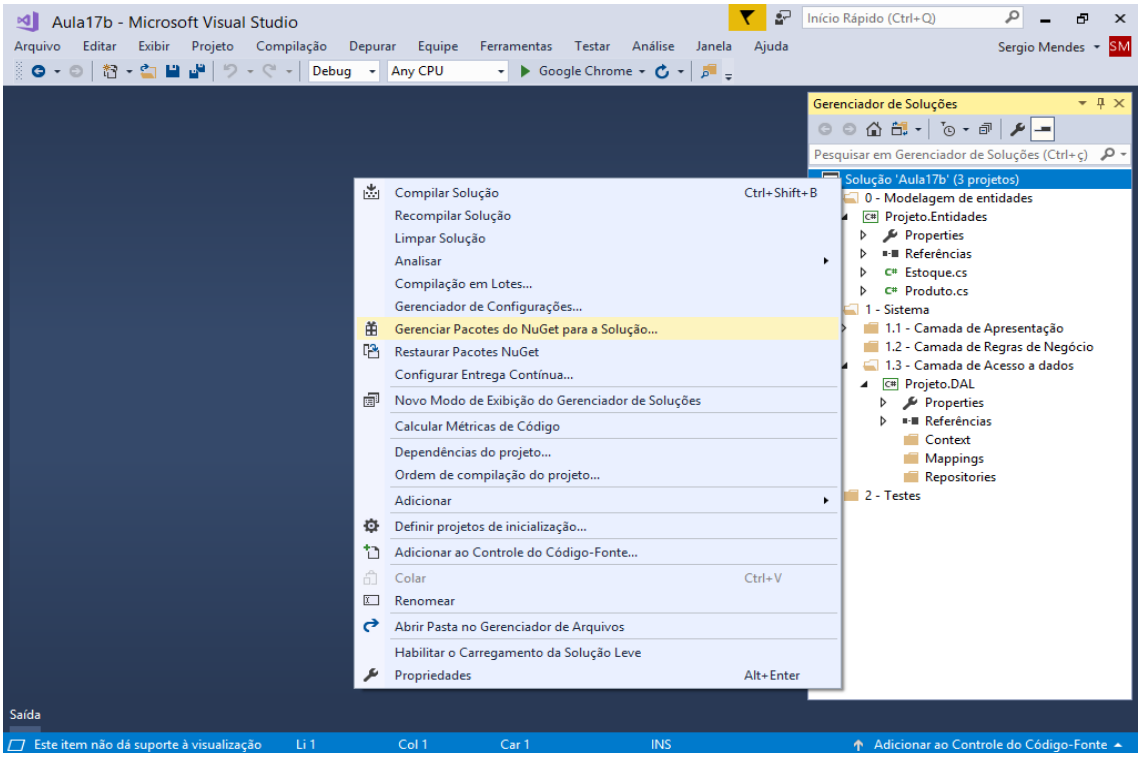
1.3 - Camada de Acesso a Dados

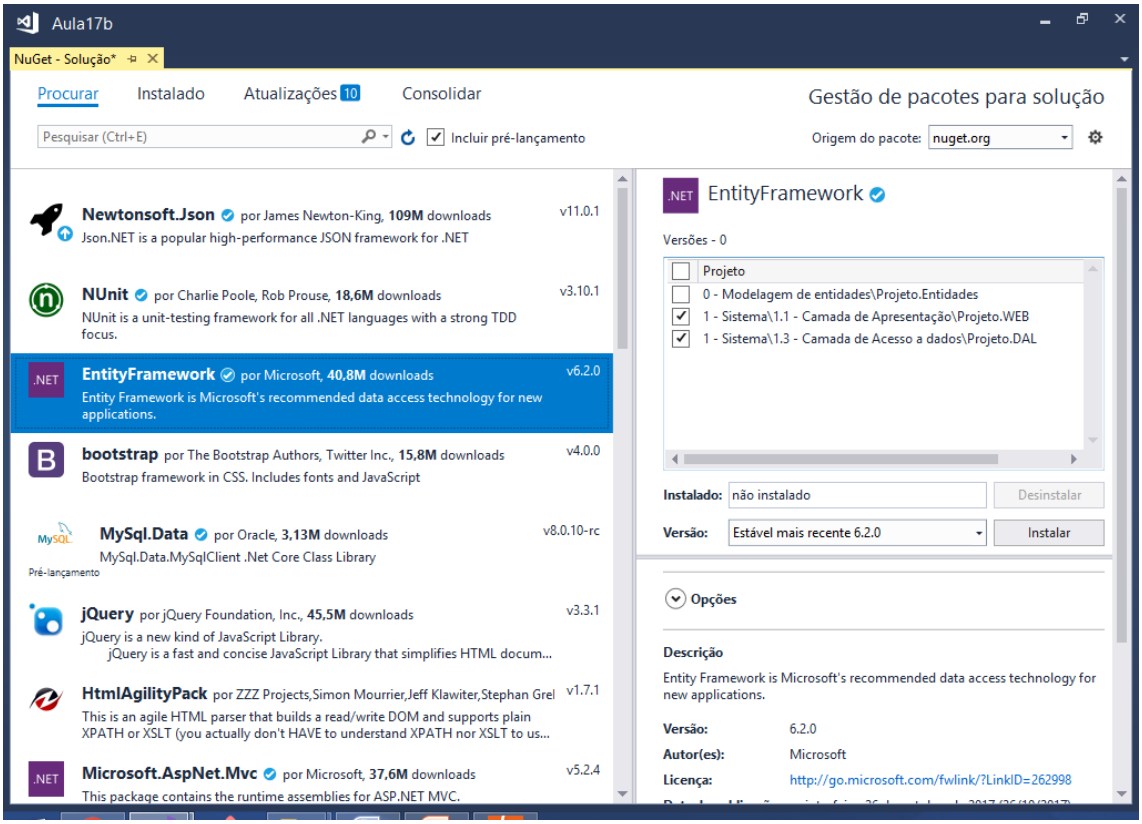
DAL - Data Access Layer



Instalando o EntityFramework

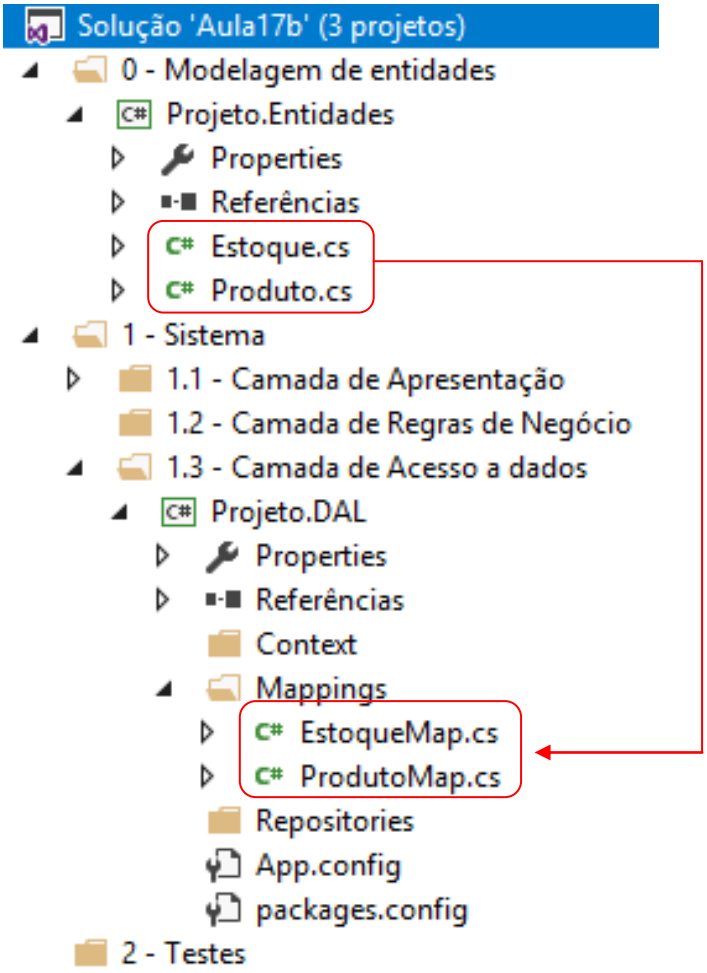
Observação: O EntityFramework deverá ser instalado no projeto DAL, mas tambem no projeto Asp.Net (ConnectionString)



