

2 - InfraStructure

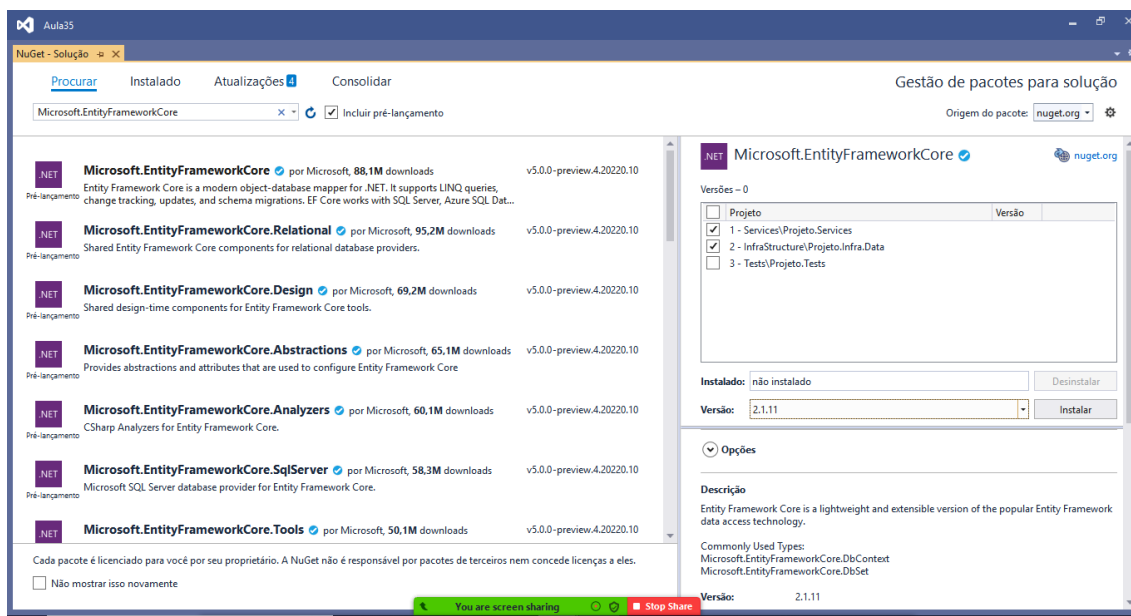
Biblioteca de classes (.NET CORE)



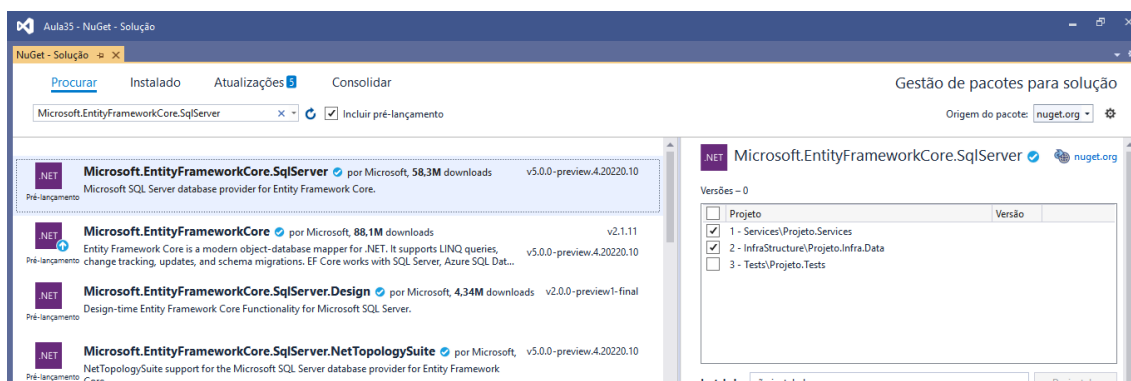
Instalando o EntityFramework

Gerenciador de pacotes do nuget

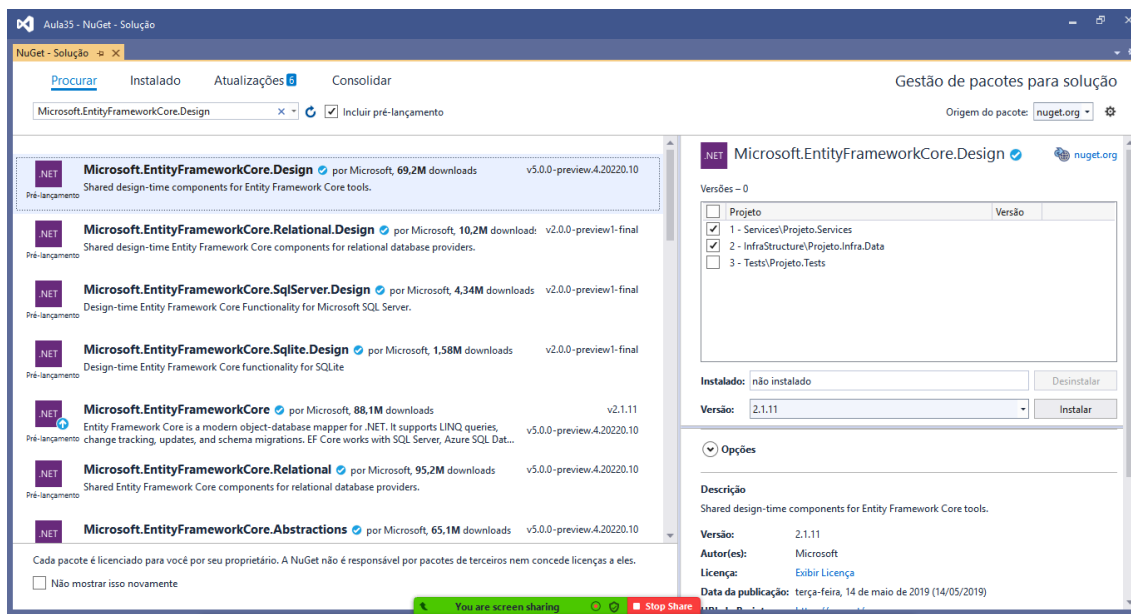
Microsoft.EntityFrameworkCore (v2.1.11)



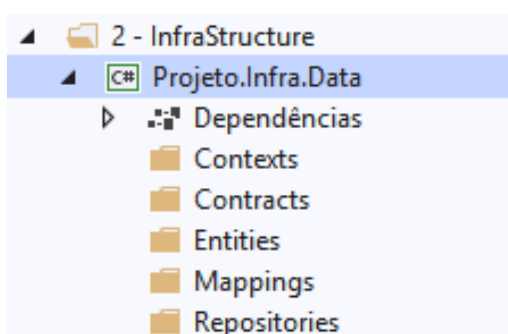
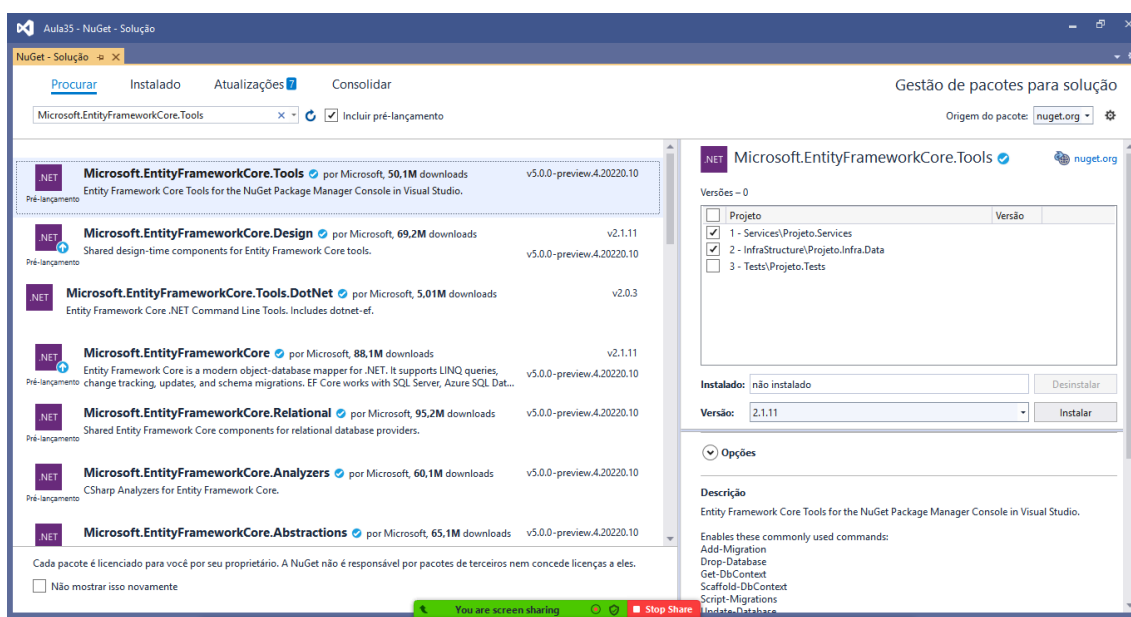
Microsoft.EntityFrameworkCore.SqlServer (v2.1.11)



