

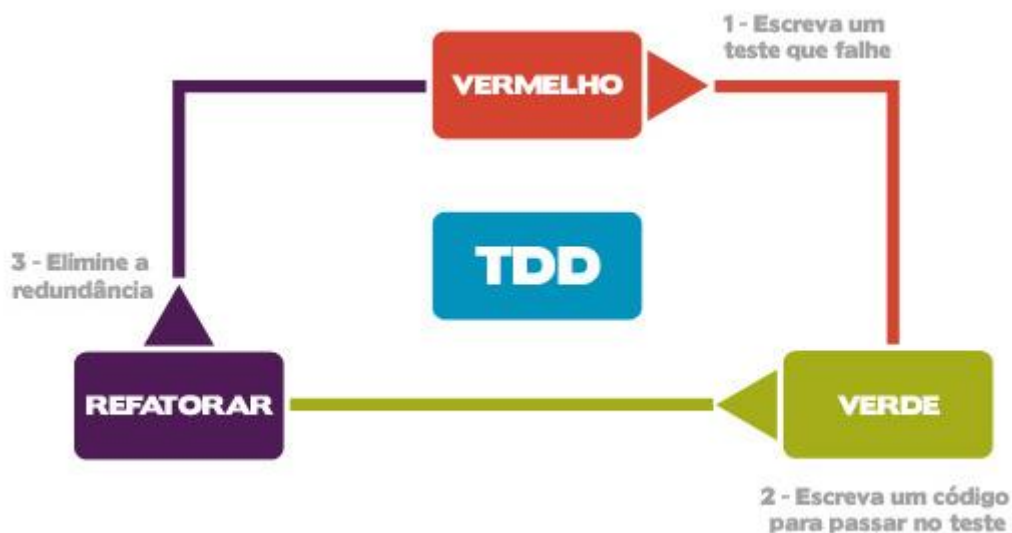
TDD – Test Driven Development

Desenvolvimento orientado a testes

Consiste em uma metodologia de desenvolvimento de software onde priorizamos a execução dos testes para validar se a aplicação está em conformidade com os requisitos.

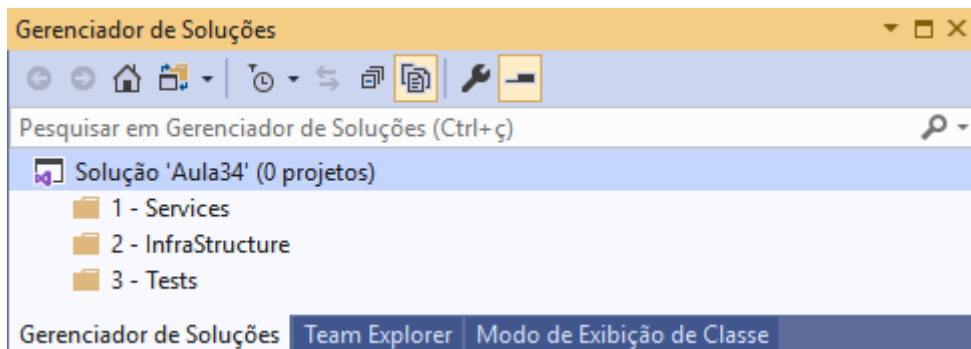
Ciclo TDD:

1. Criando um Teste
2. Desenvolver um programa para passar no teste
3. Refatorar (propor melhorias, otimizações etc.)



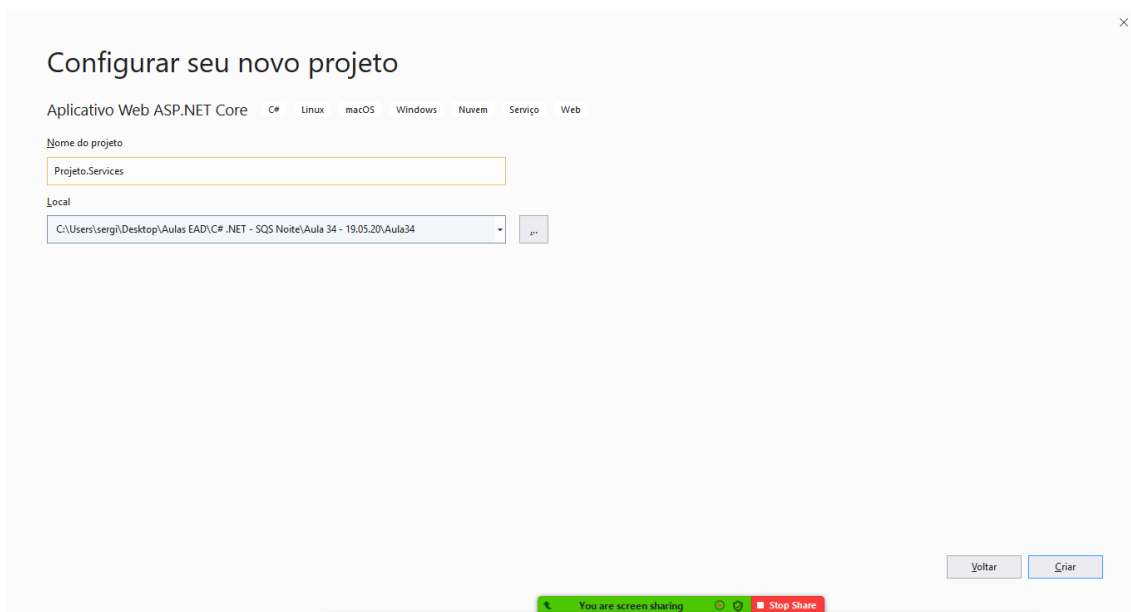
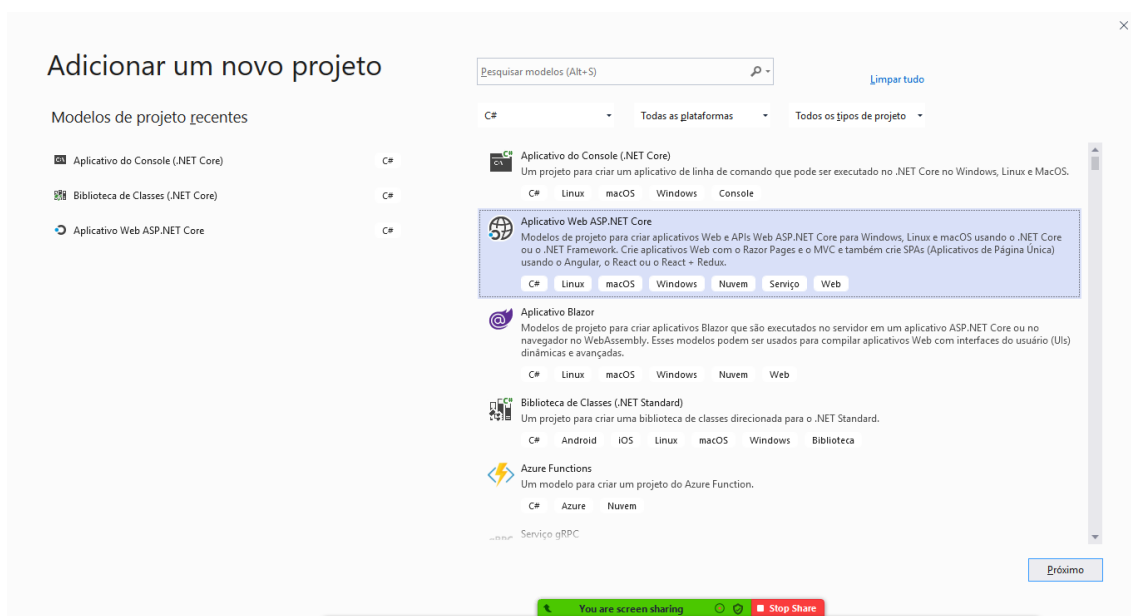
Criando uma solution em branco:

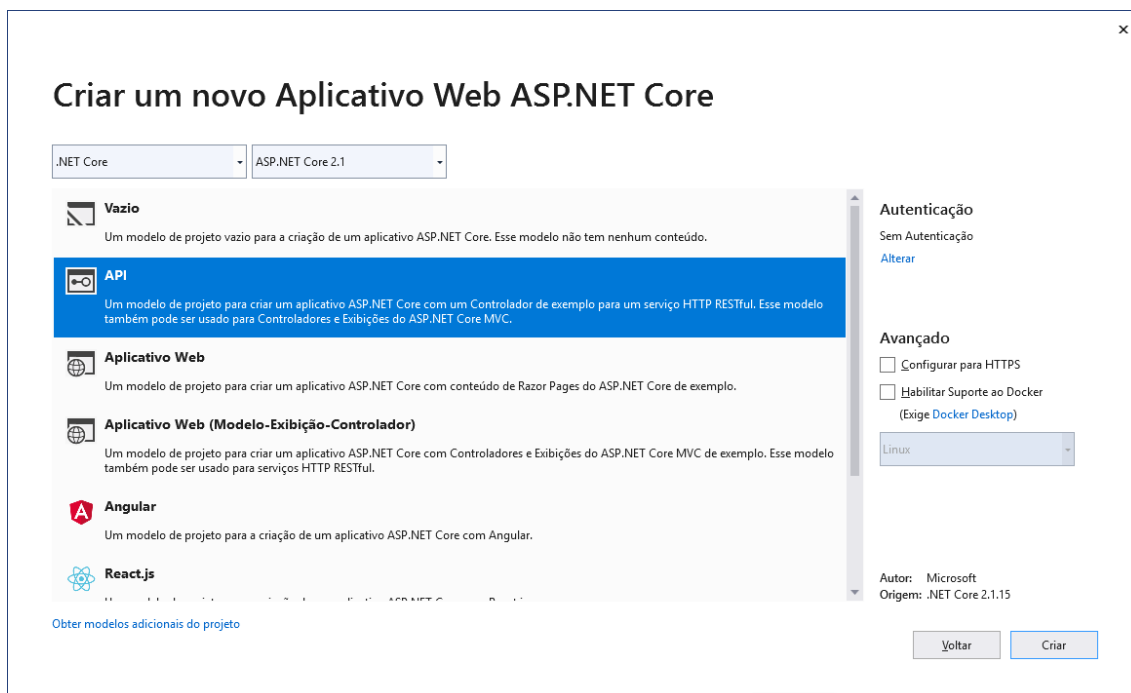




1 – Services

Camada de serviços baseado em NET CORE API





Criar um novo Aplicativo Web ASP.NET Core

.NET Core: .NET Core 2.1

Vazio
Um modelo de projeto vazio para a criação de um aplicativo ASP.NET Core. Esse modelo não tem nenhum conteúdo.

API
Um modelo de projeto para criar um aplicativo ASP.NET Core com um Controlador de exemplo para um serviço HTTP RESTful. Esse modelo também pode ser usado para Controladores e Exibições do ASP.NET Core MVC.

Aplicativo Web
Um modelo de projeto para criar um aplicativo ASP.NET Core com conteúdo de Razor Pages do ASP.NET Core de exemplo.

Aplicativo Web (Modelo-Exibição-Controlador)
Um modelo de projeto para criar um aplicativo ASP.NET Core com Controladores e Exibições do ASP.NET Core MVC de exemplo. Esse modelo também pode ser usado para serviços HTTP RESTful.

Angular
Um modelo de projeto para a criação de um aplicativo ASP.NET Core com Angular.

React.js

[Obter modelos adicionais do projeto](#)

Autenticação
Sem Autenticação
[Alterar](#)

Avançado
☐ Configurar para HTTPS
☐ Habilitar Suporte ao Docker (Exige [Docker Desktop](#))
Linux

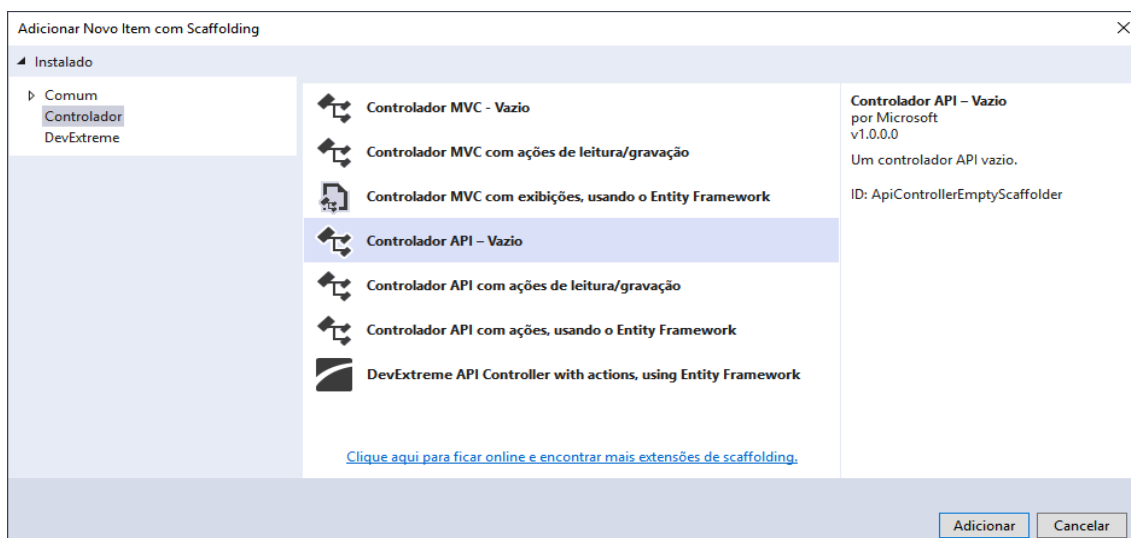
Autor: Microsoft
Origem: .NET Core 2.1.15

[Voltar](#) [Criar](#)

Tarefa: **Prototipar uma API para Clientes**

ENDPOINT: /api/Cliente

Methods: POST
PUT
DELETE
GET



Adicionar Novo Item com Scaffolding

Instalado

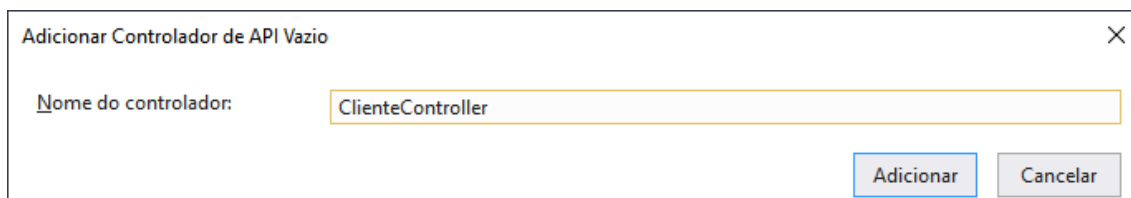
Comum
Controlador
DevExtreme

Controlador MVC - Vazio
Controlador MVC com ações de leitura/gravação
Controlador MVC com exibições, usando o Entity Framework
Controlador API - Vazio
Controlador API com ações de leitura/gravação
Controlador API com ações, usando o Entity Framework
DevExtreme API Controller with actions, using Entity Framework

[Clique aqui para ficar online e encontrar mais extensões de scaffolding.](#)

Controlador API - Vazio
por Microsoft
v1.0.0.0
Um controlador API vazio.
ID: ApiControllerEmptyScaffolder