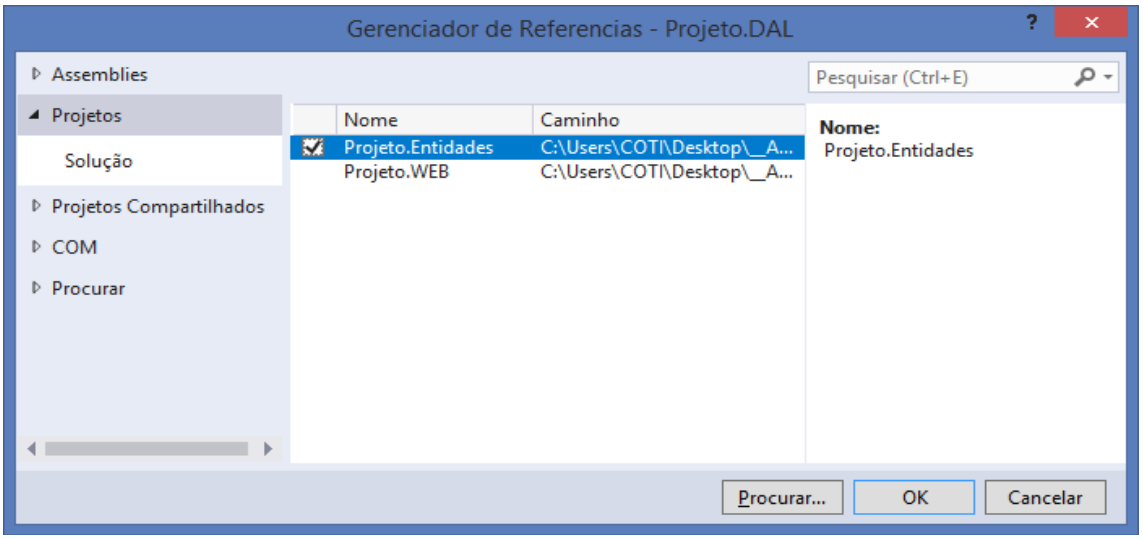


ORM - Mapeamento Objeto Relacional

Mapear cada classe de entidade para estas sejam interpretadas pelo EntityFramework como tabelas do banco de dados.



Adicionando referencia no projeto DAL para o projeto Entidades:

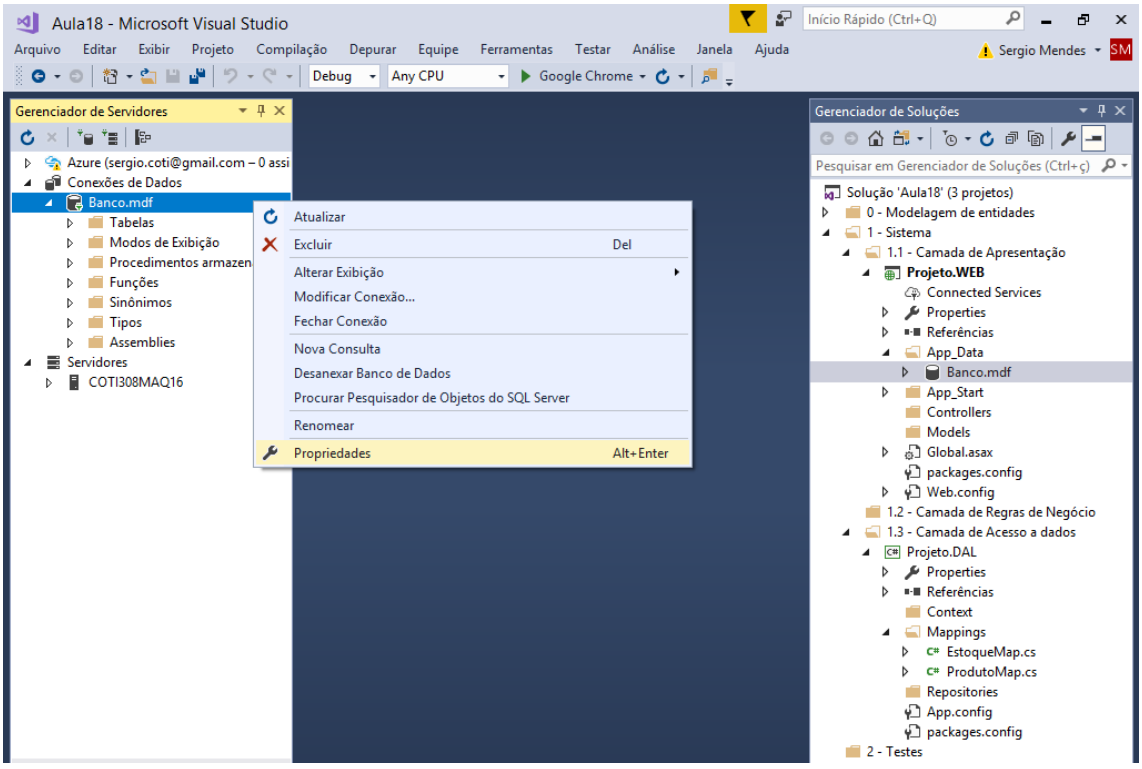


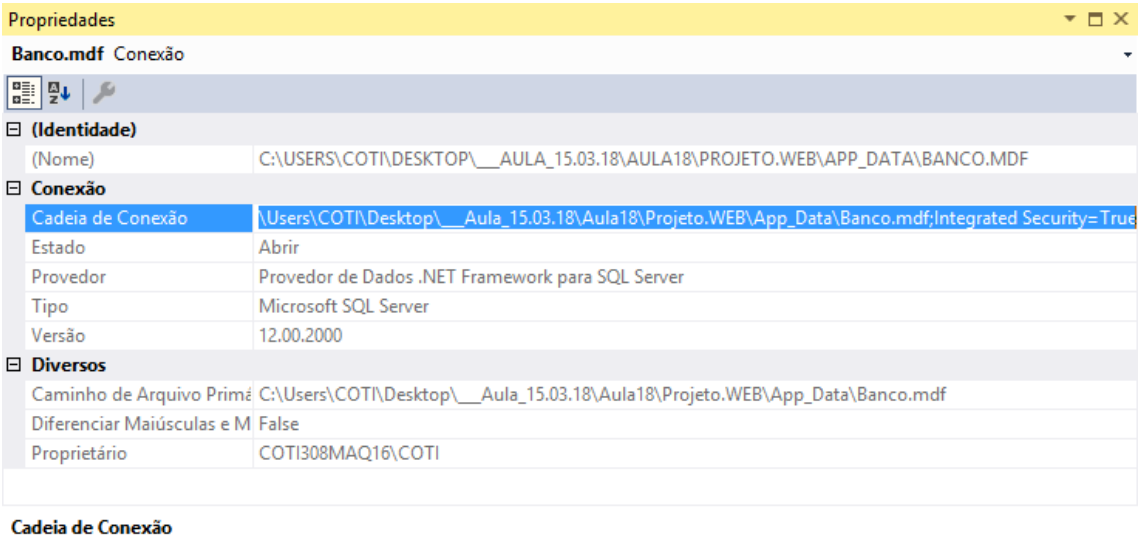
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto.Entidades; //classes de entidade..
using System.Data.Entity.ModelConfiguration; //mapeamento..

namespace Projeto.DAL.Mappings
{
    //classe de mapeamento para a entidade Estoque..
    public class EstoqueMap : EntityTypeConfiguration<Estoque>
    {
        //construtor [ctor] + 2x[tab]
        public EstoqueMap()
        {
            //nome da tabela..
            ToTable("Estoque");

            //chave primária..
            HasKey(e => e.IdEstoque);

            //mapear os campos..
            Property(e => e.IdEstoque)
                .HasColumnName("IdEstoque")
                .IsRequired();

            Property(e => e.Nome)
                .HasColumnName("Nome")
                .HasMaxLength(50)
                .IsRequired();

            Property(e => e.Descricao)
                .HasColumnName("Descricao")
                .HasMaxLength(250)
                .IsRequired();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto.Entidades; //classes de entidade..
using System.Data.Entity.ModelConfiguration; //mapeamento..

namespace Projeto.DAL.Mappings
{
    //classe de mapeamento para a entidade Produto
    public class ProdutoMap : EntityTypeConfiguration<Produto>
    {
        //construtor [ctor] + 2x[tab]
        public ProdutoMap()
        {
            //nome da tabela..
            ToTable("Produto");
```

```
            //chave primária..
            HasKey(p => p.IdProduto);

            //campos da tabela..
            Property(p => p.IdProduto)
                .HasColumnName("IdProduto")
                .IsRequired();

            Property(p => p.Nome)
                .HasColumnName("Nome")
                .HasMaxLength(50)
                .IsRequired();

            Property(p => p.Preco)
                .HasColumnName("Preco")
                .HasPrecision(18,2)
                .IsRequired();

            Property(p => p.Quantidade)
                .HasColumnName("Quantidade")
                .IsRequired();