Microsoft.EntityFrameworkCore.Design (v2.1.11)

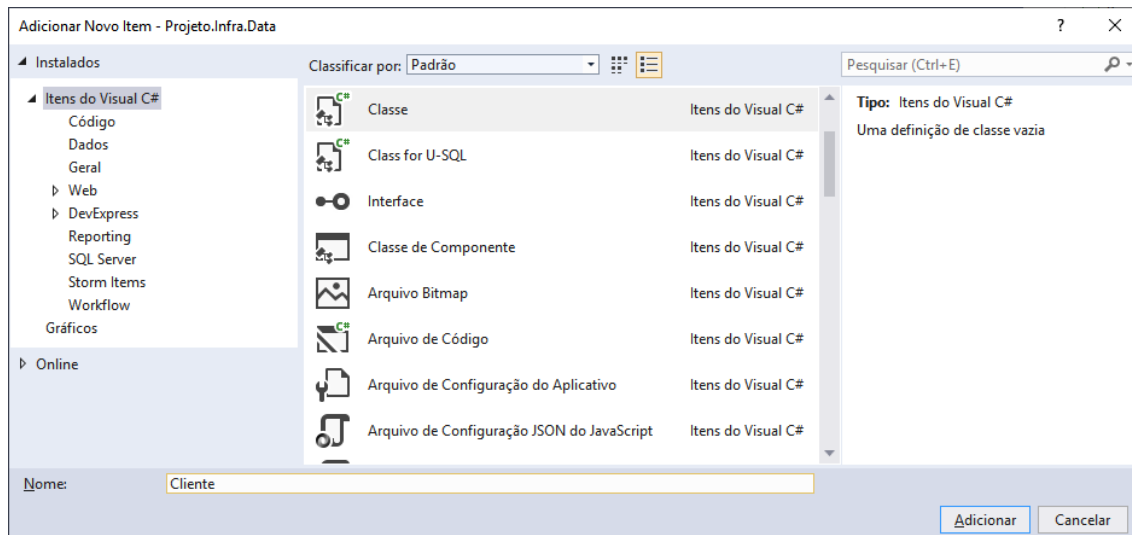


Microsoft.EntityFrameworkCore.Tools (v2.1.11)



/Entities

Modelo de entidades

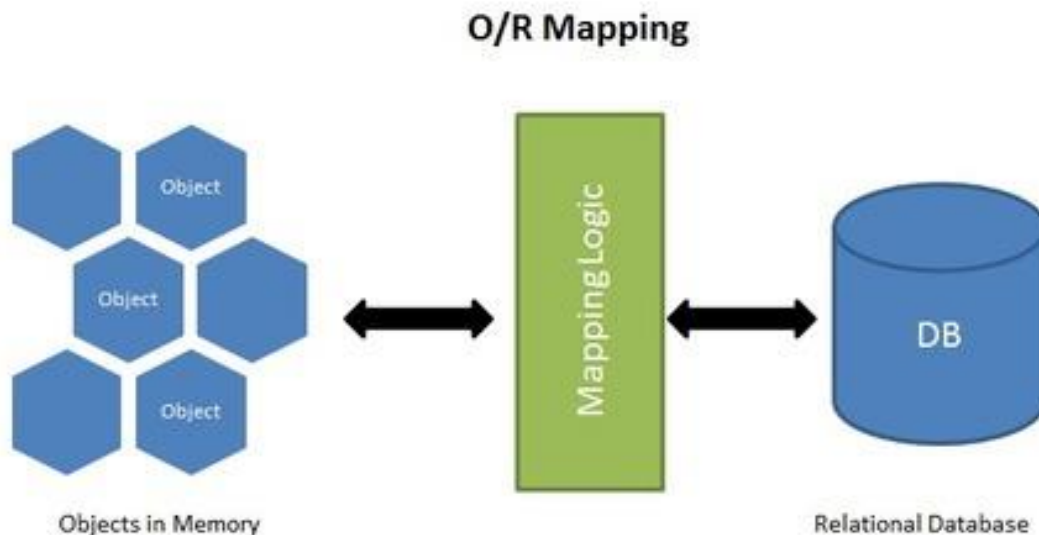


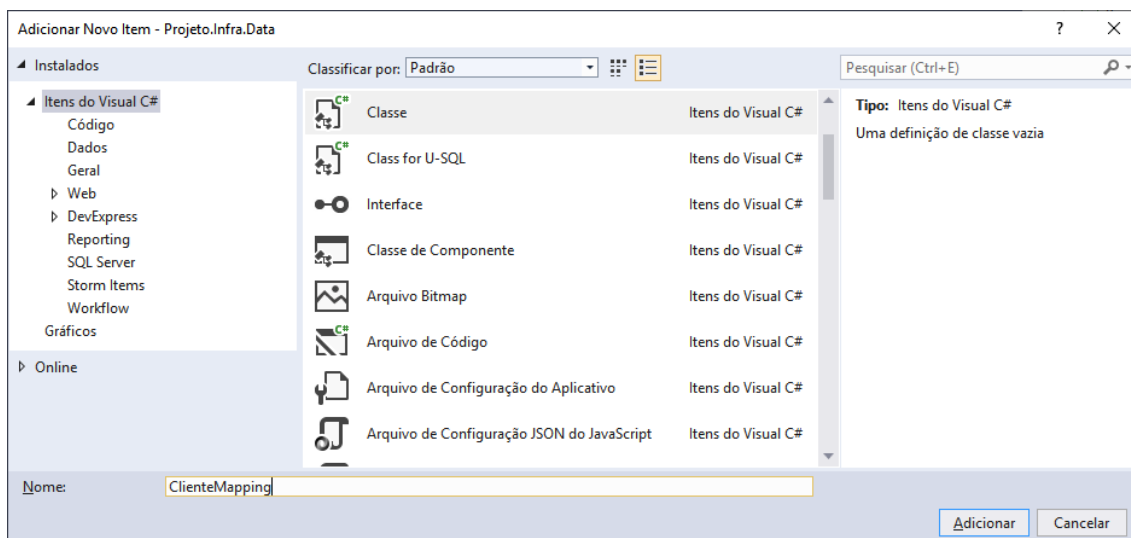
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Infra.Data.Entities
{
    public class Cliente
    {
        public int IdCliente { get; set; }
        public string Nome { get; set; }
        public string Email { get; set; }
        public DateTime DataCriacao { get; set; }
    }
}
```

/Mappings

Maapeamento objeto/relacional das classes de entidade





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

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

            //chave primária
            builder.HasKey(c => c.IdCliente);

            //campos da tabela
            builder.Property(c => c.IdCliente)
                .HasColumnName("IdCliente");

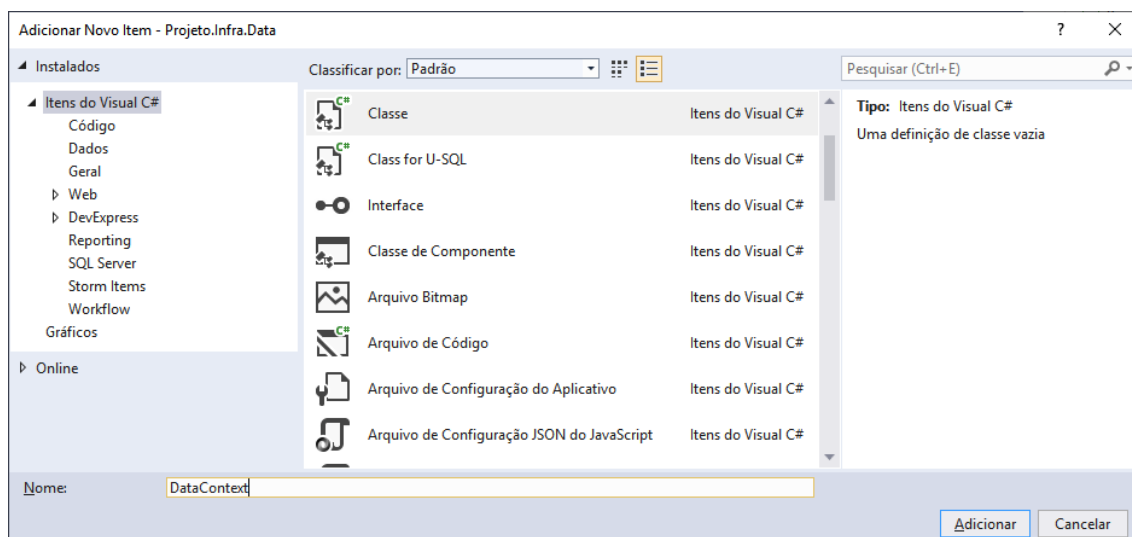
            builder.Property(c => c.Nome)
                .HasColumnName("Nome")
                .HasMaxLength(150)
                .IsRequired();

            builder.Property(c => c.Email)
                .HasColumnName("Email")
                .HasMaxLength(100)
                .IsRequired();

            builder.Property(c => c.DataCriacao)
                .HasColumnName("DataCriacao")
                .IsRequired();
        }
    }
}
```

