Engenharia de software 4

Testes unitários

Teste de aceitação

Caso de teste

Marcelino Dias de Oliveira	**RA: SP3022196**
Rafael dos Santos Ferreira	**RA: SP3021734**
Rubens Henrique do Nascimento Lotufo	**RA: SP3024121**

Sumário2

1. Testes unitários

Testes unitários são aqueles onde é possível verificar a execução da menor parte testável do seu código. Geralmente essa parte é uma função ou um método. Utilizar a automação desses testes é crucial para garantirmos que o código realize aquilo que nós desenvolvedores esperamos.

Assim, devemos escrever nossos testes em um cenário limitado e específico. Testes mais complexos e que abrangem mais funcionalidades devem ser realizados através de testes de integração e serviços. Assim, os testes unitários devem certificar os códigos dentro do seu controle, abstraindo as questões de infraestrutura.

Nesse projeto irei demonstrar os testes unitários com .NET Core 3.1 e xUnit. O primeiro passo foi criado um projeto, de uma API com Asp.NetCore utilizando a linguagem C#, ORM para conexão com banco de dados e Swagger para Documentação da API. Basicamente este Projeto faz o cadastro, atualização deleção de dados como Nome, Email, Data de cadastro e data de Update.

Figura 1: Classe UserController

Fonte: Autoria própria

Assert(Afirma) é a parte final do testes. Onde montamos as comparações dos resultados obtidos e esperados dos testes.

Figura 2: Assert

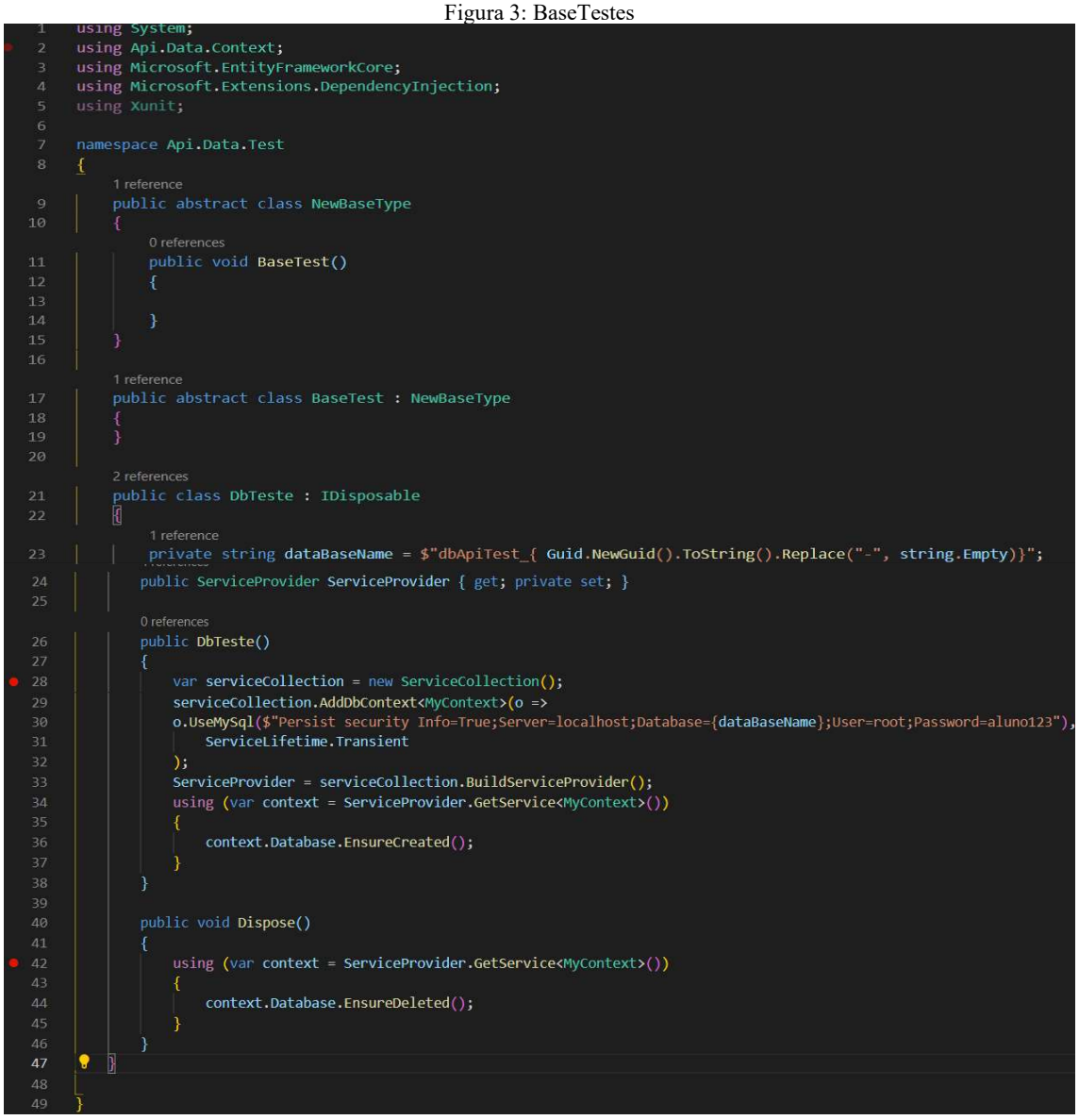
```
var _registroExiste = await _repositorio.ExistAsync(_registroAtualizado.Id);
Assert.True(_registroExiste);

var _registroSelecionado = await _repositorio.SelectAsync(_registroAtualizado.Id);
Assert.NotNull(_registroSelecionado);
Assert.Equal(_registroAtualizado.Email, _registroSelecionado.Email);
Assert.Equal(_registroAtualizado.Name, _registroSelecionado.Name);

var _todosRegistros = await _repositorio.SelectAsync();
Assert.NotNull(_todosRegistros);
Assert.True(_todosRegistros.Count() > 1);

var _removeu = await _repositorio.DeleteAsync(_registroSelecionado.Id);
Assert.True(_removeu);

var _usuarioPadrao = await _repositorio.FindByLogin("rafael_santospg@yahoo.com.br");
Assert.NotNull(_usuarioPadrao);
Assert.Equal("rafael_santospg@yahoo.com.br", _usuarioPadrao.Email);
Assert.Equal("Administrador", _usuarioPadrao.Name);
```