            //mapear o relacionamento
            //cardinalidade 1 para muitos..
            HasRequired(p => p.Estoque) //Produto TEM 1 Estoque
                .WithMany(e => e.Produtos) //Estoque TEM MUITOS Produtos
                .HasForeignKey(p => p.IdEstoque); //Chave Estrangeira
        }
    }
}
```

Mapeando a connectionstring do banco de dados



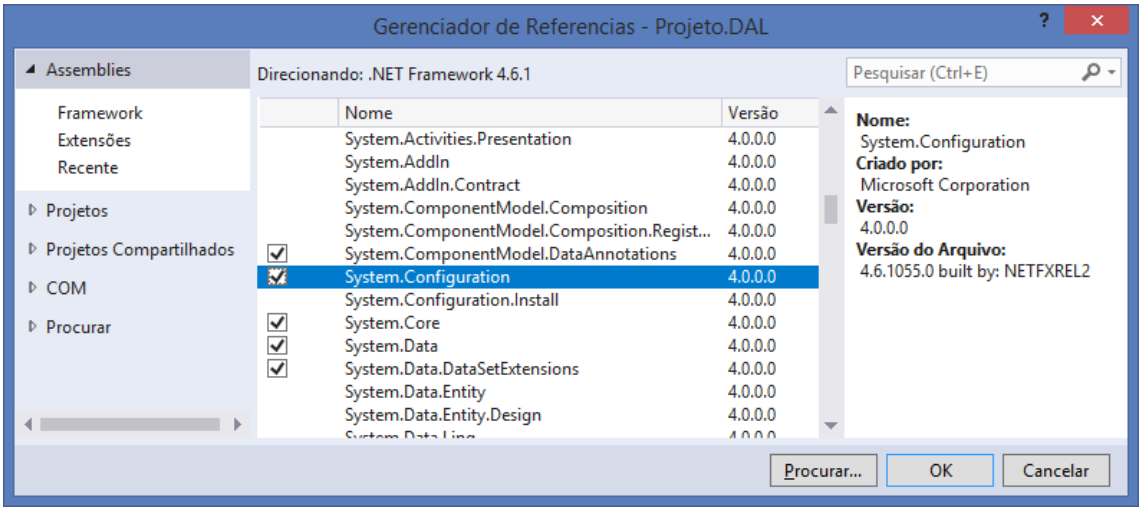


\Web.config.xml

Configurando a string de conexão..

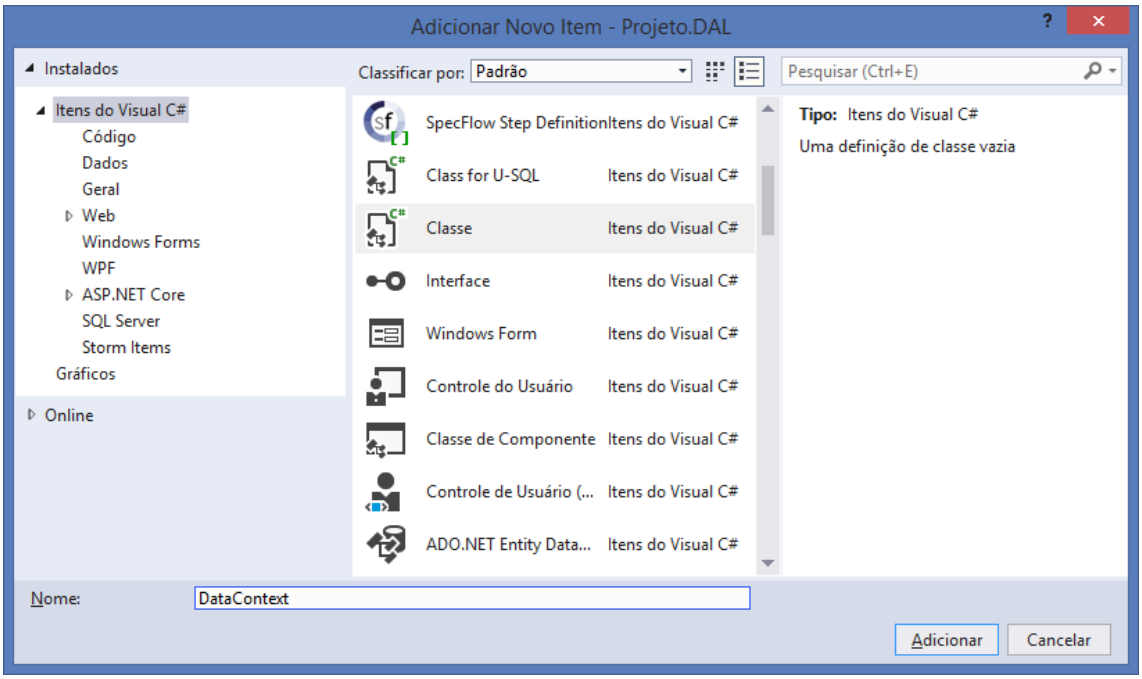
```
<!-- mapeamento da connectionstring.. -->
<connectionStrings>
  <add
    name="aula18"
    connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
     AttachDbFilename=C:\Users\COTI\Desktop\___Aula_15.03.18\
     Aula18\Projeto.WEB\App_Data\Banco.mdf;Integrated Security=True"
  />
</connectionStrings>
```

Adicionando referencia no projeto DAL para **System.Configuration**



Classe de conexão com o banco de dados em EntityFramework

Geramente esta classe é chamada de "**DataContext**"



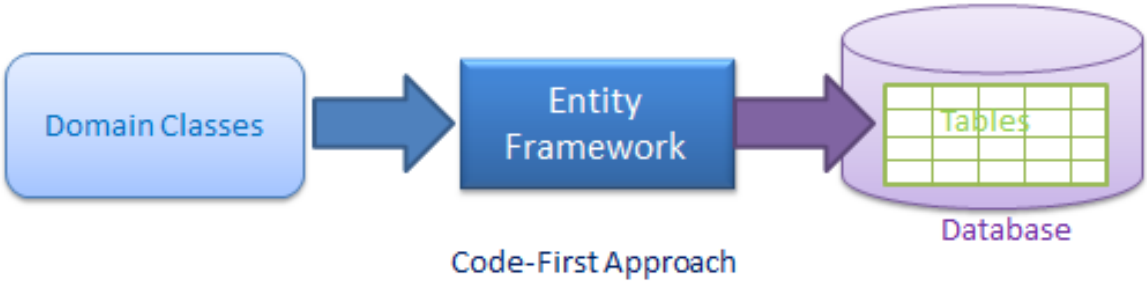

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity; //entity framework..
using System.Configuration; //connectionstring..
using Projeto.Entidades; //classes de entidade..
using Projeto.DAL.Mappings; //classes de mapeamento..

namespace Projeto.DAL.Context
{
    //Regra 1) Herdar DbContext
    public class DataContext : DbContext
    {
        //Regra 2) Declarar o construtor da classe e atraves dele, enviar
        //para o construtor da superclasse (DbContext) o caminho da
        //connectionstring
        public DataContext()
            : base(ConfigurationManager.ConnectionStrings
                    ["aula18"].ConnectionString)
        {

        }
```

```
        //Regra 3) Sobrescrever o método 'OnModelCreating'
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            //adicionar as classes de mapeamento do projeto..
            modelBuilder.Configurations.Add(new EstoqueMap());
            modelBuilder.Configurations.Add(new ProdutoMap());
        }

        //Regra 4) Declarar um set/get para cada entidade usando
        //a classe DbSet do EntityFramework..
        public DbSet<Estoque> Estoque { get; set; }
        public DbSet<Produto> Produto { get; set; }
    }

}
```