/Contexts

Criando a classe utilizada para acessar a base de dados por meio do EntityFramework (**DataContext**).



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

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

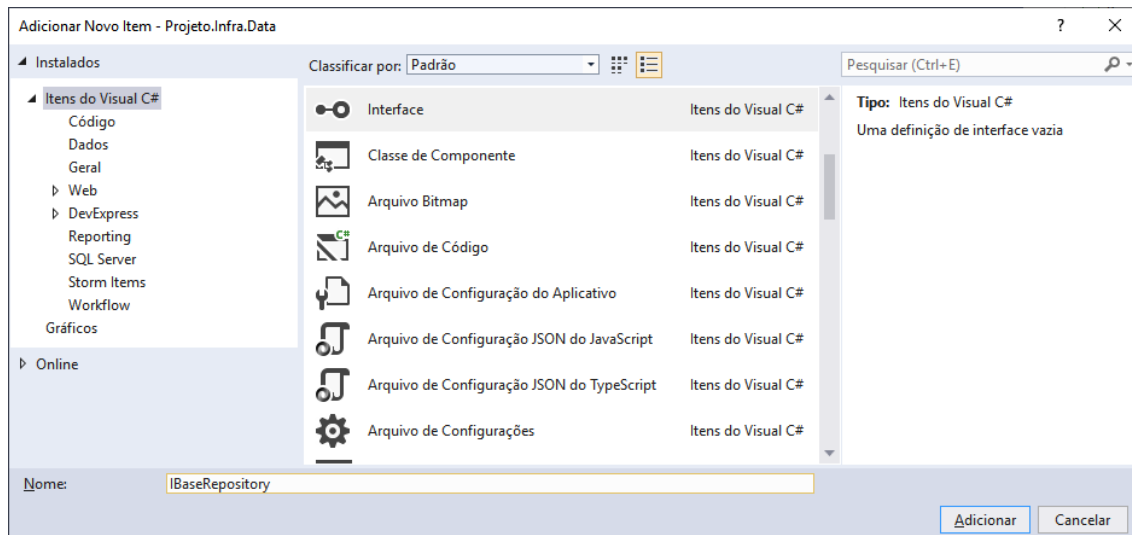
```
{
    //REGRA 1: HERDAR DbContext
    public class DataContext : DbContext
    {
        //REGRA 2: Construtor para receber via injeção de dependência
        //as configurações de acesso ao banco de dados (connectionstring)
        public DataContext(DbContextOptions<DataContext> options)
            : base(options) //construtor da superclasse
        {
        }

        //REGRA 3: Sobrescrita do método OnModelCreating
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            //adicionar cada classe de mapeamento..
            modelBuilder.ApplyConfiguration(new ClienteMapping());
        }

        //REGRA 4: Declarar uma propriedade DbSet para cada entidade
        //DbSet permite o uso do LAMBDA para cada entidade mapeada
        public DbSet<Cliente> Cliente { get; set; }
    }
}
```

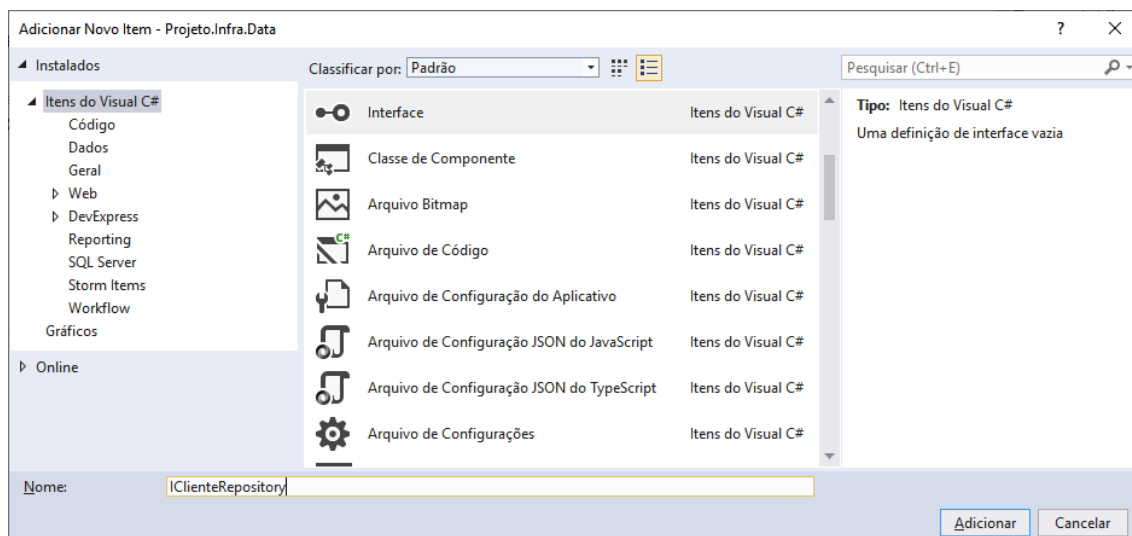
/Contracts

Criando as interfaces para o repositório.



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

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

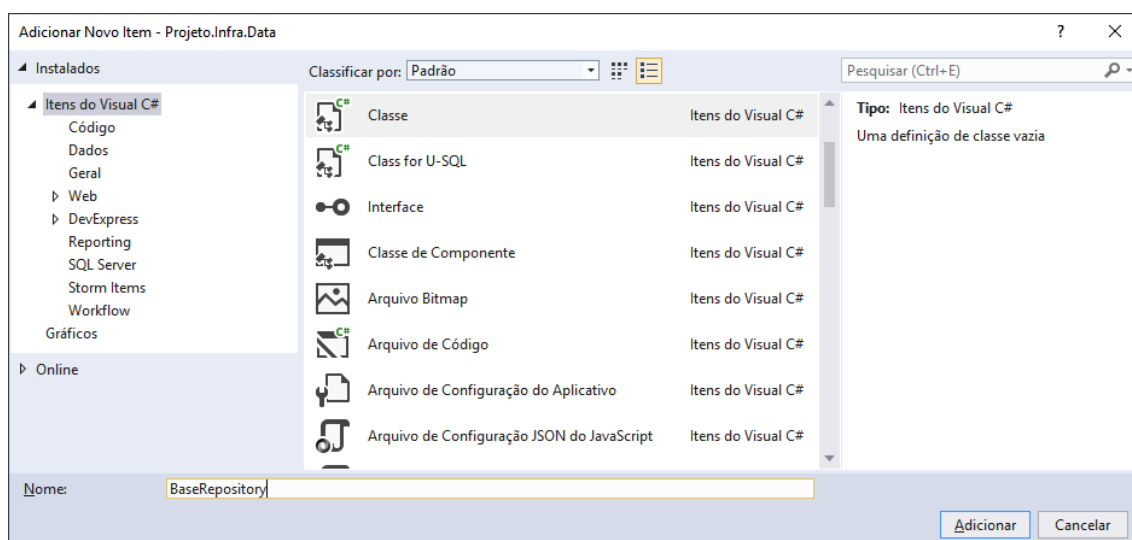


```
using Projeto.Infra.Data.Entities;
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto.Infra.Data.Contracts
{
    public interface IClienteRepository : IBaseRepository<Cliente>
    {
    }
}
```

