

Selenium com C#

Conteúdo

1	Objetivos.....	4
2	Requisitos necessários.....	4
3	Softwares necessários	5
4	Módulo 1 – Selenium.....	5
4.1	Instalação do Visual Studio.....	5
4.2	Alterando o framework de testes para MsTest Versão 1.....	8
4.3	Primeiro Teste	12
4.4	Executando o teste	13
4.5	Estrutura do projeto em C#.....	15
4.5.1	Exportando a solução	17
4.5.2	Importando a solução.....	19
4.5.3	Copiando arquivos para a solução.....	21
4.6	Inspecionando Elementos	24
4.6.1	Tipos de Identificadores	25
4.6.2	Css por Jquery.....	27
4.6.3	Buscando por xpath.....	29
4.7	Tipos de Elementos.....	32
4.7.1	Criando uma nova classe de teste	32
4.7.2	TextField	32
4.7.3	Botões	33
4.7.4	Link.....	33
4.7.5	Radio	33
4.7.6	Checkbox / Frame / Scroll To Element	34
4.7.7	Dropdown (Select).....	35
4.7.8	Tabela	35
4.7.9	Alert	36
4.7.10	Janela/Aba	37
4.7.11	Modal.....	37
4.7.12	Autocomplete	38
4.7.13	Mousehover.....	39
4.7.14	Upload de arquivo	39
4.7.15	Autenticação.....	40

4.7.16	Encerrando múltiplos processos de drivers.....	40
4.8	Esperas.....	42
4.8.1	Espera Implícita	42
4.8.2	Espera Explícita	42
4.9	Exemplo Login.....	46
4.9.1	Exercício:.....	49
4.9.2	Exercício:.....	50
4.10	Atributos de Inicialização e Finalização dos testes.....	51
4.11	Centralizando o driver	57
4.11.1	Exercício.....	59
4.12	Screenshots.....	60
4.12.1	Exercicio:.....	64
4.13	Page Objects	66
4.13.1	Notação FindsBy do Page Factory	66
4.13.2	Nomenclatura dos Elementos	66
4.13.3	Convertendo os testes para o padrão POM	67
4.13.4	Abordagens para o POM.....	72
4.13.5	Retorno de novas páginas, questão de contexto	73
4.13.6	Exercício.....	77
4.14	Lendo a massa de dados pelo Excel.....	80
4.14.1	Exercício.....	86
4.15	Faker	86
4.15.1	Exercício com faker.....	89
4.16	Arquivo de Configurações de Teste - .runsettings.....	96
4.16.1	Exercício.....	100
5	Módulo 2 - Behaviour Driven Development - BDD.....	101
5.1	Conceitualização.....	101
5.2	Estrutura Principal	102
5.3	Palavras-chave	102
5.4	Regras de Escrita.....	103
5.4.1	Definir a linguagem no inicio do documento	103
5.4.2	Funcionalidade.....	103
5.4.3	Cenário / Esquema do Cenário	103
5.4.4	Contexto	103

5.4.5	Dado / Quando / Então.....	103
5.4.6	Uso de Tags.....	104
5.4.7	Uso de Aspas.....	104
5.4.8	Doc String.....	104
5.4.9	Dados variáveis em Esquema do Cenário.....	104
5.4.10	Foco no comportamento	105
5.4.11	Refatoração	105
5.5	Specflow	106
5.6	Plugin Specflow Visual Studio.....	107
5.7	Specflow - Configuração	110
5.8	Specflow – Features e Step Definitions	110
5.9	Specflow – Alterando a língua e renomeando os passos	118
5.10	Specflow – Tabelas	121
5.11	Specflow – Lendo Tabelas com Assist.Dynamic	125
5.12	Specflow – Esquema do Cenário	126
5.13	Specflow – Scenario Context	129
5.14	Specflow – Context Injection.....	133
5.15	Specflow – Doc String	136
5.16	Specflow – Hooks.....	137
5.17	Specflow – Scoped Bindings	144
5.18	Specflow – Contexto	150
5.19	Atualizando para Specflow 3	151
5.20	Refazendo o projeto com BDD	151
5.21	Login Feature	152
5.22	Centralizando o driver no Hooks	159
5.23	Incluindo o arquivo .runsettings	165
5.24	Screenshots.....	165
5.25	Lendo de um arquivo Excel.....	173
5.26	Exercicio – Fale Conosco.....	179
5.27	Exercicio – Cadastro.....	180
6	Módulo 3 - Integração com Azure DevOps.....	184
6.1	Criar conta no Azure DevOps.....	184
6.2	Criando o primeiro projeto.....	184
6.3	Visão Geral do Projeto	186