Code First

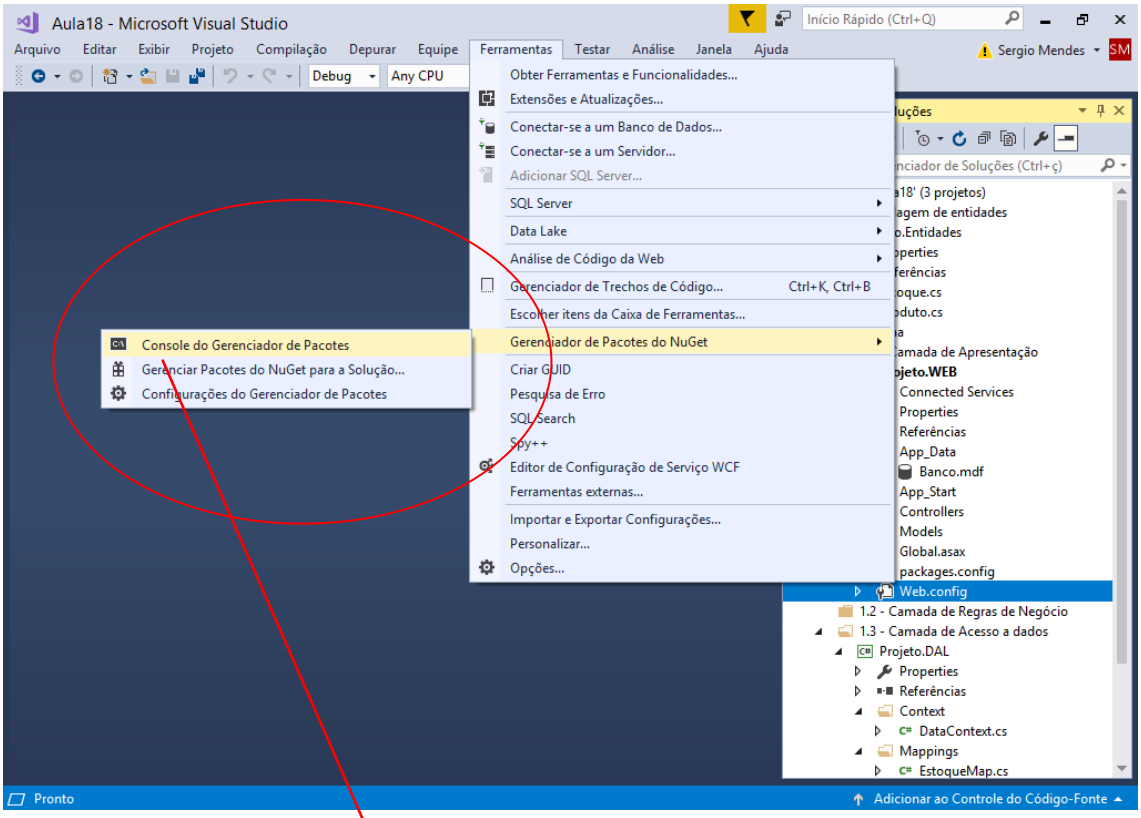
Forma de trabalho difundida pelo EntityFramework
que proproe para uma aplicação as seguintes práticas:

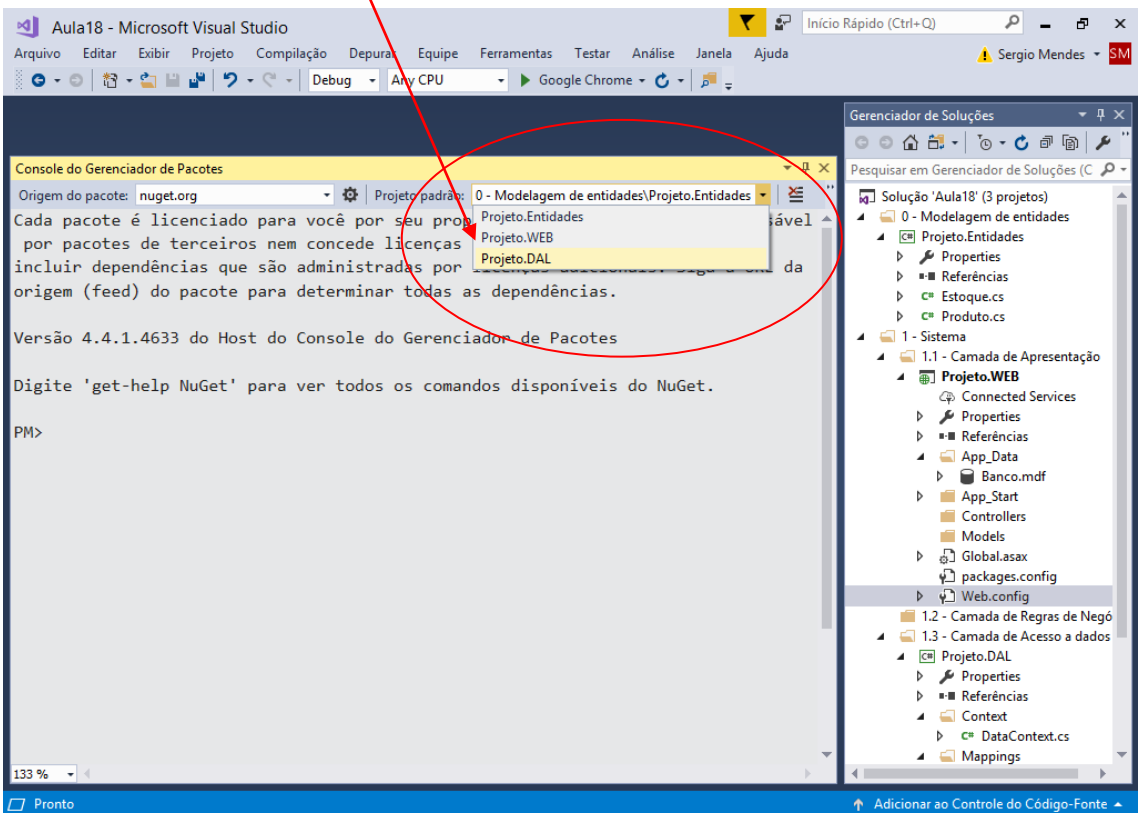
- Modelar e escrever as classes de entidade do projeto
- Mapear cada classe de entidade conforme a sua respectiva tabela (ORM - Object Relacional Mapping)
- Configurar um framework de acesso a banco de dados com suporte a ORM (Entity Framework)
- Gerar as tabelas no banco de dados (Script SQL) baseado no mapeamento
- Qualquer alteração no modelo de dados é feita primeiro na classe e depois conforme o mapeamento, o framework é quem faz a alteração na base de dados.



Migrations

Ferramenta do EntityFramework que
permite realizar o CodeFirst no projeto.





Habilitando o Migrations do EntityFramework

PM> enable-migrations -force

```
PM> enable-migrations -force
Checking if the context targets an existing database...
Code First Migrations enabled for project Projeto.DAL.
PM>
```

Após a execução do comando acima, será criado no projeto a classe: /Migrations/Configuration.cs

```
namespace Projeto.DAL.Migrations
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;

    internal sealed class Configuration : DbMigrationsConfiguration
                          <Projeto.DAL.Context.DataContext>
    {
        public Configuration()
        {
            //permissão de CREATE e ALTER..
            AutomaticMigrationsEnabled = true;

            //permissão de DROP..
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(Projeto.DAL.Context.DataContext context)
        {
            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method
            //  to avoid creating duplicate seed data.
        }
    }
}
```

Gerando um script com o codigo SQL que o entity framework irá executar no banco de dados.