/Repositories

Implementando as interfaces (contratos) do repositório.



```
using Microsoft.EntityFrameworkCore;
using Projeto.Infra.Data.Contexts;
using Projeto.Infra.Data.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Text;

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

        //construtor para injeção de dependência
        public BaseRepository(DbContext dataContext)
        {
            this.dataContext = dataContext;
        }
    }
}
```

```

public void Inserir(TEntity obj)
{
    dataContext.Entry(obj).State = EntityState.Added;
    dataContext.SaveChanges();
}

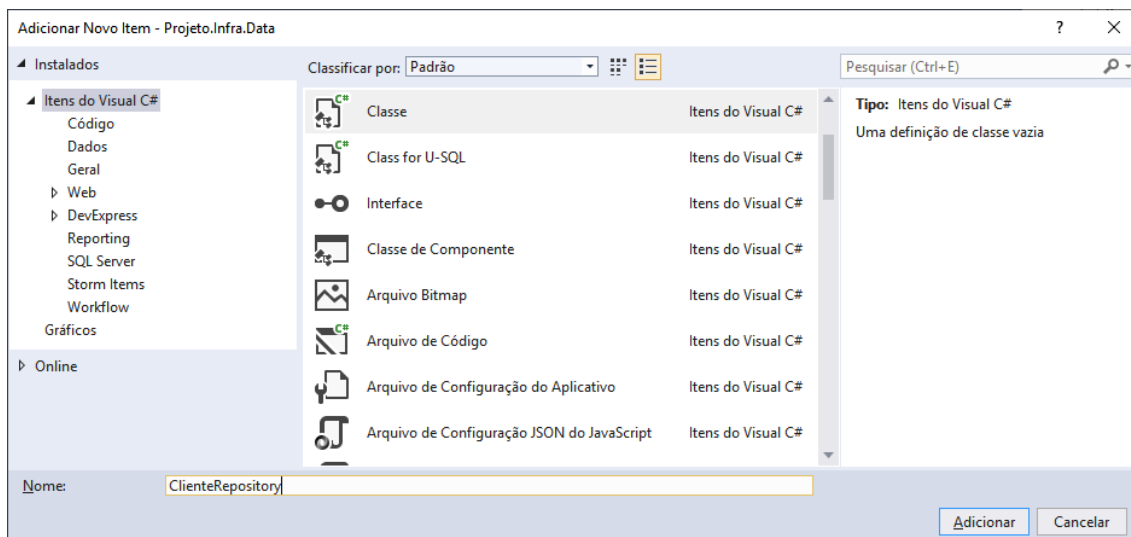
public void Alterar(TEntity obj)
{
    dataContext.Entry(obj).State = EntityState.Modified;
    dataContext.SaveChanges();
}

public void Excluir(TEntity obj)
{
    dataContext.Entry(obj).State = EntityState.Deleted;
    dataContext.SaveChanges();
}

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

public TEntity ObterPorId(int id)
{
    return dataContext.Set<TEntity>().Find(id);
}
}

```



```

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

namespace Projeto.Infra.Data.Repositories
{
    public class ClienteRepository : BaseRepository<Cliente>, IClienteRepository
    {

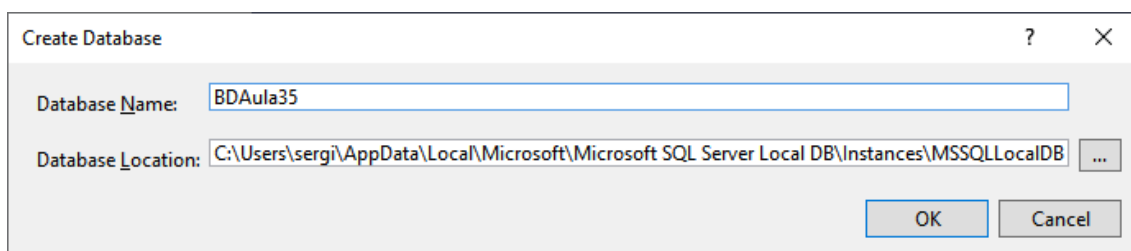
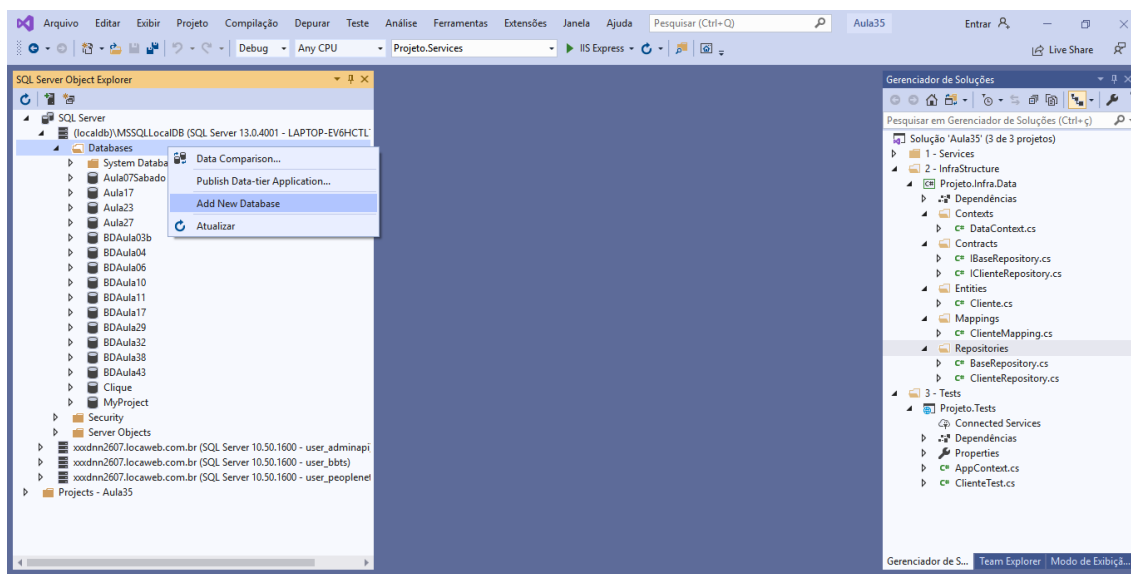
```



```
//atributo
private readonly DataContext dataContext;

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

Criando a base de dados: Pesquisador de objetos do SqlServer

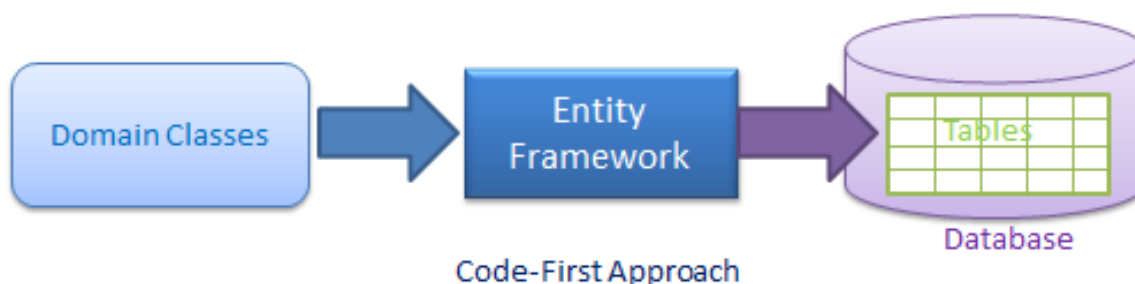


appsettings.json