Fonte: Autoria própria

1.2. Resultado dos Testes Unitários

1.2.1 Classe “BaseTestes” responsável pelo teste dos Métodos da pasta Api.Data.

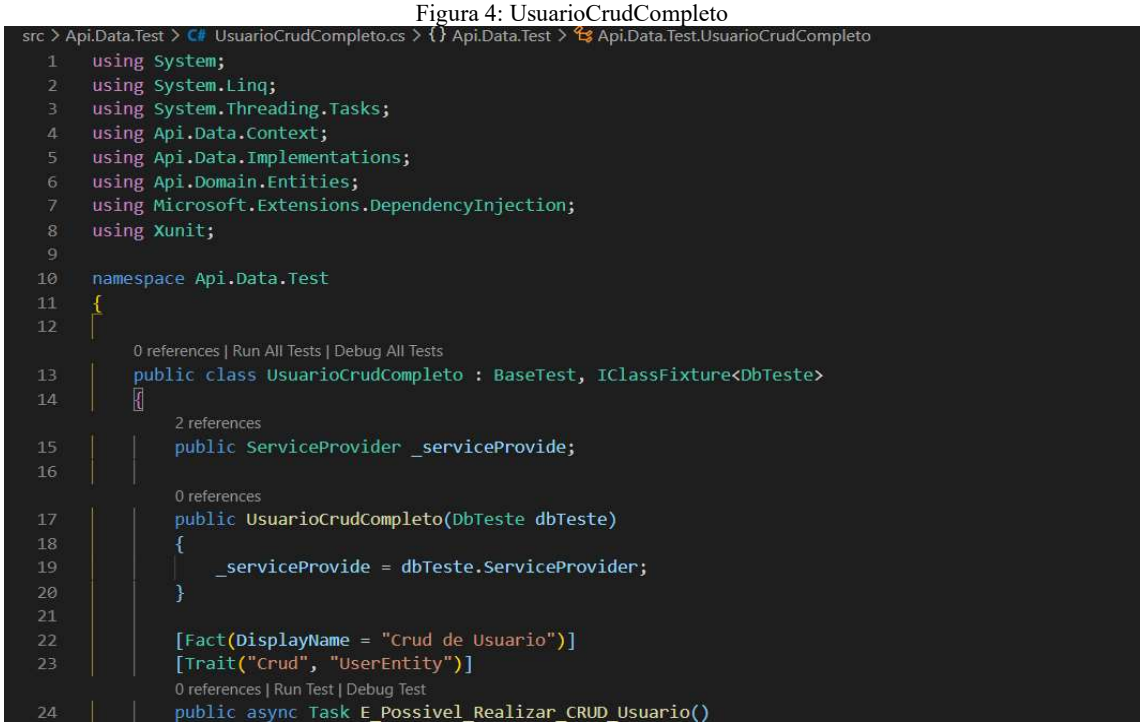
Figura 3: BaseTestes



```
using System;
using Api.Data.Context;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Xunit;

namespace Api.Data.Test
{
    1 reference
    public abstract class NewBaseType
    {
        0 references
        public void BaseTest()
        {

        }
    }

    1 reference
    public abstract class BaseTest : NewBaseType
    {
    }

    2 references
    public class DbTeste : IDisposable
    {
        1 reference
        private string dataBaseName = $"dbApiTest_{ Guid.NewGuid().ToString().Replace("-", string.Empty)}";
        public ServiceProvider ServiceProvider { get; private set; }

        0 references
        public DbTeste()
        {
            var serviceCollection = new ServiceCollection();
            serviceCollection.AddDbContext<MyContext>(o =>
            o.UseMySql($"Persist security Info=True;Server=localhost;Database={dataBaseName};User=root;Password=aluno123"),
                ServiceLifetime.Transient
            );
            ServiceProvider = serviceCollection.BuildServiceProvider();
            using (var context = ServiceProvider.GetService<MyContext>())
            {
                context.Database.EnsureCreated();
            }
        }

        public void Dispose()
        {
            using (var context = ServiceProvider.GetService<MyContext>())
            {
                context.Database.EnsureDeleted();
            }
        }
    }

}
```