PM> update-database -script

```
No pending explicit migrations.
Applying automatic migration: 201803152253563_AutomaticMigration.
PM>
```

```
CREATE TABLE [dbo].[Estoque] (
    [IdEstoque] [int] NOT NULL IDENTITY,
    [Nome] [nvarchar](50) NOT NULL,
    [Descricao] [nvarchar](250) NOT NULL,
    CONSTRAINT [PK_dbo.Estoque] PRIMARY KEY ([IdEstoque])
)

CREATE TABLE [dbo].[Produto] (
    [IdProduto] [int] NOT NULL IDENTITY,
    [Nome] [nvarchar](50) NOT NULL,
    [Preco] [decimal](18, 2) NOT NULL,
    [Quantidade] [int] NOT NULL,
    [IdEstoque] [int] NOT NULL,
    CONSTRAINT [PK_dbo.Produto] PRIMARY KEY ([IdProduto])
)

CREATE INDEX [IX_IdEstoque] ON [dbo].[Produto]([IdEstoque])
ALTER TABLE [dbo].[Produto] ADD CONSTRAINT
[FK_dbo.Produto_dbo.Estoque_IdEstoque] FOREIGN KEY ([IdEstoque])
REFERENCES [dbo].[Estoque] ([IdEstoque]) ON DELETE CASCADE

CREATE TABLE [dbo].[__MigrationHistory] (
    [MigrationId] [nvarchar](150) NOT NULL,
    [ContextKey] [nvarchar](300) NOT NULL,
    [Model] [varbinary](max) NOT NULL,
    [ProductVersion] [nvarchar](32) NOT NULL,
    CONSTRAINT [PK_dbo.__MigrationHistory] PRIMARY KEY ([MigrationId],
[ContextKey])
)
```

Executando no banco de dados:

```
PM> update-database -verbose
```

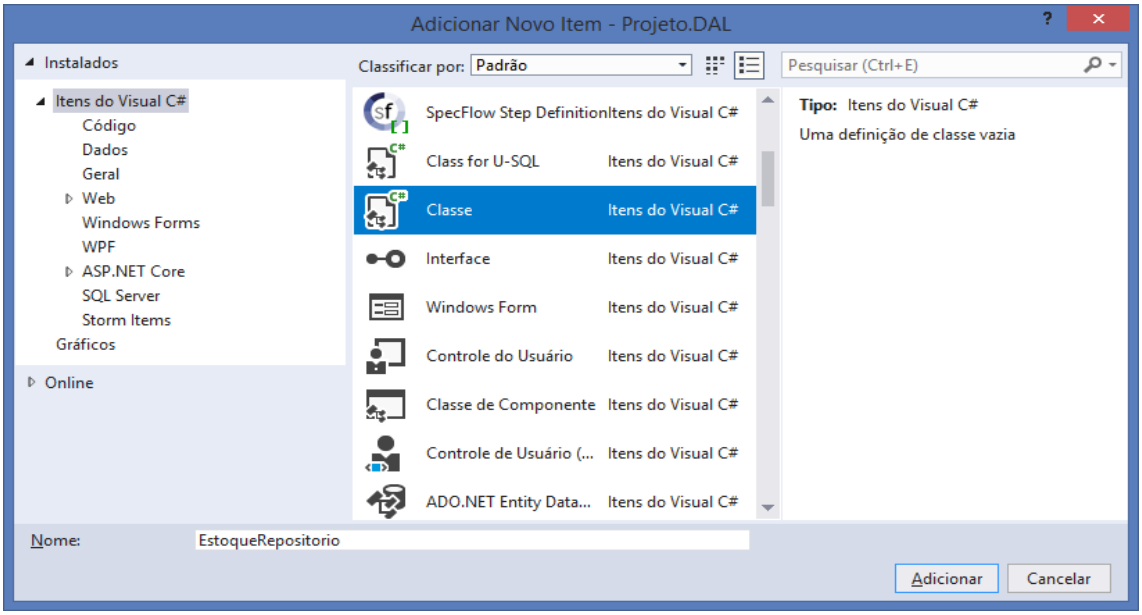
Repositorio Generico

Criar uma classe que irá implementar os métodos INSERT, UPDATE, DELETE e SELECT para qualquer entidade mapeada no projeto.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity; //entityframework
using Projeto.DAL.Context; //classe de acesso ao BD

namespace Projeto.DAL.Repositories
{
```

```
    public class GenericRepositorio<T>
        where T : class
    {
        //método para inserir um registro na base de dados..
        public virtual void Insert(T obj)
        {
            //instanciar a classe DataContext..
            using (DataContext d = new DataContext())
            {
                d.Entry(obj).State = EntityState.Added; //insert..
                d.SaveChanges(); //executando..
            }
        }

        //método para atualizar um registro na base de dados..
        public virtual void Update(T obj)
        {
            //instanciar a classe DataContext..
            using (DataContext d = new DataContext())
            {
                d.Entry(obj).State = EntityState.Modified; //update..
                d.SaveChanges(); //executando..
            }
        }

        //método para excluir um registro na base de dados..
        public virtual void Delete(T obj)
        {
            //instanciar a classe DataContext..
            using (DataContext d = new DataContext())
            {
                d.Entry(obj).State = EntityState.Deleted; //delete..
                d.SaveChanges(); //executando..
            }
        }

        //método para listar todos os registros..
        public virtual List<T> FindAll()
        {
            using (DataContext d = new DataContext())
            {
                return d.Set<T>().ToList();
            }
        }

        //método para obter 1 registro do BD..
        public virtual T FindById(int id)
        {
            using (DataContext d = new DataContext())
            {
                return d.Set<T>().Find(id);
            }
        }

    }

}
```

Criando os demais repositorios:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;
using Projeto.Entidades;
using Projeto.DAL.Context;

namespace Projeto.DAL.Repositories
{
    public class EstoqueRepositorio : GenericRepositorio<Estoque>
    {

    }
}
```