```
{
  "ConnectionStrings": {
    "Projeto": "Data Source=(localdb)\\MSSQLLocalDB;Initial
    Catalog=BDAula35;Integrated Security=True;Connect
    Timeout=30;Encrypt=False;TrustServerCertificate=False;Applicatio
    nIntent=ReadWrite;MultiSubnetFailover=False"
  }
}
```

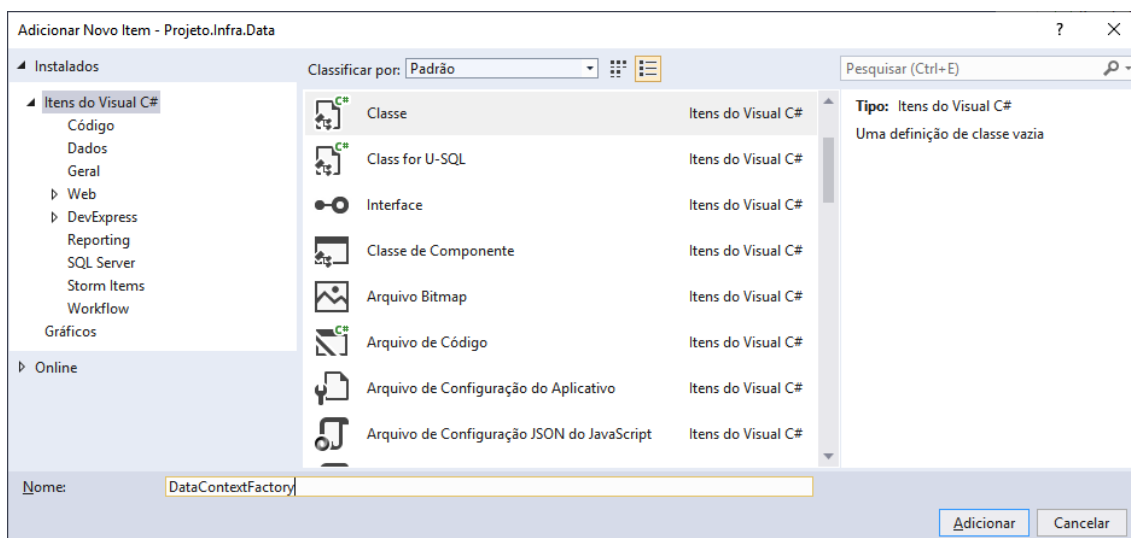
Migrations (CodeFirst)

Recurso do EntityFramework que faz com que o conteúdo do banco de dados (tabelas) sejam atualizadas (criadas ou modificadas) de acordo com o mapeamento das entidades.



Primeiro passo:

Criar uma classe que irá executar o Migrations na base de dados.



```

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=BD Aula35;Integrated Security=True;Connect
                                Timeout=30;Encrypt=False;TrustServerCertificate=False;Applica
                                tionIntent=ReadWrite;MultiSubnetFailover=False");
        }