1.2.2 Classe “UsuarioCrudCompleto” responsável pelo teste dos Métodos da pasta Api.Data.

Figura 4: UsuarioCrudCompleto



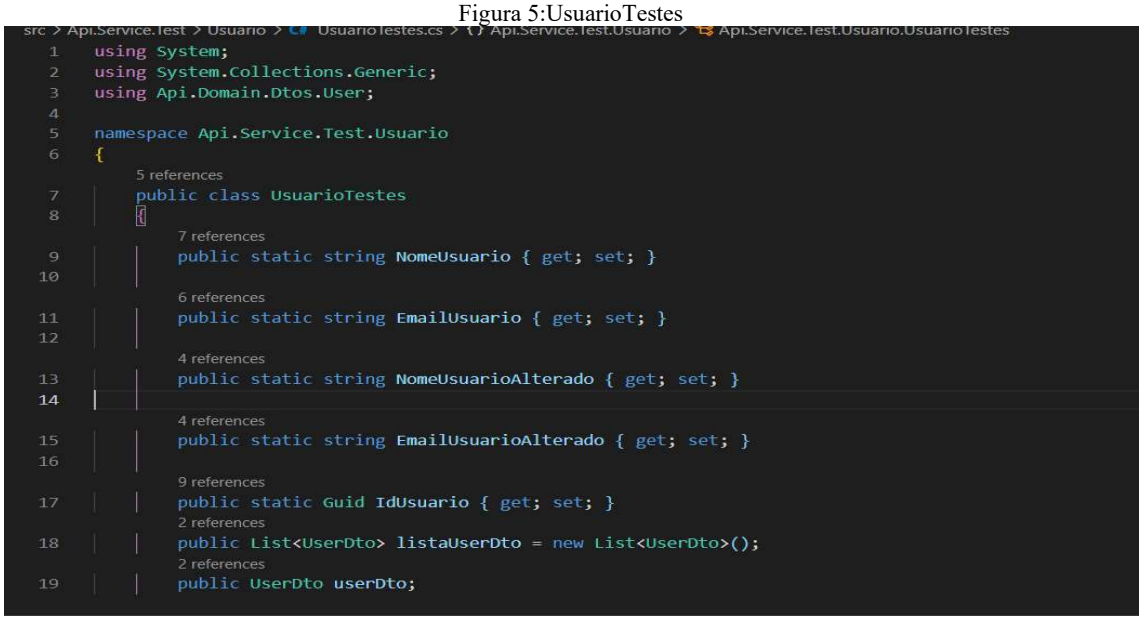
```
        {
            using (var context = _serviceProvide.GetService<MyContext>())
            {
                UserImplementation _repositorio = new UserImplementation(context);
                UserEntity _entity = new UserEntity
                {
                    Email = Faker.Internet.Email(),
                    Name = Faker.Name.FullName()
                };
                var _registroCriado = await _repositorio.InsertAsync(_entity);
                Assert.NotNull(_registroCriado);
                Assert.Equal(_entity.Email, _registroCriado.Email);
                Assert.Equal(_entity.Name, _registroCriado.Name);
                Assert.False(_registroCriado.Id == Guid.Empty);

                _entity.Name = Faker.Name.First();
                var _registroAtualizado = await _repositorio.UpdateAsync(_entity);
                Assert.NotNull(_registroAtualizado);
                Assert.Equal(_entity.Email, _registroAtualizado.Email);
                Assert.Equal(_entity.Name, _registroAtualizado.Name);

                var _registroExiste = await _repositorio.ExistAsync(_registroAtualizado.Id);
                Assert.True(_registroExiste);

                var _registroSelecionado = await _repositorio.SelectAsync(_registroAtualizado.Id);
                Assert.NotNull(_registroSelecionado);
                Assert.Equal(_registroAtualizado.Email, _registroSelecionado.Email);
                Assert.Equal(_registroAtualizado.Name, _registroSelecionado.Name);

                var _todosRegistros = await _repositorio.SelectAsync();
                Assert.NotNull(_todosRegistros);
                Assert.True(_todosRegistros.Count() > 1);

                var _removeu = await _repositorio.DeleteAsync(_registroSelecionado.Id);
                Assert.True(_removeu);

                var _usuarioPadrao = await _repositorio.FindByLogin("rafael_santospg@yahoo.com.br");
                Assert.NotNull(_usuarioPadrao);
                Assert.Equal("rafael_santospg@yahoo.com.br", _usuarioPadrao.Email);
                Assert.Equal("Administrador", _usuarioPadrao.Name);
            }
        }
    }
}
```