[Adicionar](#) [Cancelar](#)



Adicionar Controlador de API Vazio

Nome do controlador:

[Adicionar](#) [Cancelar](#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace Projeto.Services.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ClienteController : ControllerBase
    {
        [HttpPost]
        public IActionResult Post()
        {
            return StatusCode(500, new { message = "Não implementado." });
        }

        [HttpPut]
        public IActionResult Put()
        {
            return StatusCode(500, new { message = "Não implementado." });
        }

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

        [HttpGet]
        public IActionResult GetAll()
        {
            return StatusCode(500, new { message = "Não implementado." });
        }

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

/Models

Classes de modelo de dados (entrada / saída) na API:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations; //validações

namespace Projeto.Services.Models
{
    public class ClienteCadastroModel
    {
    }
```

```
[Required(ErrorMessage = "Informe o nome do cliente.")]
public string Nome { get; set; }

[EmailAddress(ErrorMessage = "Informe um email válido.")]
[Required(ErrorMessage = "Informe o email do cliente.")]
public string Email { get; set; }
}
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations; //validações

namespace Projeto.Services.Models
{
    public class ClienteEdicaoModel
    {
        [Required(ErrorMessage = "Informe o id do cliente.")]
        public Guid IdCliente { get; set; }

        [Required(ErrorMessage = "Informe o nome do cliente.")]
        public string Nome { get; set; }

        [EmailAddress(ErrorMessage = "Informe um email válido.")]
        [Required(ErrorMessage = "Informe o email do cliente.")]
        public string Email { get; set; }
    }
}

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

namespace Projeto.Services.Models
{
    public class ClienteConsultaModel
    {
        public Guid IdCliente { get; set; }
        public string Nome { get; set; }
        public string Email { get; set; }
        public DateTime DataCriacao { get; set; }
    }
}
```

Voltando na classe **ClienteController.cs**

Utilizando as models para criação dos serviços

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

```
namespace Projeto.Services.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ClienteController : ControllerBase
    {
        [HttpPost]
        public IActionResult Post(ClienteCadastroModel 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." });
        }

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

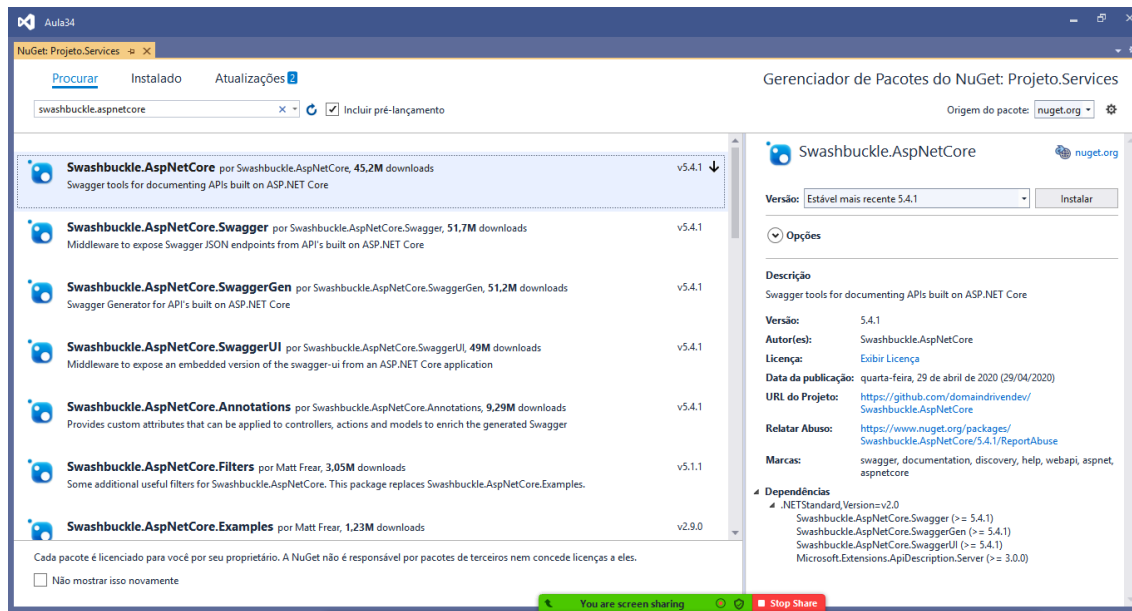
            return StatusCode(500, new { message = "Não implementado." });
        }

        [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." });
        }
    }
}
```

Instalando o Swagger (**Swashbuckle.AspNetCore**) Documentação da API



Startup.cs Classe de inicialização do NET CORE

```
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.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Microsoft.OpenApi.Models;