    }
}

```

```

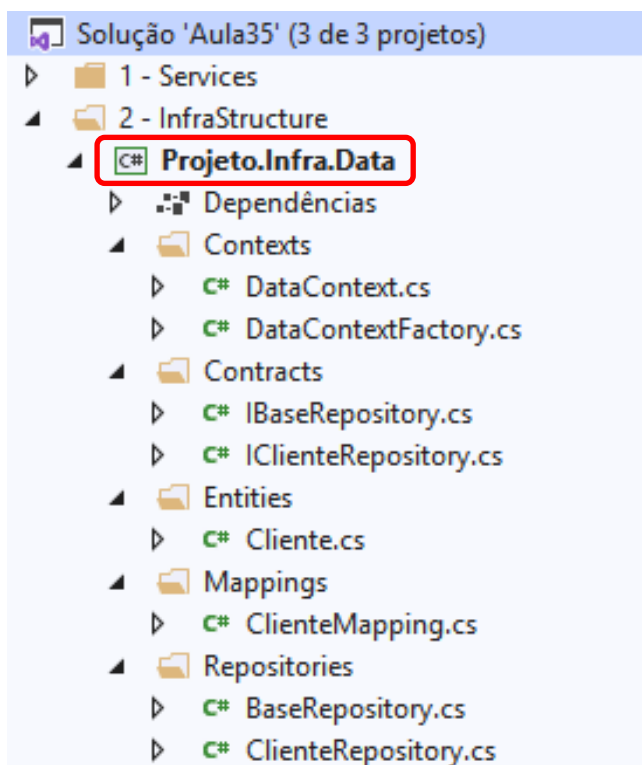
        return new DataContext(builder.Options);
    }
}
}

```

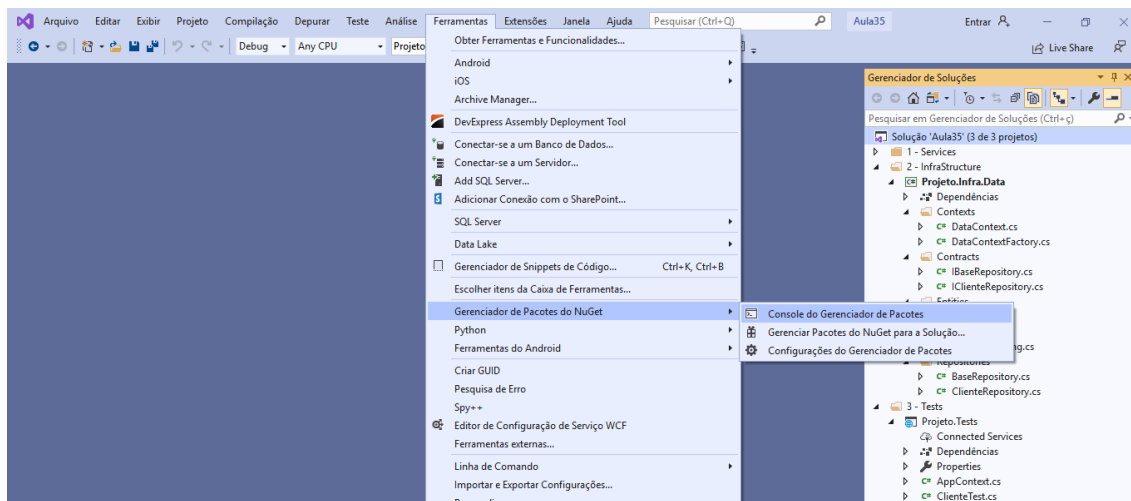
Primeiro passo:

Executando o Migrations

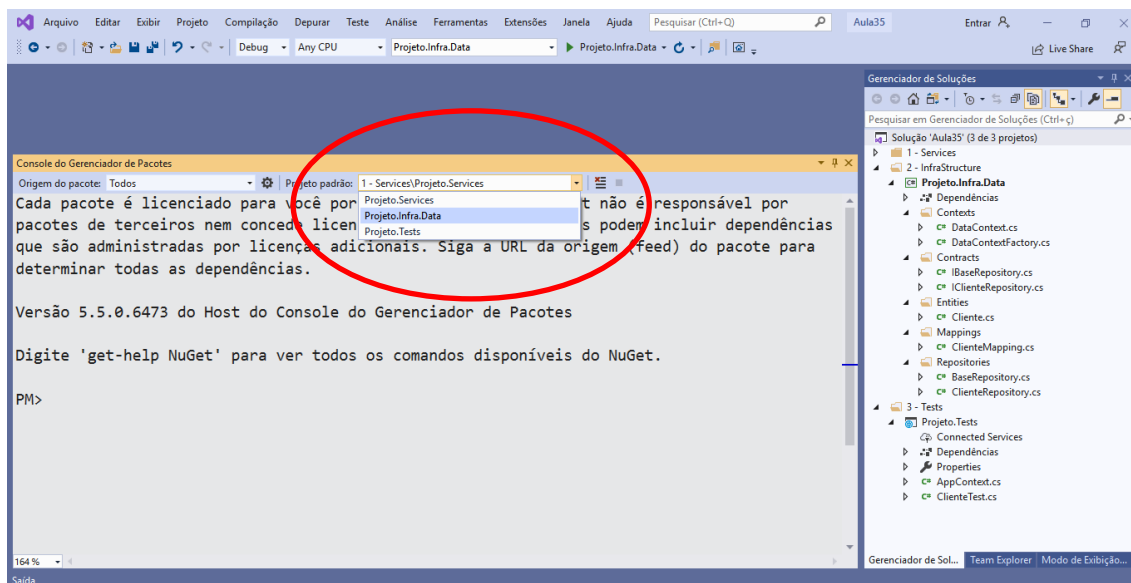
Definir o projeto **Infra.Data** como projeto de inicialização da solution



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

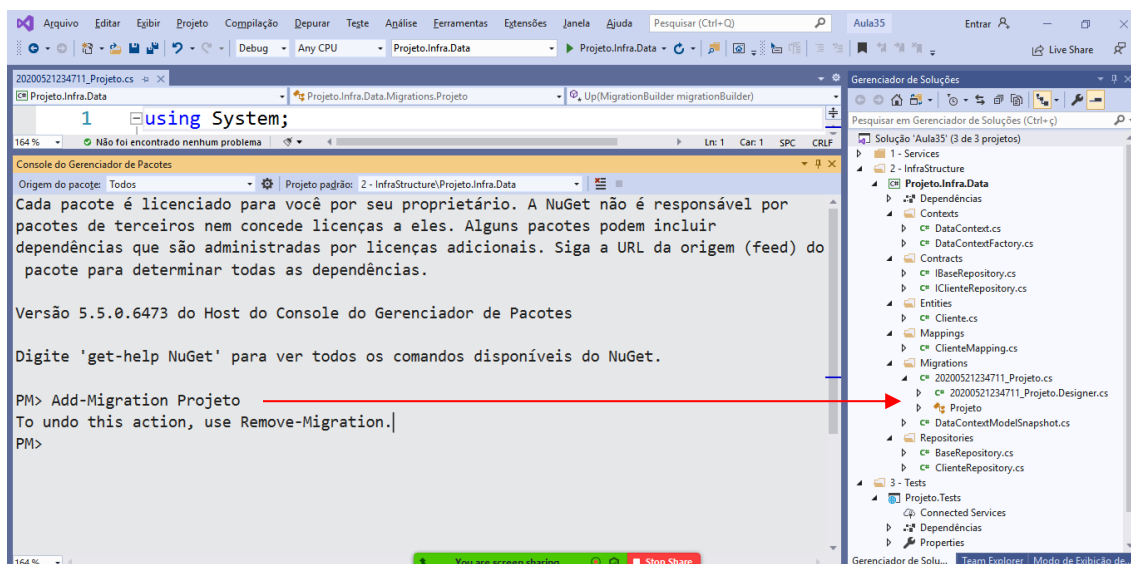


Selecione o **Projeto.Infra.Data** como projeto padrão:



PM> Add-Migration Projeto

To undo this action, use Remove-Migration.

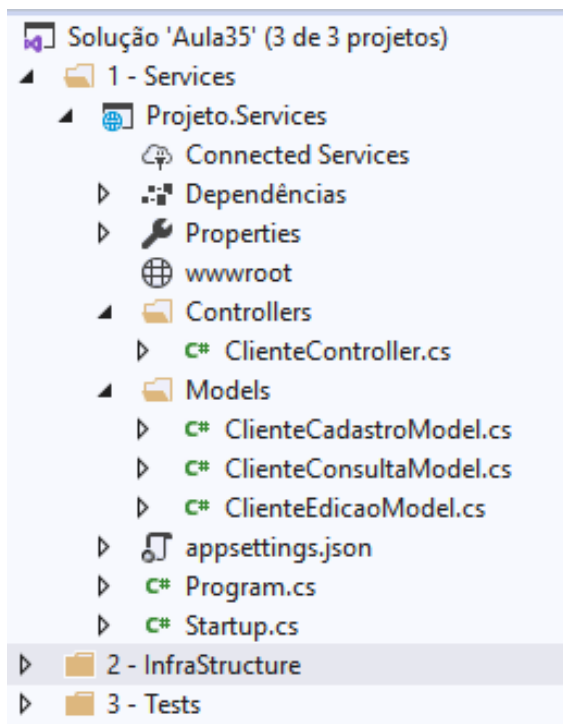


PM> Update-Database

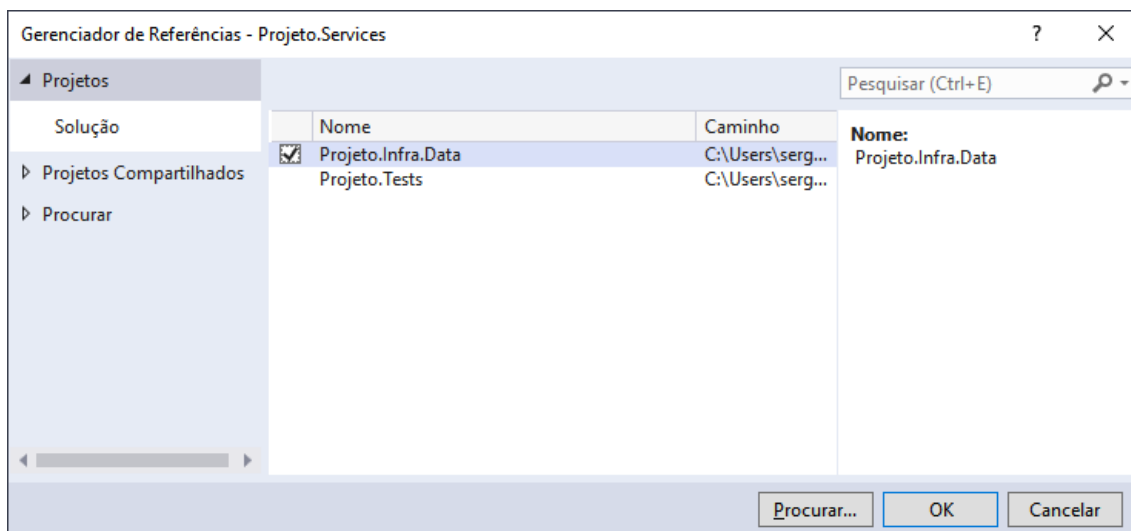
Applying migration '20200521234711_Projeto'.

Done.

Tabelas criadas:



Adicionando referência no projeto Services para Infra.Data



Startup.cs

Configurando o EntityFramework e a injeção de dependência das classes.