Resultado:

```
----- Test Execution Summary -----

Api.Data.Test.UsuarioCrudCompleto.E_Possivel_Realizar_CRUD_Usuario:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

1.2.3 Classe “UsuarioTestes” responsável pelo teste do Método Usuario da pasta Api.Service.

Figura 5:UsuarioTestes



```
5 references
public UserDtoCreate userDtoCreate;
3 references
public UserDtoCreateResult userDtoCreateResult;
3 references
public UserDtoUpdate userDtoUpdate;
2 references
public UserDtoUpdateResult userDtoUpdateResult;

0 references
public UsuarioTestes()
{
    IdUsuario = Guid.NewGuid();
    NomeUsuario = Faker.Name.FullName();
    EmailUsuario = Faker.Internet.Email();
    NomeUsuarioAlterado = Faker.Name.FullName();
    EmailUsuarioAlterado = Faker.Internet.Email();

    for (int i = 0; i < 10; i++)
    {
        var dto = new UserDto()
        {
            Id = Guid.NewGuid(),
            Name = Faker.Name.FullName(),
            Email = Faker.Internet.Email()
        };
        listaUserDto.Add(dto);
    }
    userDto = new UserDto
    {
        Id = IdUsuario,
        Name = NomeUsuario,
        Email = EmailUsuario
    };
    userDtoCreate = new UserDtoCreate
    {
        Name = NomeUsuario,
        Email = EmailUsuario
    };
    userDtoCreateResult = new UserDtoCreateResult
    {
        Id = IdUsuario,
        Name = NomeUsuario,
        Email = EmailUsuario,
        CreateAt = DateTime.UtcNow
    };
    userDtoUpdate = new UserDtoUpdate
    {
        Id = IdUsuario,
        Name = NomeUsuarioAlterado,
        Email = EmailUsuarioAlterado
    };
    userDtoUpdateResult = new UserDtoUpdateResult
    {
        Id = IdUsuario,
        Name = NomeUsuarioAlterado,
```

1.2.4 Classe “QuandoForExecutadoCreate” responsável pelo teste do Método Create da pasta Api.Service.

Figura 6:QuandoForExecutadoCreate

```
namespace Api.Service.Test.Usuario
{
    0 references | Run All Tests | Debug All Tests
    public class QuandoForExecutadoCreate : UsuarioTestes
    {
        2 references
        private IUserService _service;
        3 references
        private Mock<IUserService> _serviceMock;

        [Fact(DisplayName = "É possivel executar o Método Create.")]

        0 references | Run Test | Debug Test
        public async Task E_Possivel_executar_Metodo_Create()
        {
            _serviceMock = new Mock<IUserService>();
            _serviceMock.Setup(m => m.Post(userDtoCreate)).ReturnsAsync(userDtoCreateResult);
            _service = _serviceMock.Object;

            var result = await _service.Post(userDtoCreate);
            Assert.NotNull(result);
            Assert.Equal(NomeUsuario, result.Name);
            Assert.Equal(EmailUsuario, result.Email);
        }
    }
}
```

Resultado:

```
[xUnit.net 00:00:00.99]   Finished:    Api.Service.Test
----- Test Execution Summary -----

Api.Service.Test.Usuario.QuandoForExecutadoCreate.E_Possivel_executar_Metodo_Create:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