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

// 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();
    }

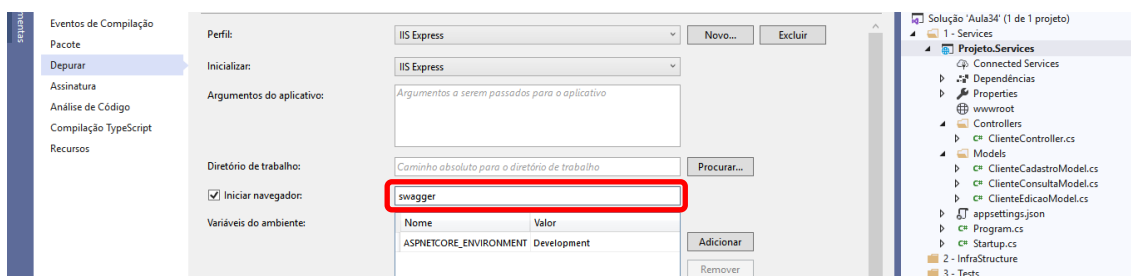
    #region Swagger

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

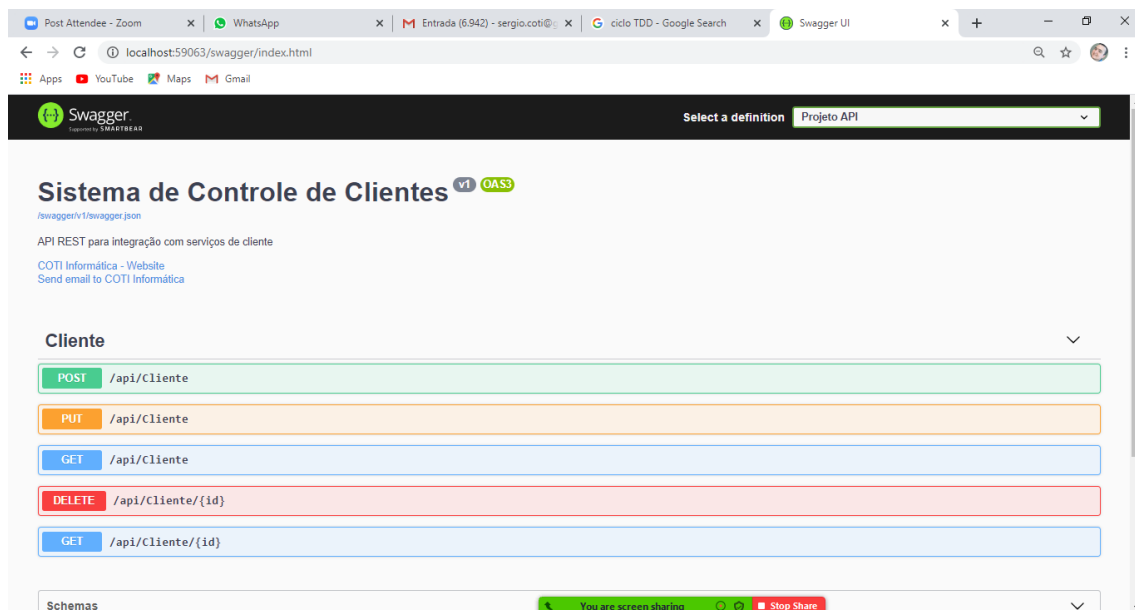
    #endregion

    app.UseMvc();
}
}
```

Modificando a página inicial do projeto para **Swagger**:

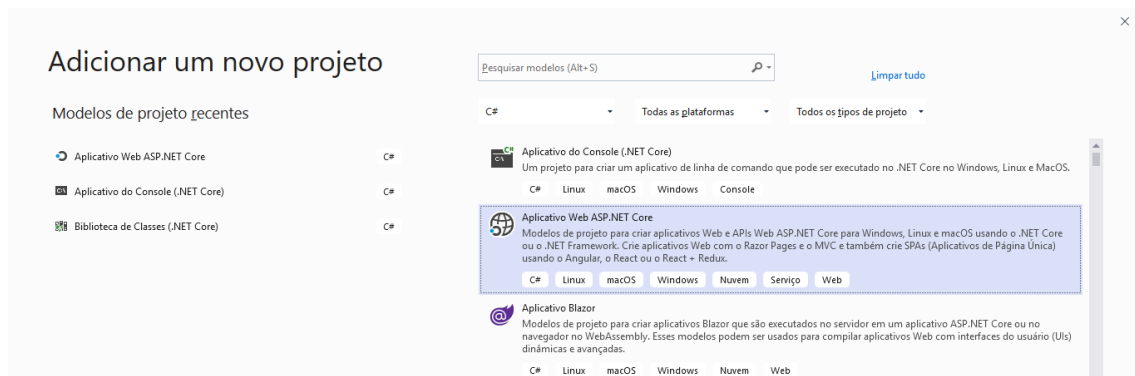


<http://localhost:59063/swagger/index.html>

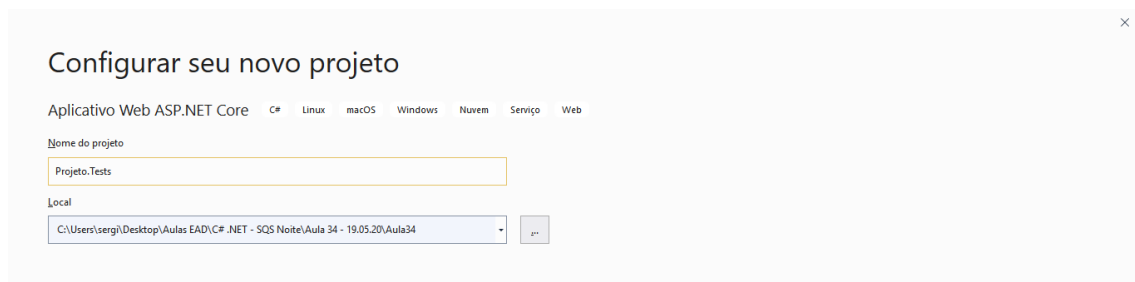


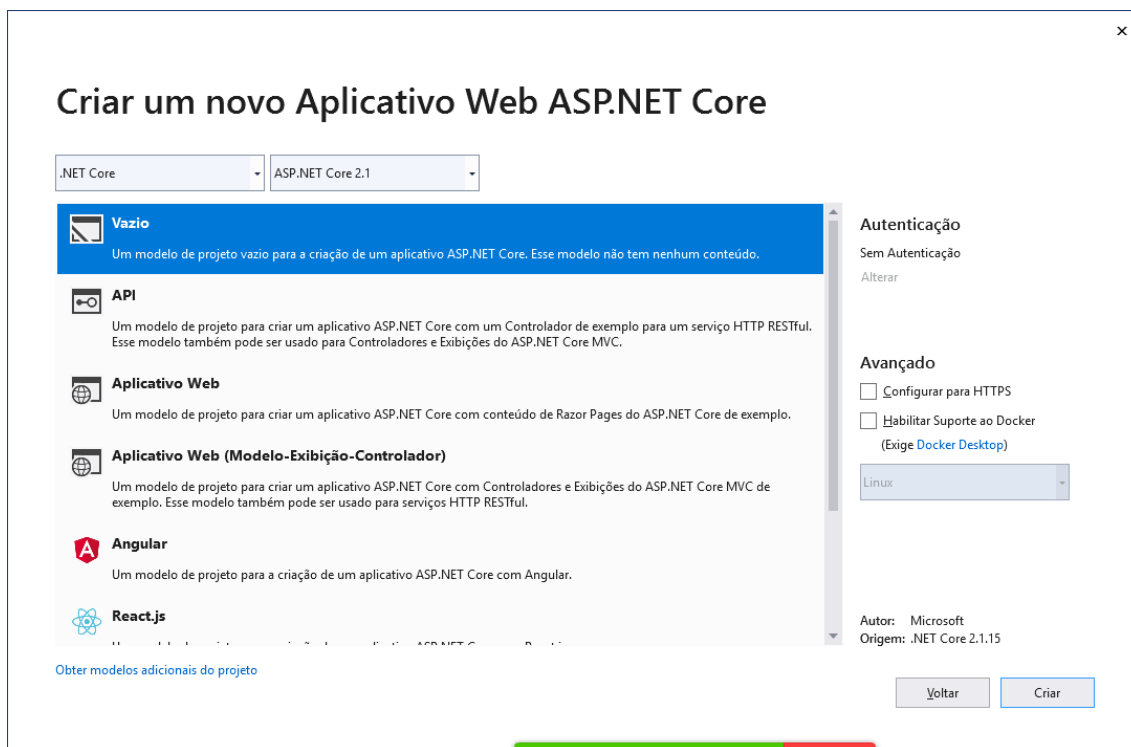
3 - Tests

Criando um projeto para testar a API:
(Aplicativo web ASP.NET Core)



Nome: **Projeto.Tests**