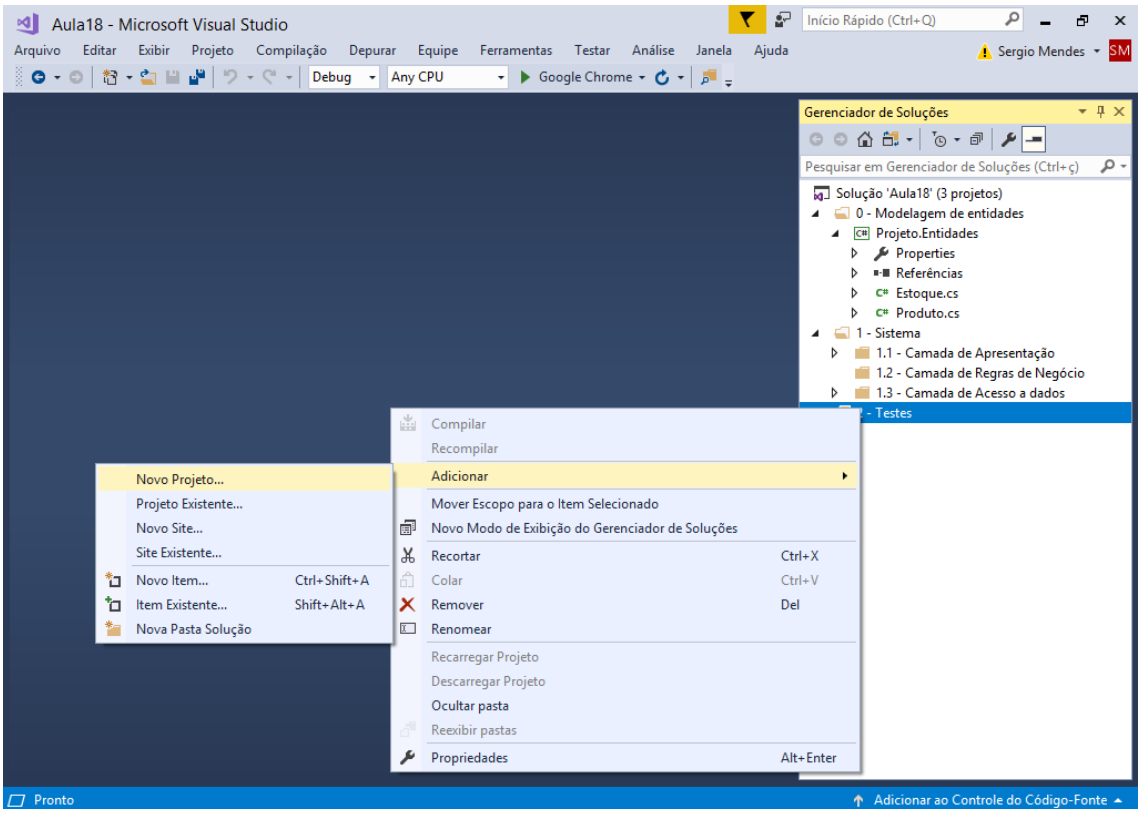
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;
using Projeto.Entidades;
using Projeto.DAL.Context;

namespace Projeto.DAL.Repositories
{
    public class ProdutoRepositorio : GenericRepositorio<Produto>
    {

    }
}
```

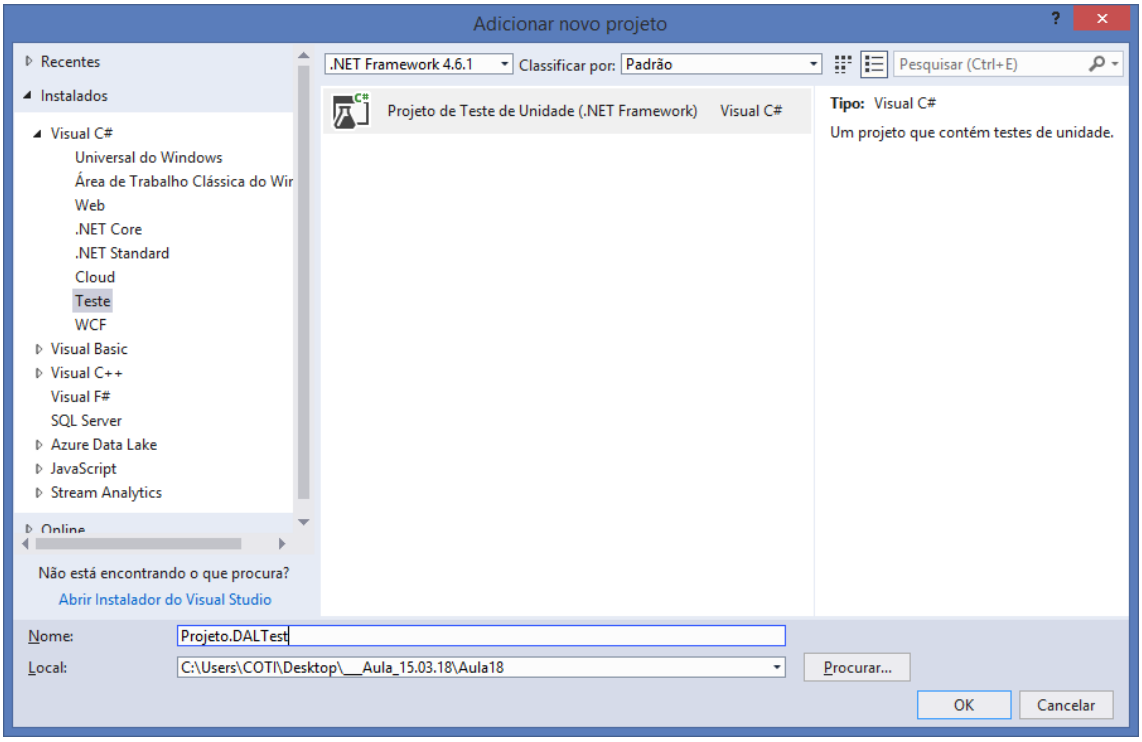
Sobrescrevendo métodos da classe Genérica e criando consultas com LAMBDA:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;
using Projeto.Entidades;
using Projeto.DAL.Context;

namespace Projeto.DAL.Repositories
{
    public class ProdutoRepositorio : GenericRepositorio<Produto>
    {
        public override List<Produto> FindAll()
        {
            using (DataContext d = new DataContext())
            {
                return d.Produto //consulta na tabela de produto..
                        .Include(p => p.Estoque) //inner join..
                        .ToList(); //retornando uma lista..
            }
        }

        //método para retornar produtos pelo nome..
        public List<Produto> FindByNome(string nome)
        {
            using (DataContext d = new DataContext())
            {
                return d.Produto
                        .Include(p => p.Estoque)
                        .Where(p => p.Nome.Contains(nome))
                        .OrderBy(p => p.Nome)
                        .ToList();
            }
        }

        //método para retornar produtos pelo preço..
        public List<Produto> FindByPreco(decimal precoIni, decimal precoFim)
        {
            using (DataContext d = new DataContext())
            {
                return d.Produto
                        .Include(p => p.Estoque)
                        .Where(p => p.Preco >= precoIni && p.Preco <= precoFim)
                        .OrderByDescending(p => p.Preco)
                        .ToList();
            }
        }
    }
}
```