1.2.5 Classe “QuandoForExecutadoGet” responsável pelo teste do Método Get da pasta Api.Service.

Figura 7:QuandoForExecutadoGet

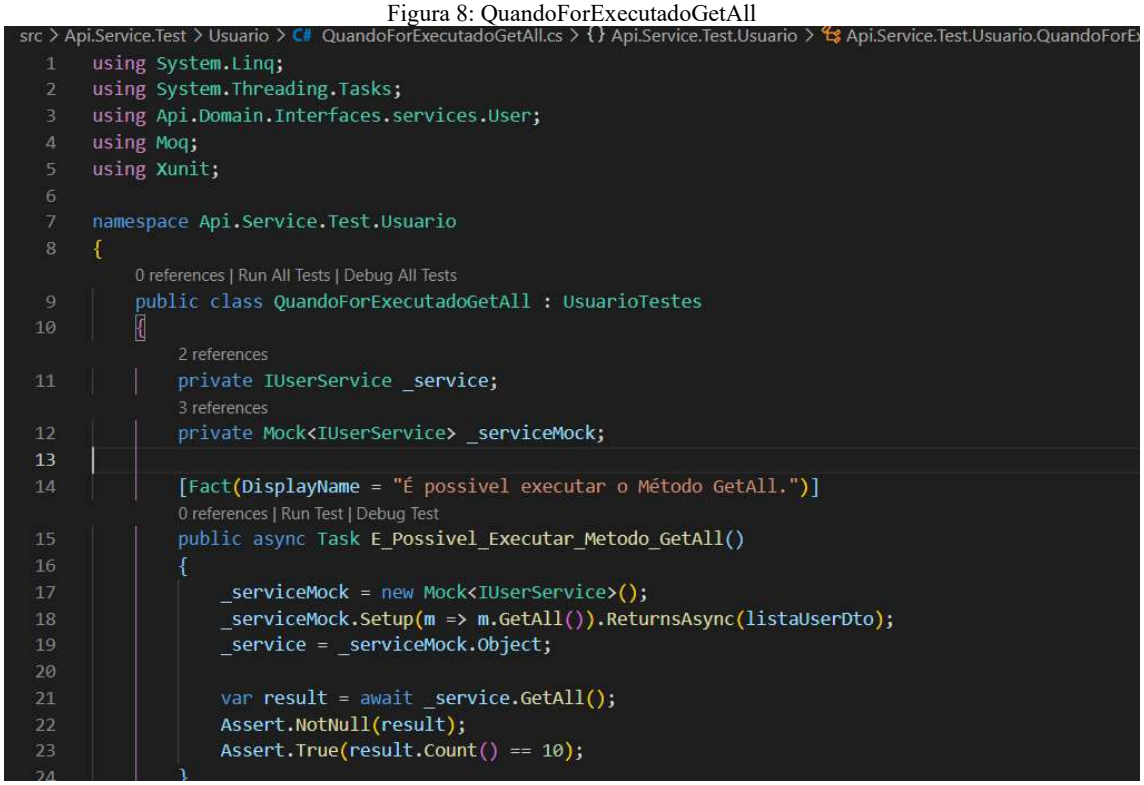
```
src > Api.Service.Test > Usuario > C# QuandoForExecutadoGet.cs > { } Api.Service.Test.Usuario > Api.Service.Test.Usuario.QuandoForExecut
1   using System.Collections.Generic;
2   using System.Linq;
3   using System.Threading.Tasks;
4   using Api.Domain.Dtos.User;
5   using Api.Domain.Interfaces.services.User;
6   using Moq;
7   using Xunit;
8
9   namespace Api.Service.Test.Usuario
10  {
        0 references | Run All Tests | Debug All Tests
11      public class QuandoForExecutadoGet : UsuarioTestes
12      {
            4 references
13          private IUserService _service;
            6 references
14          private Mock<IUserService> _serviceMock;
15
16          [Fact(DisplayName = "É possivel executar o Método GET.")]
            0 references | Run Test | Debug Test
17          public async Task E_Possivel_executar_Metodo_Get()
18          {
19              _serviceMock = new Mock<IUserService>();
20              _serviceMock.Setup(m => m.Get(IdUsuario)).ReturnsAsync(userDto);
21              _service = _serviceMock.Object;
22
23              var result = await _service.Get(IdUsuario);
24              Assert.NotNull(result);
25              Assert.True(result.Id == IdUsuario);
26              Assert.Equal(NomeUsuario, result.Name);
27
28              var _listResult = new List<UserDto>();
29              _serviceMock = new Mock<IUserService>();
30              _serviceMock.Setup(m => m.GetAll()).ReturnsAsync(_listResult.AsEnumerable);
31              _service = _serviceMock.Object;
32
33              var _resultEmpty = await _service.GetAll();
34              Assert.Empty(_resultEmpty);
35              Assert.True(_resultEmpty.Count() == 0);
36          }
37
38      }
39  }
40
```

Resultado:

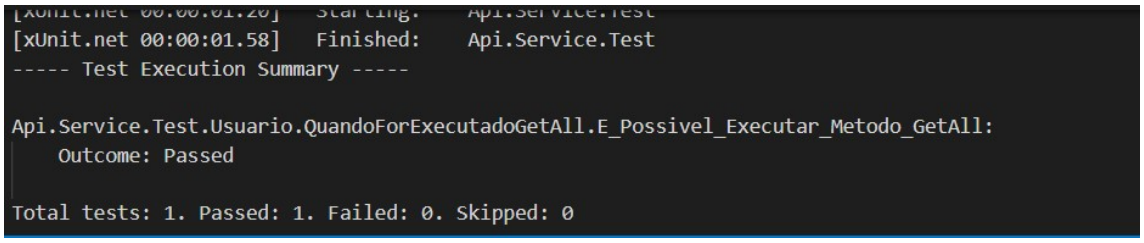
```
[xUnit.net 00:00:01.21]   Starting:    Api.Service.Test
[xUnit.net 00:00:01.66]   Finished:    Api.Service.Test
----- Test Execution Summary -----

Api.Service.Test.Usuario.QuandoForExecutadoGet.E_Possivel_executar_Metodo_Get:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

1.2.6 Classe “QuandoForExecutadoGetAll” responsável pelo teste do Método GetAll da pasta Api.Service.

Figura 8: QuandoForExecutadoGetAll

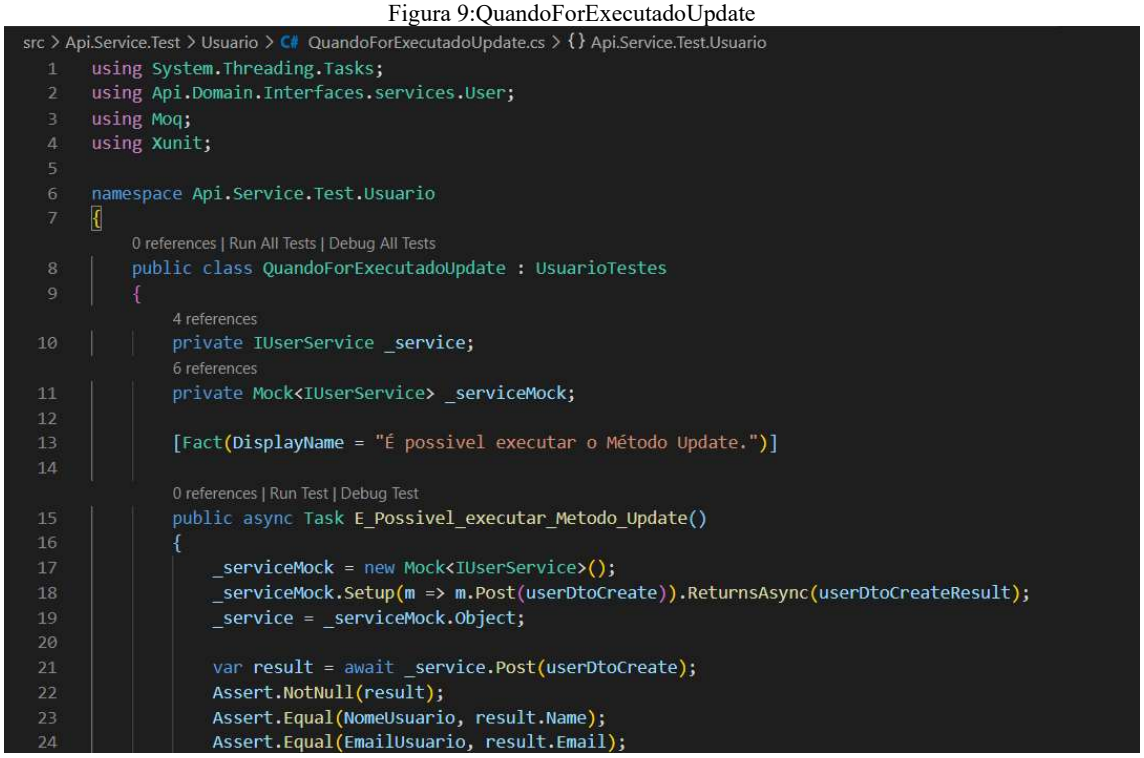


Resultado:



1.2.7 Classe “QuandoForExecutadoUpdate” responsável pelo teste do Método Update da pasta Api.Service.

Figura 9:QuandoForExecutadoUpdate



Resultado:

```
[xUnit.net 00:00:01.66]   Finished:    Api.Service.Test
----- Test Execution Summary -----

Api.Service.Test.Usuario.QuandoForExecutadoUpdate.E_Possivel_executar_Metodo_Update:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