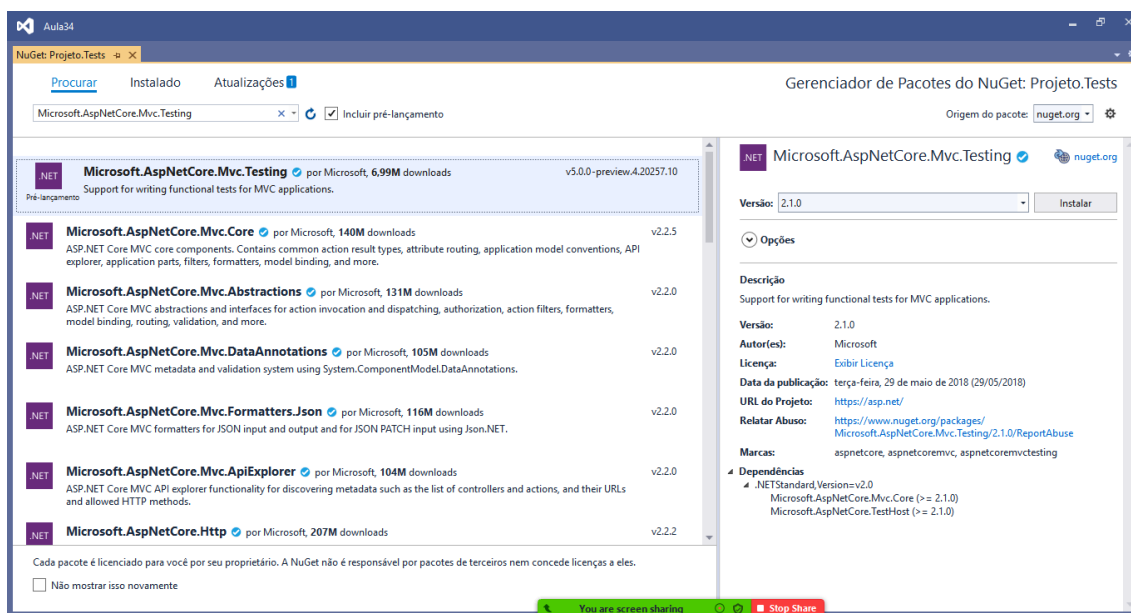


** Apague do projeto os itens:

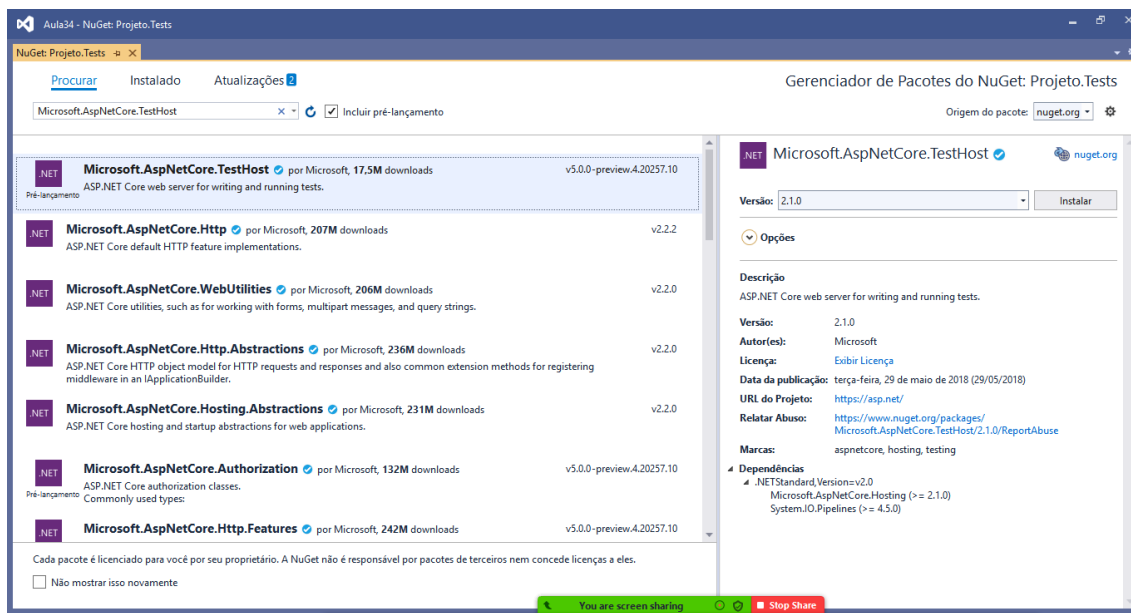
- /wwwroot
- Program.cs
- Startup.cs

Instalando as bibliotecas para execução de testes:

Microsoft.AspNetCore.Mvc.Testing (v2.1.0)

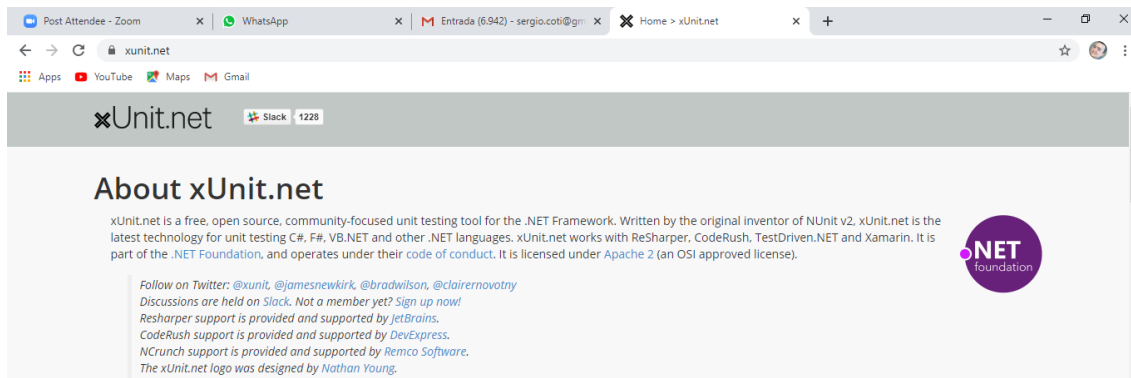


Microsoft.AspNetCore.TestHost (v2.1.0)

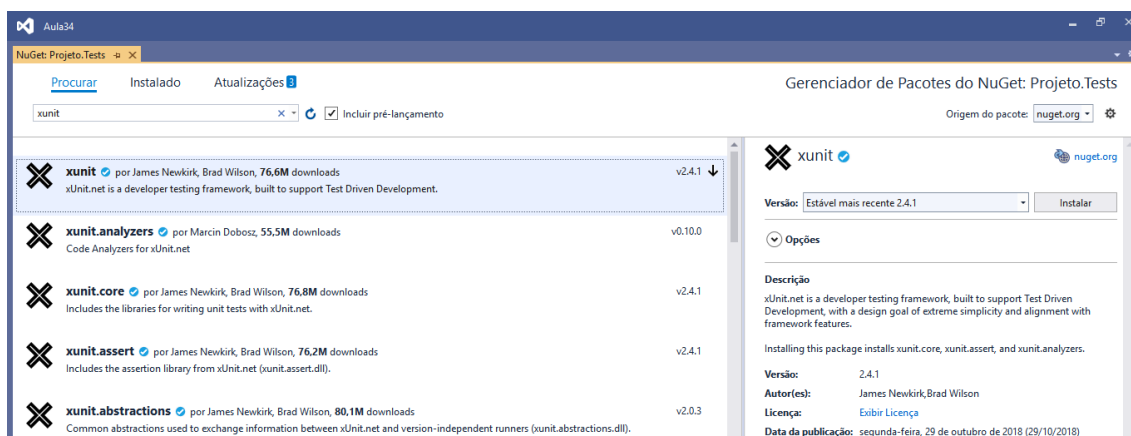


XUnit (<https://xunit.net/>)

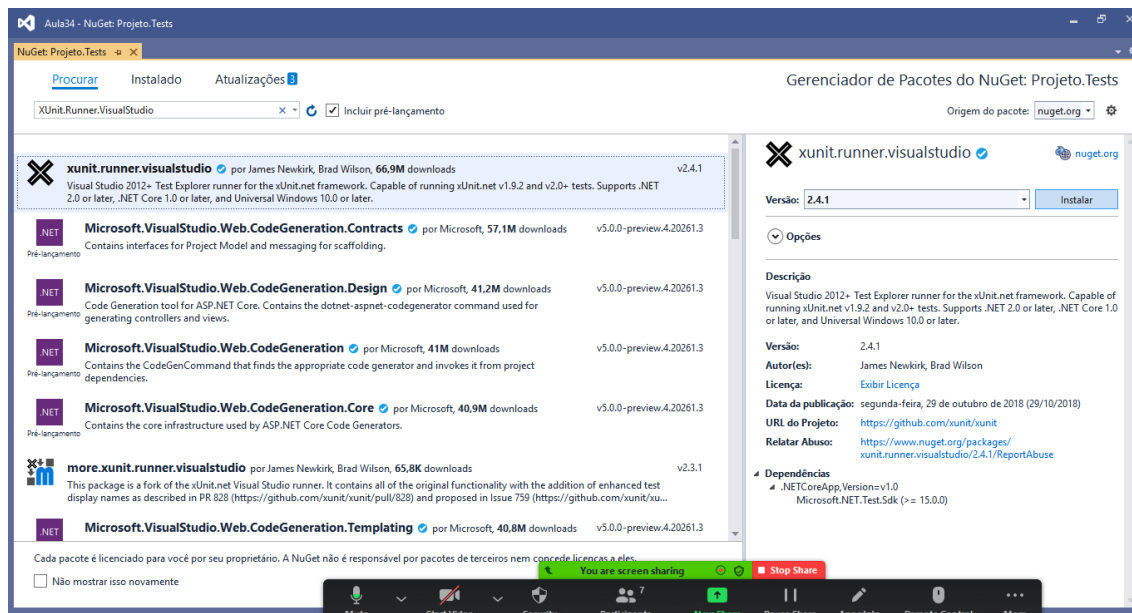
Principal biblioteca para desenvolvimento de testes em aplicações .NET



Xunit (v2.4.1)

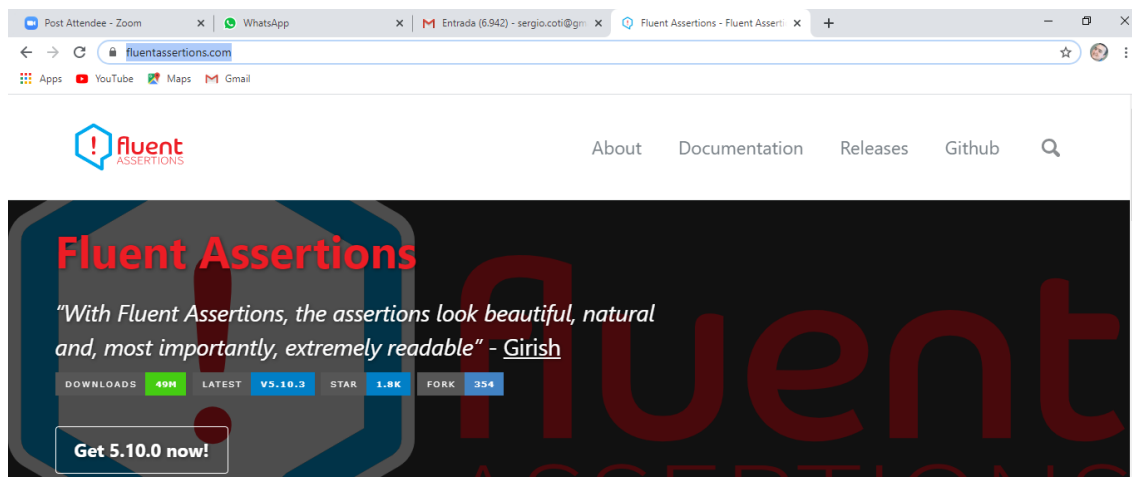


XUnit.Runner.VisualStudio (v2.4.1)

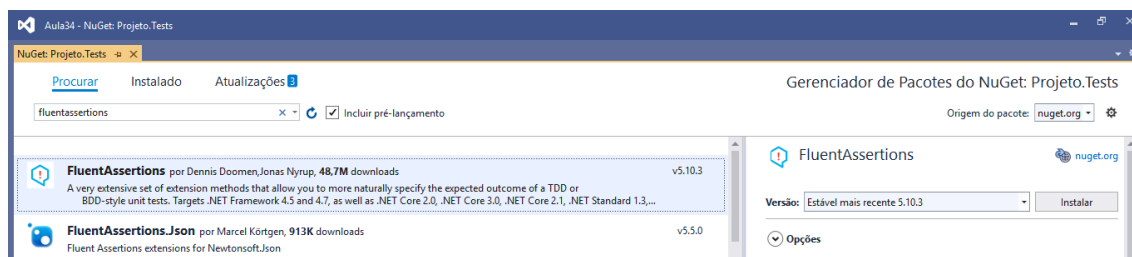


FluentAssertions (<https://fluentassertions.com/>)