UnitTest

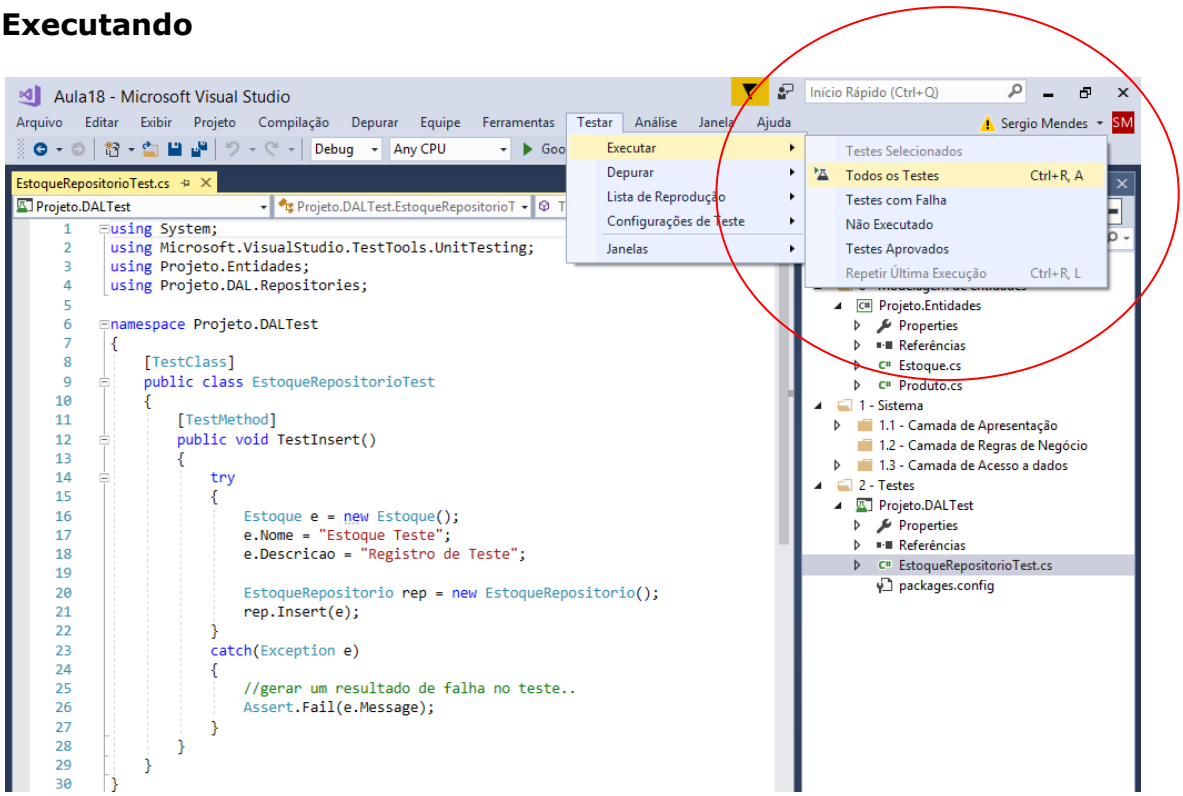
Biblioteca do .NET voltado para implementação de rotinas de teste unitário.



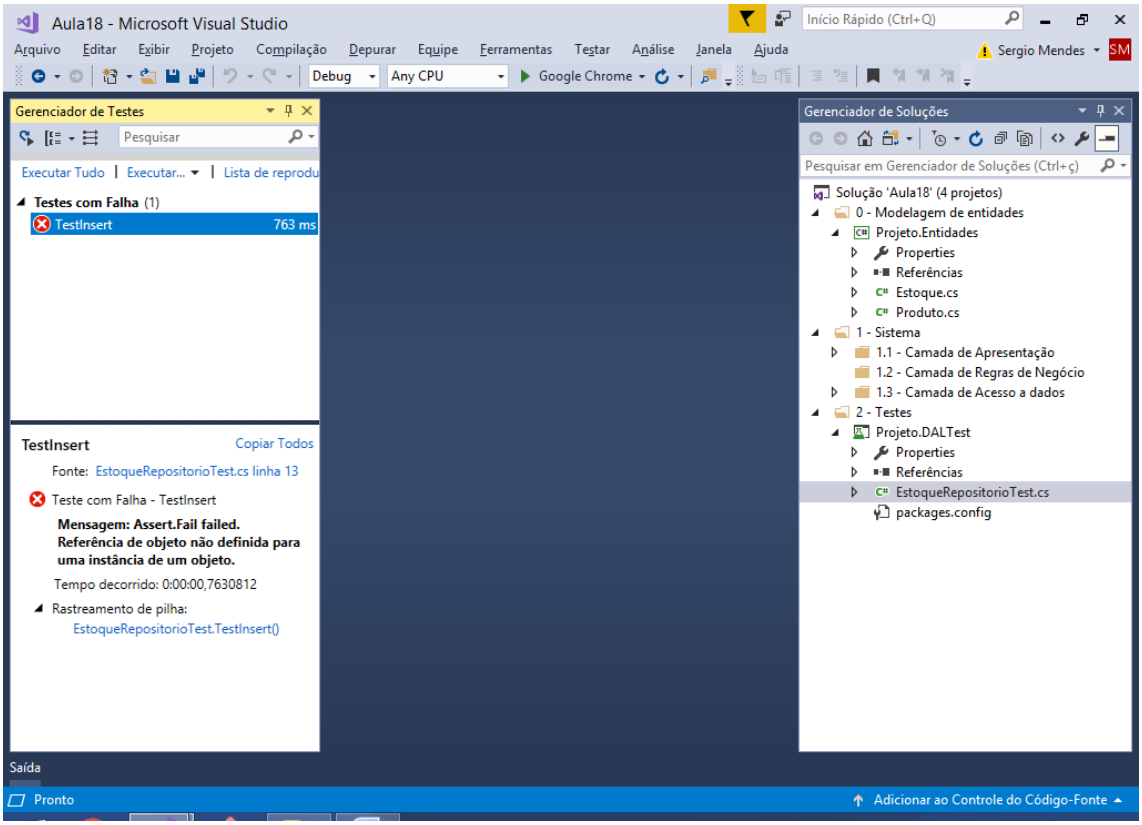


```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Projeto.Entidades;
using Projeto.DAL.Repositories;

namespace Projeto.DALTest
{
    [TestClass]
    public class EstoqueRepositorioTest
    {
        [TestMethod]
        public void TestInsert()
        {
            try
            {
                Estoque e = new Estoque();
                e.Nome = "Estoque Teste";
                e.Descricao = "Registro de Teste";

                EstoqueRepositorio rep = new EstoqueRepositorio();
                rep.Insert(e);
            }
            catch(Exception e)
            {
                //gerar um resultado de falha no teste..
                Assert.Fail(e.Message);
            }
        }
    }
}
```

Executando



Resultado: **Teste falhou**



Continua...