```
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 Microsoft.OpenApi.Models;
using Projeto.Infra.Data.Contexts;
using Projeto.Infra.Data.Contracts;
using Projeto.Infra.Data.Repositories;

namespace Projeto.Services
{
    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 Swagger

            services.AddSwaggerGen(
                c =>
                {
                    c.SwaggerDoc("v1", new OpenApiInfo
                    {
                        Title = "Sistema de Controle de Clientes",
                        Description = "API REST para integração  
com serviços de cliente",
                        Version = "v1",
                        Contact = new OpenApiContact
                        {
                            Name = "COTI Informática",
                            Url = new Uri("http://www.cotiinformatica.com.br/"),
                            Email = "contato@cotiinformatica.com.br"
                        }
                    });
                });

            #endregion

            #region EntityFramework

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

            #endregion
        }
    }
}
```

```
#region Injeção de Dependência

services.AddTransient<IClienteRepository, ClienteRepository>();

#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();
    }

    #region Swagger

    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Projeto API");
    });

    #endregion

    app.UseMvc();
}
}
```

/Controllers/ClienteController.cs

Desenvolvendo os métodos da API

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Projeto.Infra.Data.Contracts;
using Projeto.Infra.Data.Entities;
using Projeto.Services.Models;

namespace Projeto.Services.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ClienteController : ControllerBase
    {
        //atributo
        private readonly IClienteRepository clienteRepository;

        //construtor para injeção de dependência
        public ClienteController(IClienteRepository clienteRepository)
        {
            this.clienteRepository = clienteRepository;
        }
    }
}
```

```
[HttpPost]
public IActionResult Post(ClienteCadastroModel model)
{
    //verificar se algum campo da model está com erro!
    if (!ModelState.IsValid)
        return BadRequest(); //HTTP 400 (BadRequest)

    try
    {
        var cliente = new Cliente();

        cliente.Nome = model.Nome;
        cliente.Email = model.Email;
        cliente.DataCriacao = DateTime.Now;

        clienteRepository.Inserir(cliente);

        return Ok("Cliente cadastrado com sucesso.");
    }
    catch(Exception e)
    {
        return StatusCode(500, "Erro: " + e.Message);
    }
}

[HttpPut]
public IActionResult Put(ClienteEdicaoModel model)
{
    //verificar se algum campo da model está com erro!
    if (!ModelState.IsValid)
        return BadRequest(); //HTTP 400 (BadRequest)

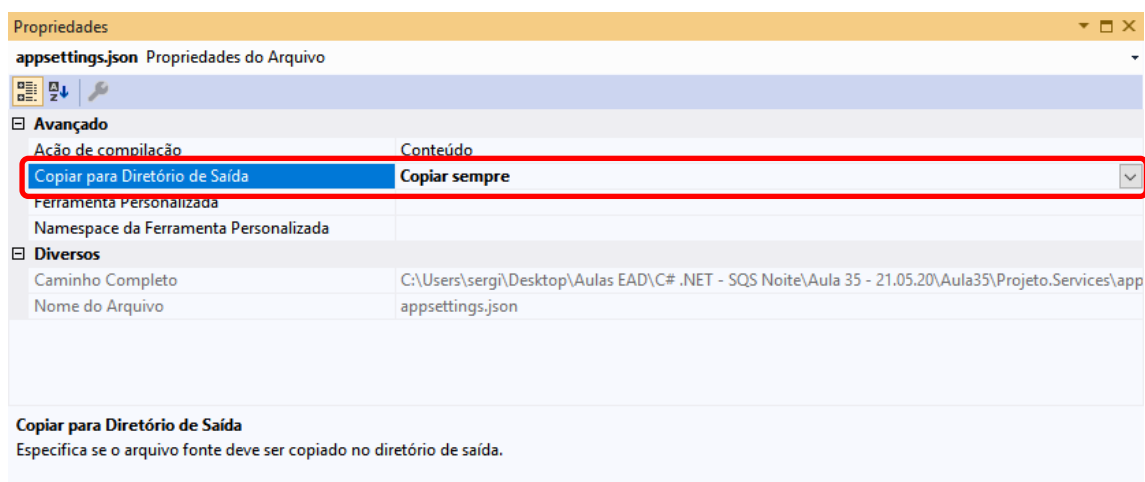
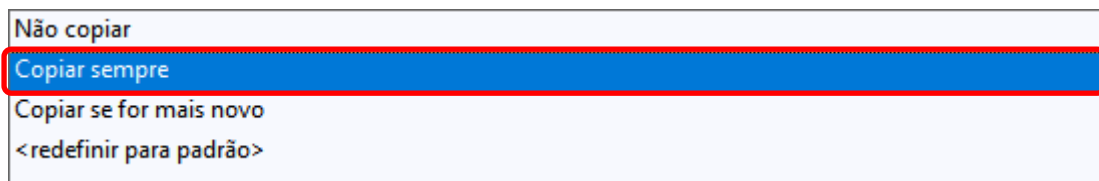
    return StatusCode(500, new { message = "Não implementado." });
}

[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    return StatusCode(500, new { message = "Não implementado." });
}

[HttpGet]
[ProducesResponseType(200, Type = typeof(List<ClienteConsultaModel>))]
public IActionResult GetAll()
{
    return StatusCode(500, new { message = "Não implementado." });
}

[HttpGet("{id}")]
[ProducesResponseType(200, Type = typeof(ClienteConsultaModel))]
public IActionResult GetById(int id)
{
    return StatusCode(500, new { message = "Não implementado." });
}
}
}
```


Para executar os testes, é necessário modificar as propriedades do arquivo **appsettings.json** para o seguinte valor:



Em seguida, precisamos modificar a classe **AppContext** no projeto de testes para ler o arquivo **appsettings.json**

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.TestHost;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

namespace Projeto.Tests
{
    public class AppContext
    {
        //classe para executar chamadas HTTP na API..
        //prop + 2x[tab]
        public HttpClient Client { get; set; }

        //servidor de testes
        private readonly TestServer testServer;

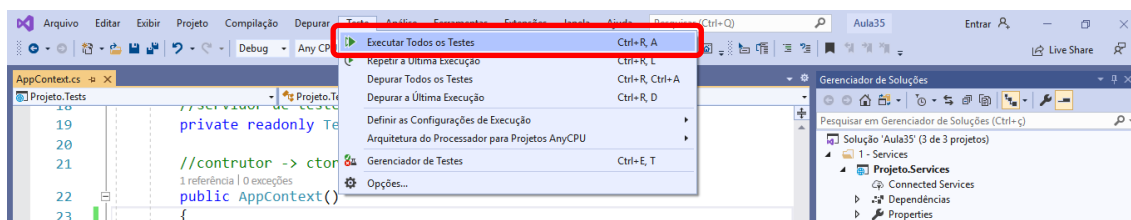
        //contrutor -> ctor + 2x[tab]
    }
}
```

```
public AppContext()
{
    //lendo o arquivo appsettings.json
    var configuration = new ConfigurationBuilder()
        .AddJsonFile("appsettings.json")
        .Build();

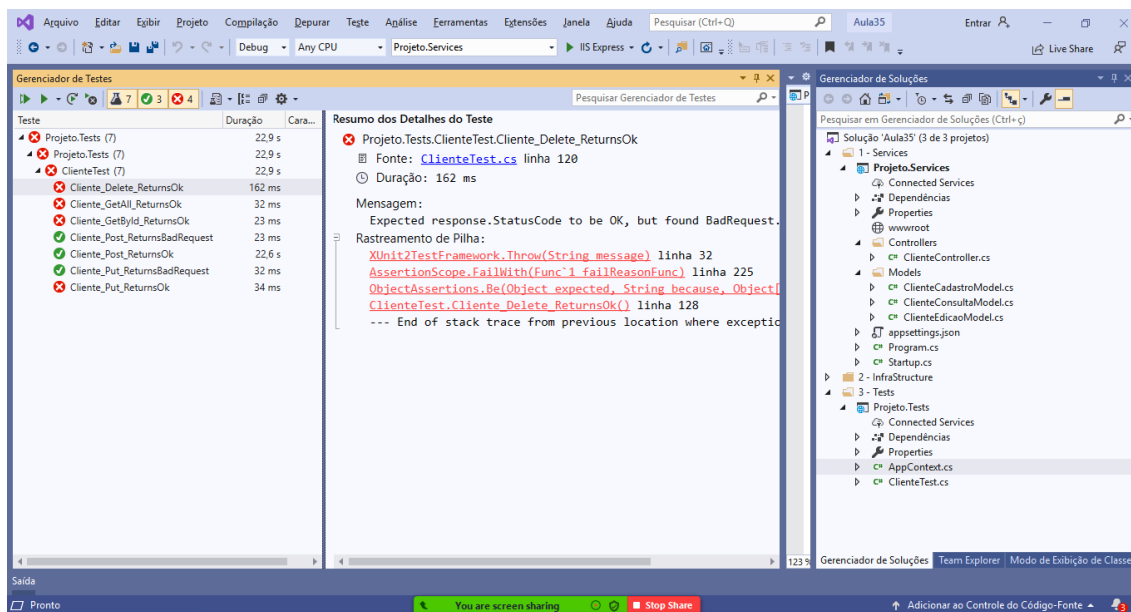
    //inicializar o servidor de testes do projeto (TestServer)
    //este projeto de testes irá executar
    //a API por meio da classe 'Startup'
    testServer = new TestServer(new WebHostBuilder()
        .UseConfiguration(configuration)
        .UseStartup<Services.Startup>());

    //instanciando a classe utilizada para executar as chamadas na API
    Client = testServer.CreateClient();
}
}
```

Executando os testes:



Resultado:



```
using FluentAssertions;
using Newtonsoft.Json;
using Projeto.Services.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Xunit;

namespace Projeto.Tests
{
    public class ClienteTest
    {
        //atributos..
        private readonly AppContext appContext;
        private readonly string endpoint;

        //construtor -> ctor + 2x[tab]
        public ClienteTest()
        {
            appContext = new AppContext();
            endpoint = "/api/Cliente";
        }

        [Fact] //método para execução de teste do XUnit
        //async -> método executado como uma Thread (assíncrono)
        public async Task Cliente_Post_ReturnsOk()
        {
            //preencher os campos da model
            var model = new ClienteCadastroModel()
            {
                Nome = "Sergio Mendes",
                Email = "sergio.coti@gmail.com"
            };

            //montando os dados em JSON que serão enviados para a API
            var request = new StringContent(JsonConvert.SerializeObject(model),
                Encoding.UTF8, "application/json");

            //executando o serviço da API..
            var response = await appContext.Client.PostAsync(endpoint, request);

            //critério de teste (Serviço da API retornar HTTP OK (200))
            response.StatusCode.Should().Be(HttpStatusCode.OK);
        }

        [Fact] //método para execução de teste do XUnit
        //async -> método executado como uma Thread (assíncrono)
        public async Task Cliente_Post_ReturnsBadRequest()
        {
            //preencher os campos da model
            var model = new ClienteCadastroModel()
            {
                Nome = string.Empty, //vazio
                Email = string.Empty //vazio
            };
        }
    }
}
```