6.4	Criando os casos de Teste.....	188
6.5	Subindo o projeto no repositório	189
6.5.1	Conectando com o repositório do Azure DevOps	189
6.5.2	Criando o arquivo tfignore.....	197
6.5.3	Fazendo o check-in das alterações	199
6.6	Associando o script a um caso teste.....	201
6.7	Configurando o Agent de Execução.....	204
6.7.1	Gerando um Token de acesso	204
6.7.2	Download do agent	206
6.8	Configurando o Build	211
6.8.1	Incluindo testes na fase de Build	223
6.9	Criando uma release.....	230
6.10	Configurando a release para execução via Test Plan	236
Anexo I		243

1 Objetivos

Introduzir os conceitos básicos de testes automatizados com Selenium e criar um framework para execução dos testes.

O framework será integrado ao Azure DevOps (Antigo VSTS) para execução dos casos de teste e visualização dos resultados na na ferramenta.

Este material foi criado como auxílio para o treinamento interno de testes automatizados.

2 Requisitos necessários

Conhecimentos em testes de software.

Conhecimento em lógica de programação e orientação a objetos.

Para quem necessitar rever, ou aprender os conceitos de programação e orientação a objetos em C# segue uma lista com algumas sugestões de cursos:

Udemy (Pago): <https://www.udemy.com/programacao-orientada-a-objetos-csharp/>

Youtube – (Gratuito em inglês):

<https://www.youtube.com/watch?v=ZsWGfbgBXsw&list=PL6tu16kXT9Pp3NFZgLbPZXEykeGQwxGSx>

Devmedia – (Gratuito): <https://www.devmedia.com.br/curso/introducao-a-programacao-com-csharp/368>

Apostila Caelum – (Gratuito) - <https://www.caelum.com.br/apostila-csharp-orientacao-objetos/>

3 Softwares necessários

Como este é projeto para estudos, usaremos a versão Community que é gratuita e irá atender nossos requisitos.

Para projetos comerciais é necessário o uso das versões Enterprise ou Professional.

Visual Studio Community 2017: <https://visualstudio.microsoft.com/pt-br/vs/older-downloads/>

O Selenium e as demais dependências serão baixadas através do próprio visual Studio.

Apenas para referência (não é necessário baixar ou instalar nada):

Site oficial Selenium: <https://www.seleniumhq.org/>

Site da Microsoft sobre MsTest (voltado para testes unitários): <https://docs.microsoft.com/pt-br/dotnet/core/testing/unit-testing-with-mstest>

Obs: Os sites de teste que serão usados aqui no curso não são de nossa propriedade.