Biblioteca utilizada para implementar os critérios de teste (Maior que, igual a, de acordo com, etc)

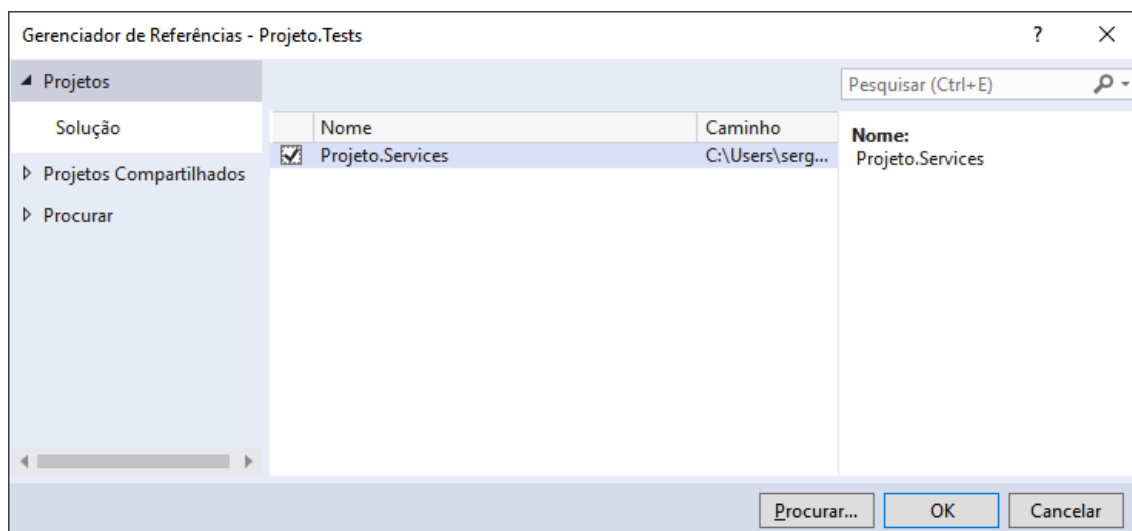


FluentAssertions (v5.10.3)



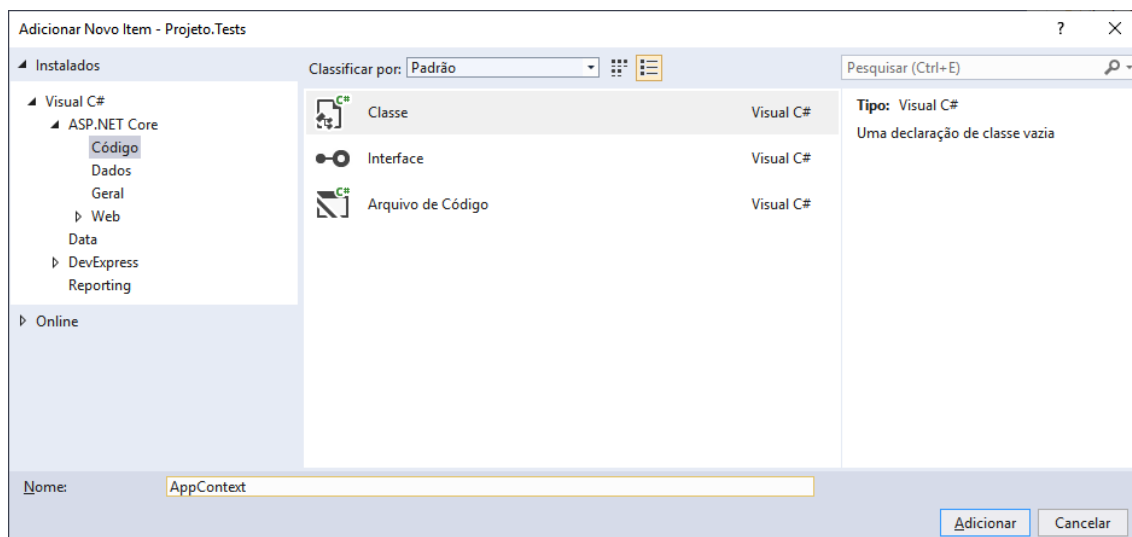
Adicionando referência no projeto

Tests para o projeto **Services**:



AppContext

Classe que iremos criar para configurar e preparar o projeto de teste.



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

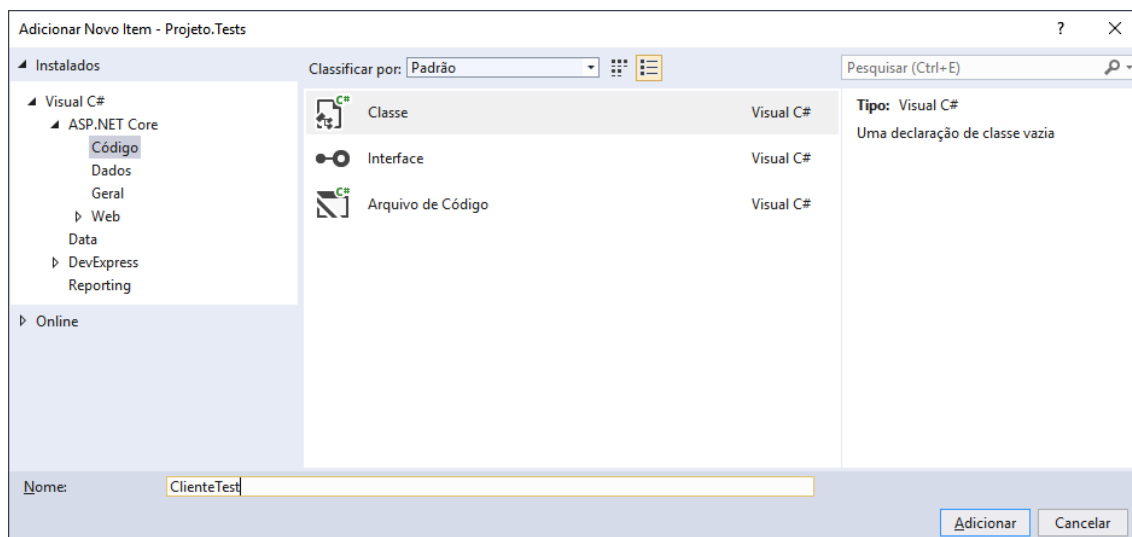
```
namespace Projeto.Tests
{
    public class AppContext
    {
        //classe para executar chamadas HTTP na API..
        //prop + 2x[tab]
        public HttpClient Client { get; set; }