1.2.8 Classe “QuandoForExecutadoDelete” responsável pelo teste do Método Delete da pasta Api.Service.

Figura 10:QuandoForExecutadoDelete

```
src > Api.Service.Test > Usuario > QuandoForExecutadoDelete.cs > { } Api.Service.Test.Usuario > Api.Service.Test.U
1   using System;
2   using System.Threading.Tasks;
3   using Api.Domain.Interfaces.services.User;
4   using Moq;
5   using Xunit;
6
7   namespace Api.Service.Test.Usuario
8   {
        0 references | Run All Tests | Debug All Tests
9       public class QuandoForExecutadoDelete : UsuarioTestes
10      {
            4 references
11          private IUserService _service;
            6 references
12          private Mock<IUserService> _serviceMock;
13
14          [Fact(DisplayName = "É possivel executar o Método Delete.")]
15
            0 references | Run Test | Debug Test
16          public async Task E_Possivel_executar_Metodo_Delete()
17          {
18              _serviceMock = new Mock<IUserService>();
19              _serviceMock.Setup(m => m.Delete(It.IsAny<Guid>())).ReturnsAsync(true);
20              _service = _serviceMock.Object;
21
22              var deletado = await _service.Delete(IdUsuario);
23              Assert.True(deletado);
24
25              _serviceMock = new Mock<IUserService>();
26              _serviceMock.Setup(m => m.Delete(It.IsAny<Guid>())).ReturnsAsync(false);
27              _service = _serviceMock.Object;
28
29              deletado = await _service.Delete(Guid.NewGuid());
30              Assert.False(deletado);
31          }
32      }
33  }
```

Resultado:

```
----- Test Execution Summary -----

Api.Service.Test.Usuario.QuandoForExecutadoUpdate.E_Possivel_executar_Metodo_Update:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

1.2.9 Classe "QuandoForExecutadoDelete" responsável pelo teste do Método Delete da pasta Api.Service.

Figura 11:QuandoForExecutadoDelete

```
6
7   namespace Api.Service.Test.Usuario
8   {
        0 references | Run All Tests | Debug All Tests
9       public class QuandoForExecutadoDelete : UsuarioTestes
10      {
            4 references
11          private IUserService _service;
            6 references
12          private Mock<IUserService> _serviceMock;
13
14          [Fact(DisplayName = "É possivel executar o Método Delete.")]
15
            0 references | Run Test | Debug Test
16          public async Task E_Possivel_executar_Metodo_Delete()
17          {
18              _serviceMock = new Mock<IUserService>();
19              _serviceMock.Setup(m => m.Delete(It.IsAny<Guid>())).ReturnsAsync(true);
20              _service = _serviceMock.Object;
21
22              var deletado = await _service.Delete(IdUsuario);
23              Assert.True(deletado);
24
25              _serviceMock = new Mock<IUserService>();
26              _serviceMock.Setup(m => m.Delete(It.IsAny<Guid>())).ReturnsAsync(false);
27              _service = _serviceMock.Object;
28
29              deletado = await _service.Delete(Guid.NewGuid());
30              Assert.False(deletado);
```

Resultado:

```
----- Test Execution Summary -----

Api.Service.Test.Usuario.QuandoForExecutadoDelete.E_Possivel_executar_Metodo_Delete:
    Outcome: Passed