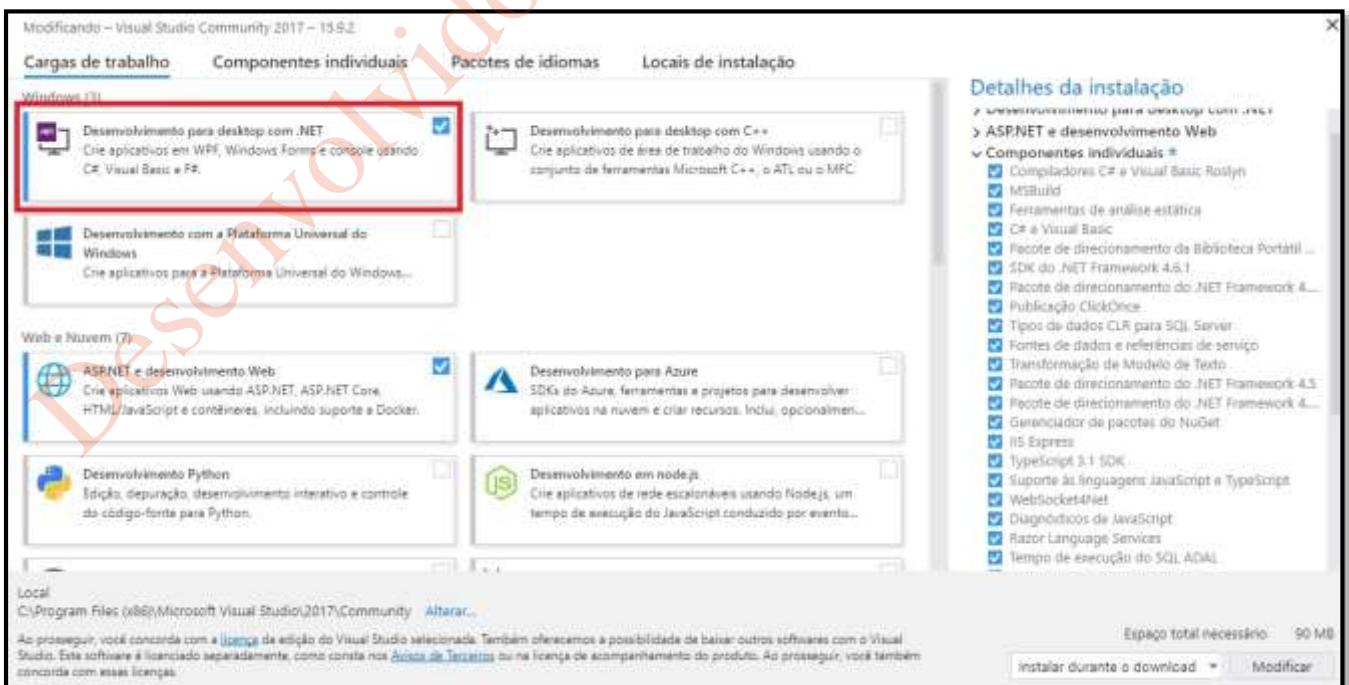
- <https://automacaocombatista.herokuapp.com/>
- <http://automationpractice.com/index.php>