        //servidor de testes
        private readonly TestServer testServer;

        //contrutor -> ctor + 2x[tab]
        public AppContext()
        {
            //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()
                .UseStartup<Services.Startup>());

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

Criando uma classe para testar os serviços de cliente:



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

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";
}
}
```

Criando métodos para testar cada serviço da API:
/api/Cliente (POST, PUT, DELETE e GET)

```
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);
    }
}
```

Adicionando um método para testar requisições com erro de validação:

```
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);
        }
    }
}
```



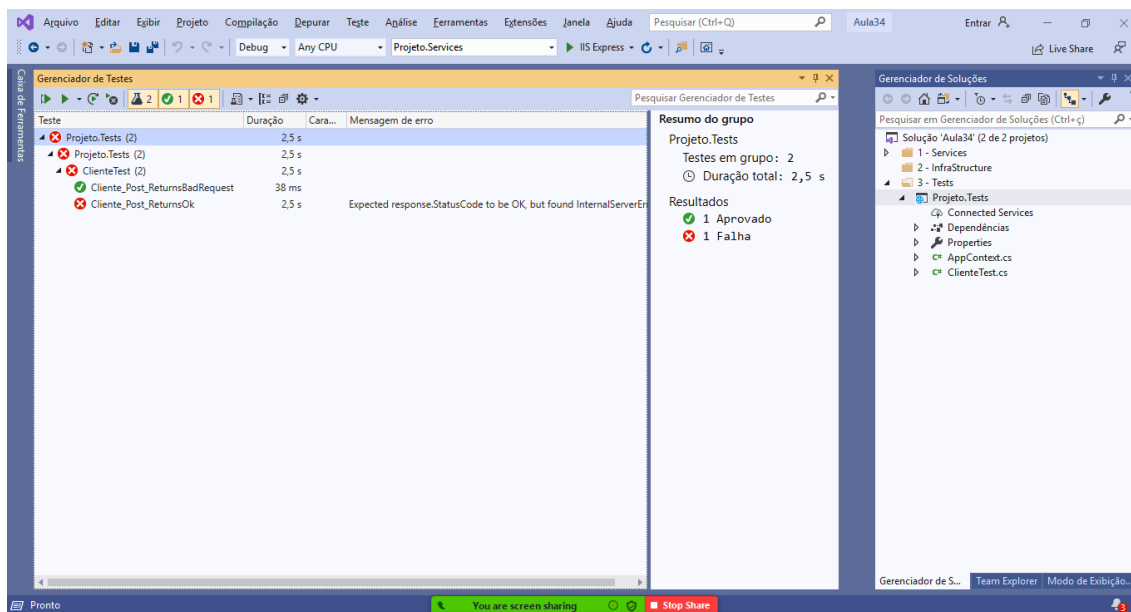
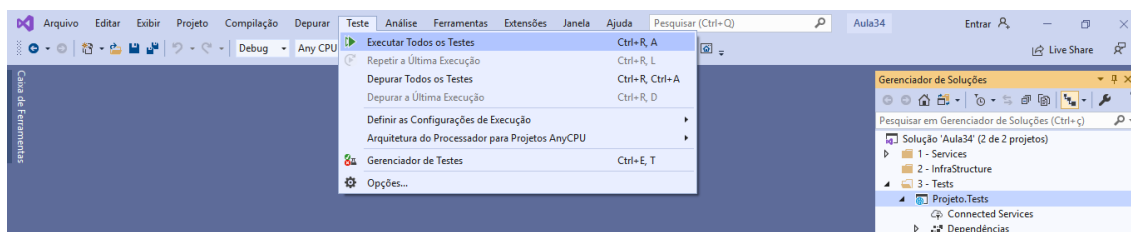
```
[Fact] //método para execução de teste do XUnit
//async -> método executado como uma Thread (assíncrono)
public async Task Cliente_Post_ReturnsBadRequest()
{
    //preencher os campos da model
    var model = new ClienteCadastroModel()
    {
        Nome = string.Empty, //vazio
        Email = string.Empty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

Executando os testes:



Teste	Duração	Cara...
Projeto.Tests (2)	2,5 s	
Projeto.Tests (2)	2,5 s	
ClienteTest (2)	2,5 s	
Cliente_Post_ReturnsBadRequest	38 ms	
Cliente_Post_ReturnsOk	2,5 s	