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0
```

2. Teste de aceitação

Cenários	Aprovado	Reprovado
Classe UsuarioCrudCompleto realização teste do Método Create da Api.Data responsável pela comunicação com o banco de dados Mysql	**X**	
Classe UsuarioCrudCompleto realização teste do Método Get da Api.Data responsável pela comunicação com o banco de dados Mysql	**X**	

Classe UsuarioCrudCompleto realização teste do Método GetAll da Api.Data responsável pela comunicação com o banco de dados Mysql	**X**	
Classe UsuarioCrudCompleto realização teste do Método Update da Api.Data responsável pela comunicação com o banco de dados Mysql	**X**	
Classe UsuarioCrudCompleto realização teste do Método Delete da Api.Data responsável pela comunicação com o banco de dados Mysql	**X**	
Classe QuandoForExecutadoCreate testa o Método Create da pasta Api.Service.	**X**	
Classe QuandoForExecutadoGet testa o Método Get da pasta Api.Service.	**X**	
Classe QuandoForExecutadoGetAll testa o Método GetAll da pasta Api.Service.	**X**	
Classe QuandoForExecutadoDelete testa o Método Delete da pasta Api.Service.	**X**	

3. Caso de teste

Objetivo	**Ação**	**Resultado Esperado**
Obter Token	O usuário realiza uma requisição para http:localhost:5000/api/login para obter a autorização	Retorna um Token

Objetivo	**Ação**	**Resultado Esperado**
Fazer Autenticação do usuário	O usuário envia o token na requisição para obter a autorização	Exibi a mensagem: “Usuário Logado com sucesso”

Objetivo	Ação	Resultado Esperado
Fazer logout do usuário	O usuário para de fazer a requisição por 60 segundos ou clicar no botão “logout” localizado na parte superior direita.	Exibi a mensagem: “uthenticated: false”

Objetivo	Ação	Resultado Esperado
Obter dados do Banco	O usuário envia o token na requisição Get http://localhost:5000/api/Users para retornar todos os dados.	Em response aparece a coluna “code: 200” “description: success”

Objetivo	Ação	Resultado Esperado
Adicionar dados no Banco	O usuário envia o token na requisição Post http://localhost:5000/api/Users para retornar todos os dados.	Em response aparece a coluna “code: 200” “description: success”

Objetivo	Ação	Resultado Esperado
Atualizar dados no Banco	O usuário envia o token na requisição Put http://localhost:5000/api/Users para retornar todos os dados.	Em response aparece a coluna “code: 200” “description: success”

Objetivo	Ação	Resultado Esperado
deletar dados no Banco	O usuário envia o token na requisição Delete http://localhost:5000/api/Users/{id} para retornar todos os dados.	Em response aparece a coluna “code: 200” “description: success”

Objetivo	Ação	Resultado Esperado
Obter dado por ID do Banco	O usuário envia o token na requisição Get http://localhost:5000/api/Users/{id} para retornar todos os dados.	Em response aparece a coluna “code: 200” “description: success”

<PWebAPI>
Padrão IEEE 829-1998

Plano de Teste

Versão 1.0

Identificador: ES4A4 - DT

Referências

O *"Plano de Teste"* segue todas as normas e metodologias adotas no *"Plano de Projeto - MinhaWebAPI."* E ainda, segue o padrão internacional IEEE 829 para sua criação.

Introdução

Este plano de API possui os seguintes objetivos :

- garantir que todos os requisitos do projeto foram atendidos
- observar e documentar o comportamento aplicação
- definir o processo de teste

Itens de teste

O sistema a ser testado é um API que funciona como uma interface de comunição entre aplicação e banco de dados *MySQL*, a escolha da linguagem de programação da aplicação é critério do desenvolvedor.

As seguintes classes e métodos da API serão testadas:

- Get()
- getAll()
- Update()
- Delete()

Funcionalidade que serão testadas

Todas as operações de CRUD *(Create, Read, Update e Delete)* serão testadas.

Funcionalidades que não serão testada

Não se aplica, pois todas as funcionalidades serão testadas.

Abordagem

O teste de unidade será feito pela equipe de desenvolvimento com supervisão do gerente de desenvolvimento. A equipe de testes e gerente de projetos são responsáveis pelo desenvolvimento do teste de aceitação.

Critérios de Aceite

Todas as funcionalidades devem em perfeito funcionamento. Durante todas as etapas de teste uma equipe será responsável na geração de um relatório avaliativo que será encaminhado para o gerente de projeto.

Critérios de Suspensão

O teste deve ser pausado caso a API não consiga implementar as operações CRUD.

Produtos

Após o término dos testes, a equipe de testes deve gerar relatórios para documentar os eventos vistos.

Tarefas de teste

- planejamento dos testes
- documentar metodologia e entregar para a equipe de testes
- preparar o ambiente de teste
- executar o teste
- gerar relatórios dos testes feitos

Requisitos para ambiente de teste

Item	Detalhes
WorkStation	CPU: dual core 2,0 Ghz ou superior
	RAM: 8gb
	Armazenamento: 120 GB
	Banco de Dados: MySQL 8.0.21
	IDE: VisualStudio 2019
	SO: Windows 10 64bits

Responsabilidades

O gerente de projeto será responsável no acompanhamento dos testes, e todos os artefatos gerados devem ser entregue ao cliente. A equipe de desenvolvimento terá a função de executar, validar e documentar os testes unitários.

Treinamento e equipe

Será necessário ao menos um indivíduo da equipe de teste em tempo integral para poder conduzir todos os testes do projeto. Nenhum treinamento será necessário.

Planejamento

Todos os testes devem ser concluídos com uma semana de antecedência em relação a entrega do projeto.

Riscos

O único risco vigente é a não conclusão das atividades planejadas no tempo estipulado.

Aprovações

Gerente de desenvolvimento	
Gerente de projeto	
Gerente de testes	

Anexo

Histórias de usuário

Título: Adição de dados no sistema	**Prioridade: Alta**
Como <*Administrador*> **eu quero** <*a funcionalidade de INSERSÃO* >	

Título: Excluir dados do sistema	**Prioridade: Alta**
Como <*Administrador*> **eu quero** <*a funcionalidade EXCLUSÃO* >	

Título: Modificar dados do sistema	**Prioridade: Alta**
Como <*Administrador*> **eu quero** <*a funcionalidade ATUALIZAR* >	

Título: Mostrar dados do sistema	**Prioridade: Alta**
Como <*Administrador*> **eu quero** <*a funcionalidade VISUALIZAR* >	

Título: Interface de comunicação	**Prioridade: Alta**
Como <*Administrador*> **eu quero** <*uma via de comunicação com o banco de dados* >	