4 Módulo 1 - Selenium

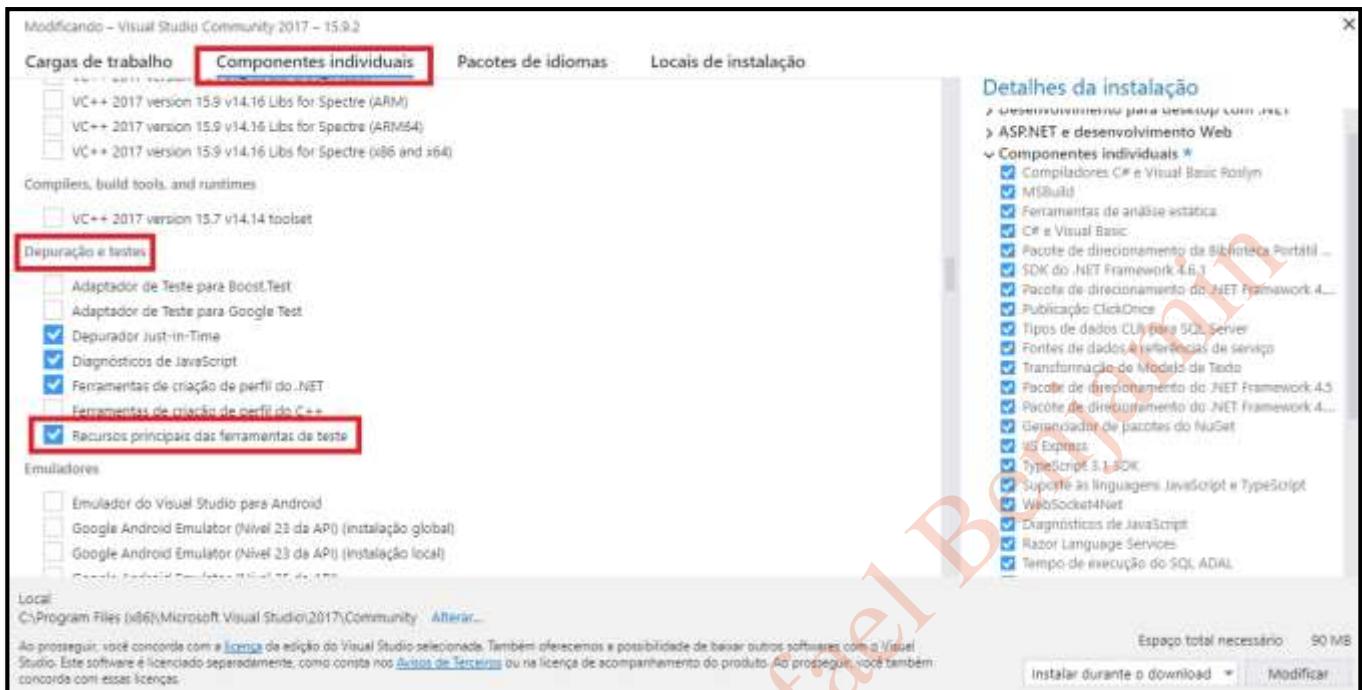
4.1 Instalação do Visual Studio

Durante a instalação do Visual Studio serão necessários selecionar apenas dois componentes.

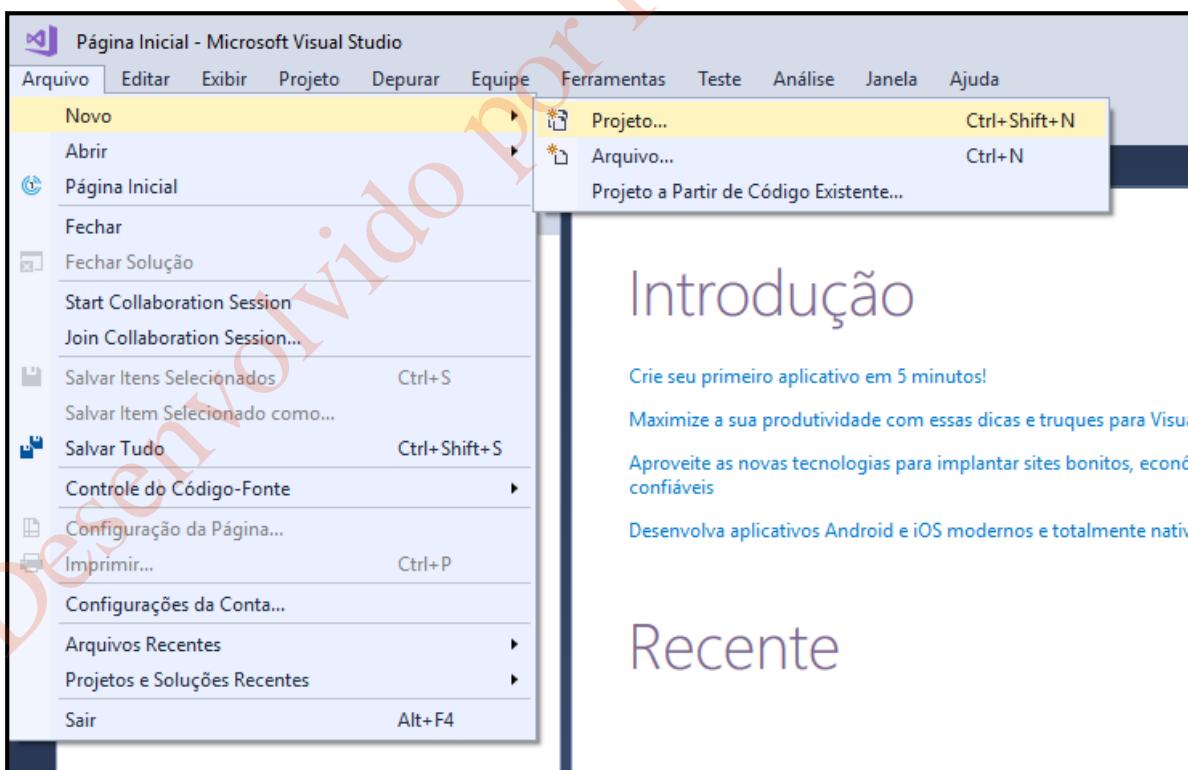
Em 'Cargas de trabalho', selecionar o item 'Desenvolvimento para desktop com .NET'.