```
//montando os dados em JSON que serão enviados para a API
var request = new StringContent(JsonConvert.SerializeObject(model),
    Encoding.UTF8, "application/json");

//executando o serviço da API..
var response = await appContext.Client.PostAsync(endpoint, request);

//critério de teste (Serviço da API retornar HTTP BADREQUEST (400))
response.StatusCode.Should().Be(HttpStatusCode.BadRequest);
}

[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_Put_ReturnsOk()
{
    //preencher os campos da model
    var model = new ClienteEdicaoModel()
    {
        IdCliente = Guid.NewGuid(),
        Nome = "Sergio Mendes",
        Email = "sergio.coti@gmail.com"
    };

    //montando os dados em JSON que serão enviados para a API
    var request = new StringContent(JsonConvert.SerializeObject(model),
        Encoding.UTF8, "application/json");

    //executando o serviço da API..
    var response = await appContext.Client.PutAsync(endpoint, request);

    //critério de teste (Serviço da API retornar HTTP OK (200))
    response.StatusCode.Should().Be(HttpStatusCode.OK);
}

[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_Put_ReturnsBadRequest()
{
    //preencher os campos da model
    var model = new ClienteEdicaoModel()
    {
        IdCliente = Guid.NewGuid(),
        Nome = string.Empty, //vazio
        Email = string.Empty //vazio
    };

    //montando os dados em JSON que serão enviados para a API
    var request = new StringContent(JsonConvert.SerializeObject(model),
        Encoding.UTF8, "application/json");

    //executando o serviço da API..
    var response = await appContext.Client.PutAsync(endpoint, request);

    //critério de teste (Serviço da API retornar HTTP BADREQUEST (400))
    response.StatusCode.Should().Be(HttpStatusCode.BadRequest);
}
```

```
[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_Delete_ReturnsOk()
{
    var id = Guid.NewGuid().ToString();

    //executando o serviço da API..
    var response = await appContext.Client.DeleteAsync
        (endpoint + "/" + id);

    //critério de teste (Serviço da API retornar HTTP OK (200))
    response.StatusCode.Should().Be(HttpStatusCode.OK);
}

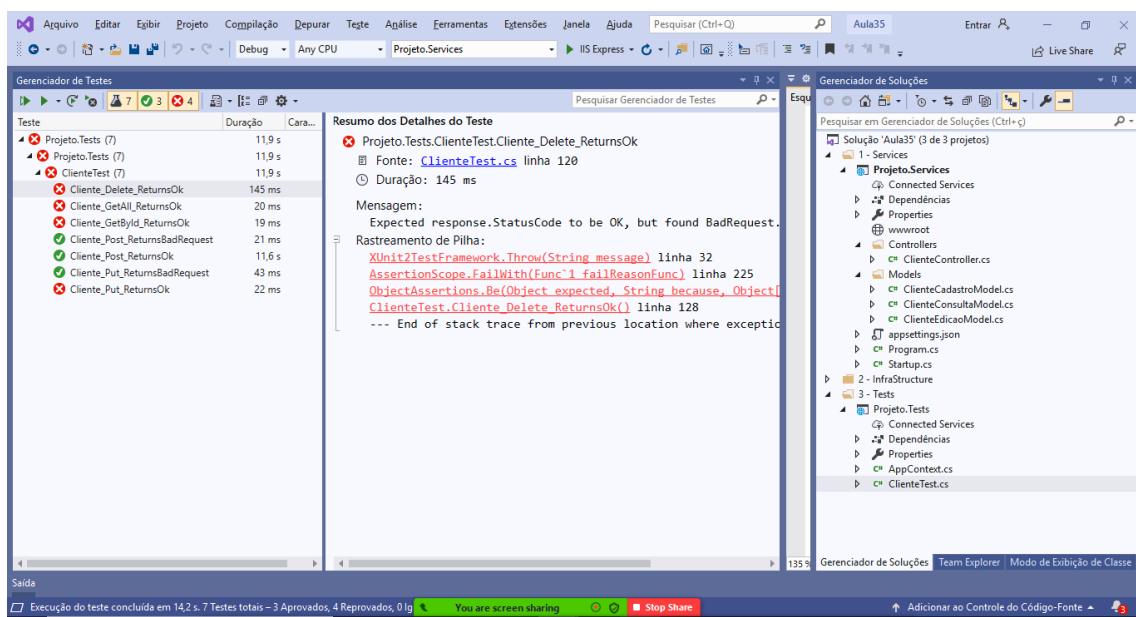
[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_GetAll_ReturnsOk()
{
    //executando o serviço da API..
    var response = await appContext.Client.GetAsync(endpoint);

    //critério de teste (Serviço da API retornar HTTP OK (200))
    response.StatusCode.Should().Be(HttpStatusCode.OK);
}

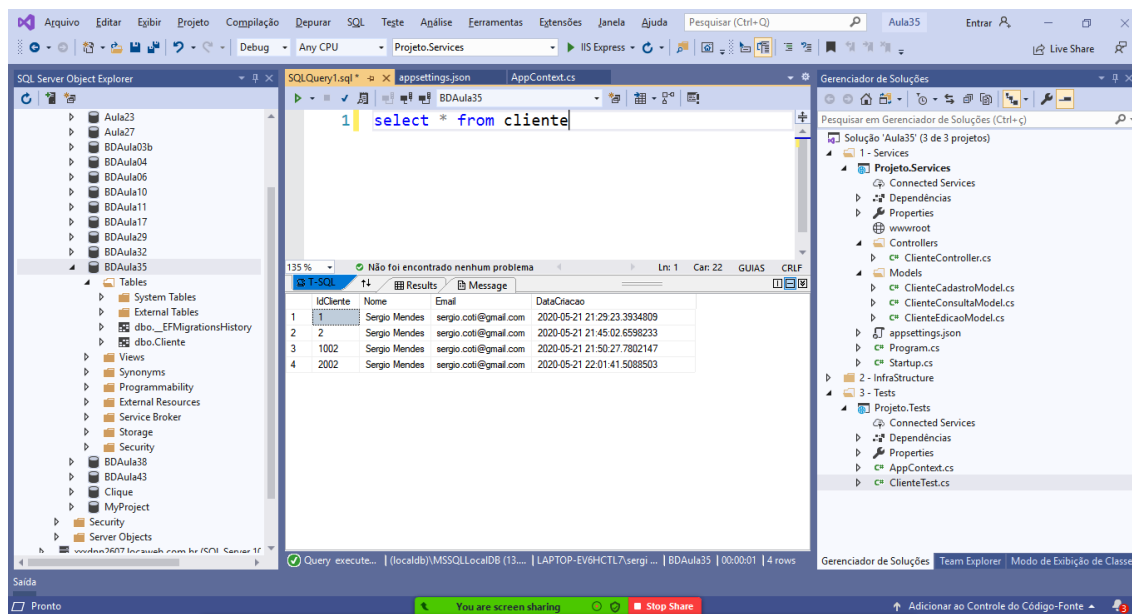
[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_GetById_ReturnsOk()
{
    var id = Guid.NewGuid().ToString();

    //executando o serviço da API..
    var response = await appContext.Client.GetAsync(endpoint + "/" + id);

    //critério de teste (Serviço da API retornar HTTP OK (200))
    response.StatusCode.Should().Be(HttpStatusCode.OK);
}
}
```



No banco de dados:



The screenshot shows the Visual Studio IDE with the SQL Server Enterprise Edition interface. The SQL query editor displays the query: `select * from cliente`. The results are shown in a table with 4 rows and 5 columns: IdCliente, Nome, Email, DataCriacao, and Message.

| IdCliente | Nome | Email | DataCriacao | Message |
|-----------|---------------|-----------------------|-----------------------------|---------|
| 1 | Sergio Mendes | sergio.coti@gmail.com | 2020-05-21 21:29:23.3934809 | |
| 2 | Sergio Mendes | sergio.coti@gmail.com | 2020-05-21 21:45:02.6598233 | |
| 1002 | Sergio Mendes | sergio.coti@gmail.com | 2020-05-21 21:50:27.7802147 | |
| 2002 | Sergio Mendes | sergio.coti@gmail.com | 2020-05-21 22:01:41.5088503 | |