Voltando na classe ClienteTest

Criando os demais métodos de teste

```
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);

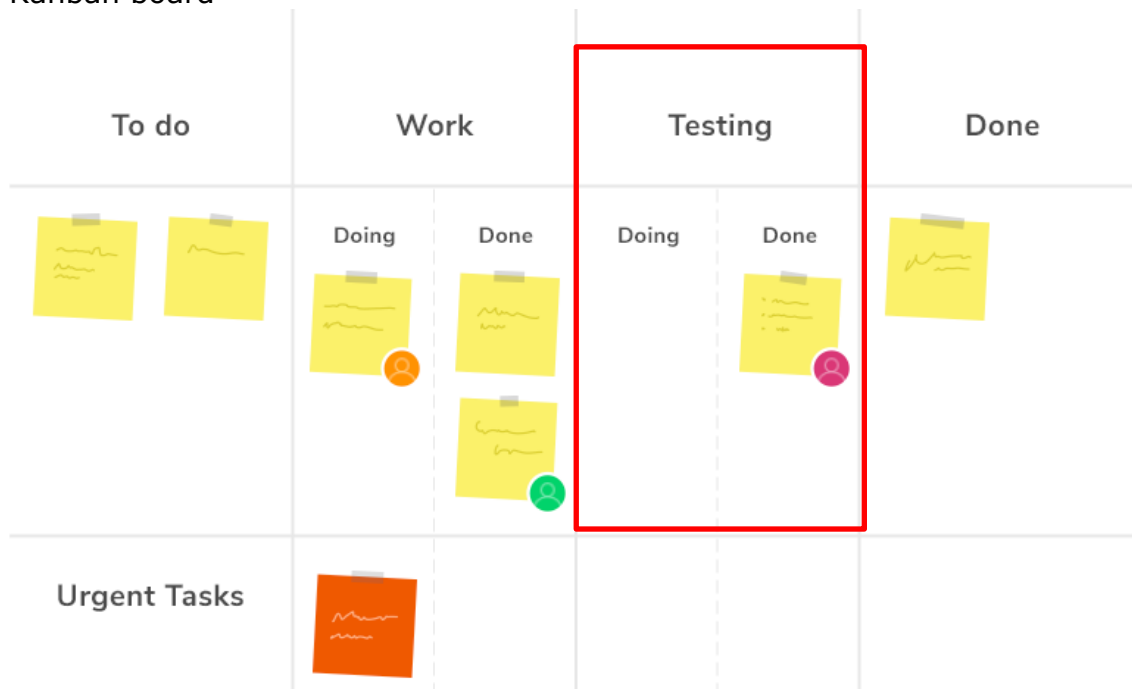
    //critério de teste (Serviço da API retornar HTTP OK (200))
    response.StatusCode.Should().Be(HttpStatusCode.OK);
}

[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();

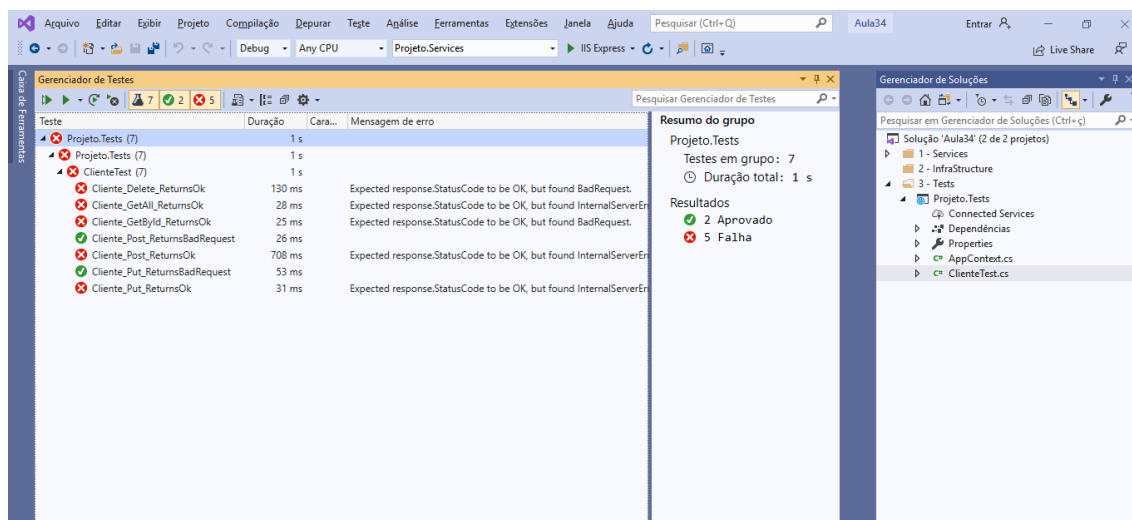
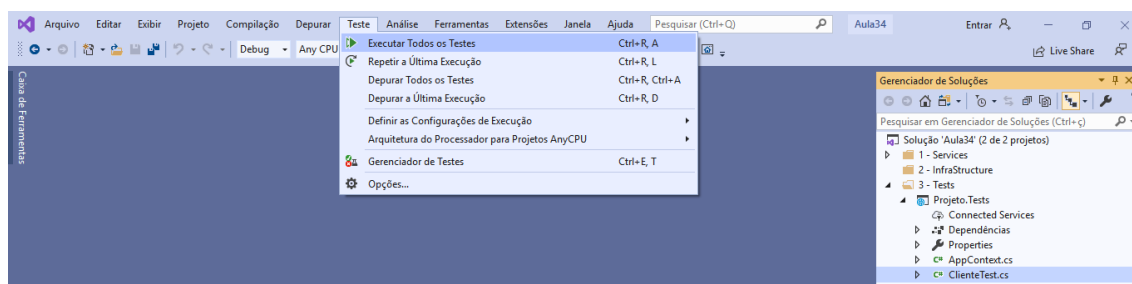
    //executando o serviço da API..
    var response = await appContext.Client.GetAsync(endpoint + "/" + id);

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


TDD aplicado a metodologias ágeis:
Kanban board



Executando:

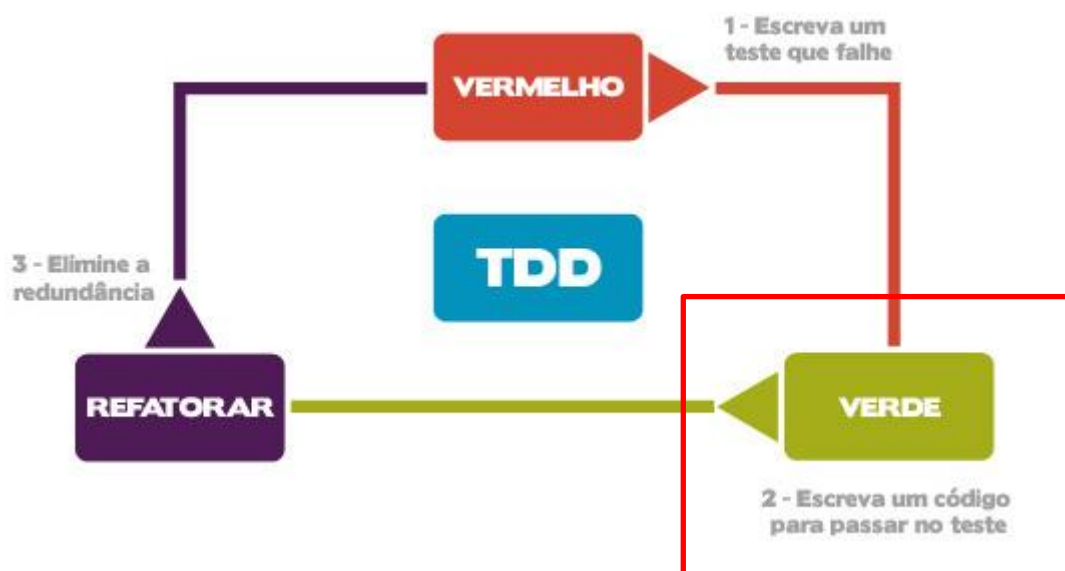


Resultado:

Gerenciador de Testes		
		
Teste	Duração	Cara...
✖ Projeto.Tests (7)	1 s	
✖ Projeto.Tests (7)	1 s	
✖ ClienteTest (7)	1 s	
✖ Cliente_Delete_ReturnsOk	130 ms	
✖ Cliente_GetAll_ReturnsOk	28 ms	
✖ Cliente_GetById_ReturnsOk	25 ms	
✔ Cliente_Post_ReturnsBadRequest	26 ms	
✖ Cliente_Post_ReturnsOk	708 ms	
✔ Cliente_Put_ReturnsBadRequest	53 ms	
✖ Cliente_Put_ReturnsOk	31 ms	

Voltando ao Ciclo TDD (**Test Driven Development**)

Próxima etapa: **Entregar uma aplicação que passe no teste**



Continua...