Em ‘Componentes individuais’, na seção ‘Depuração e testes’, selecionar a opção ‘Recursos principais das ferramentas de teste’

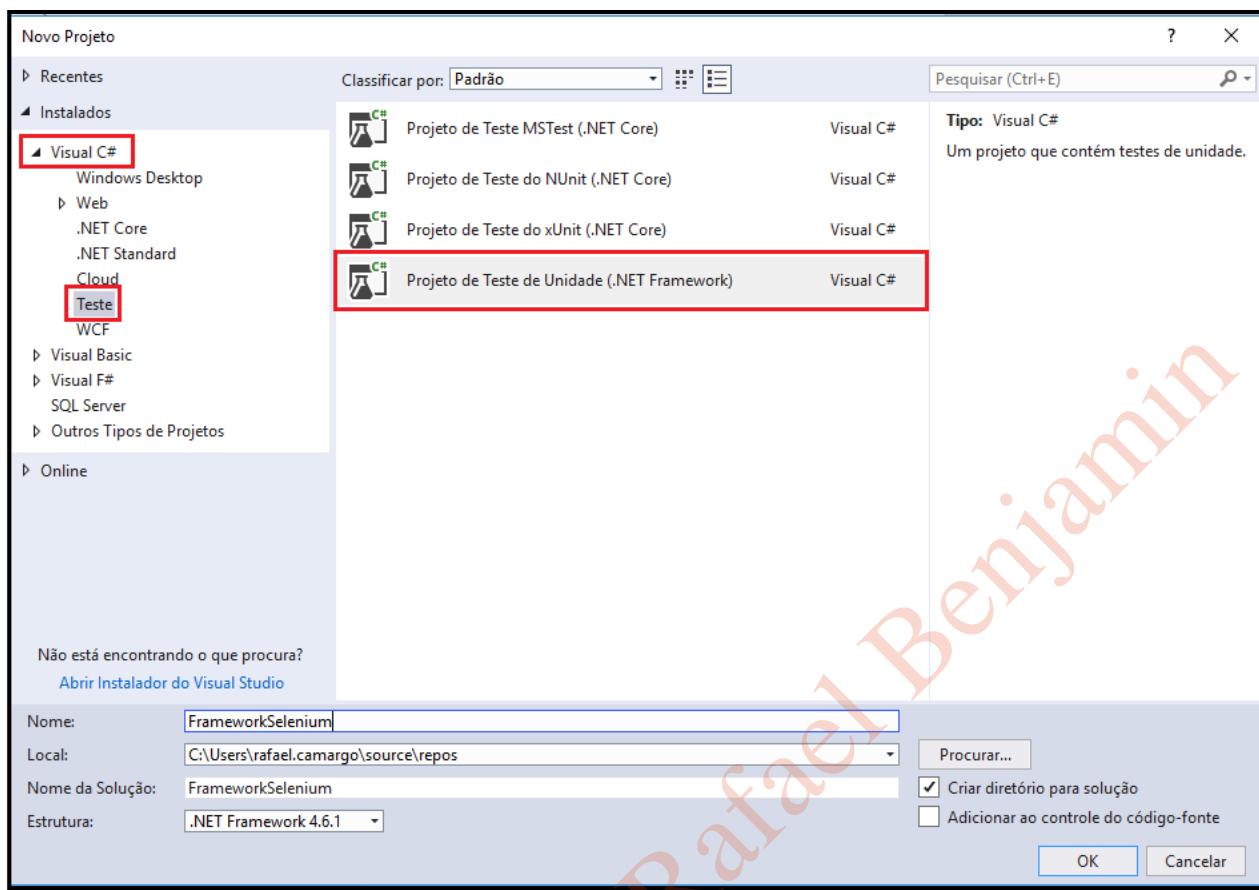


Após a instalação, acessar o menu **Arquivo > Novo > Projeto...**

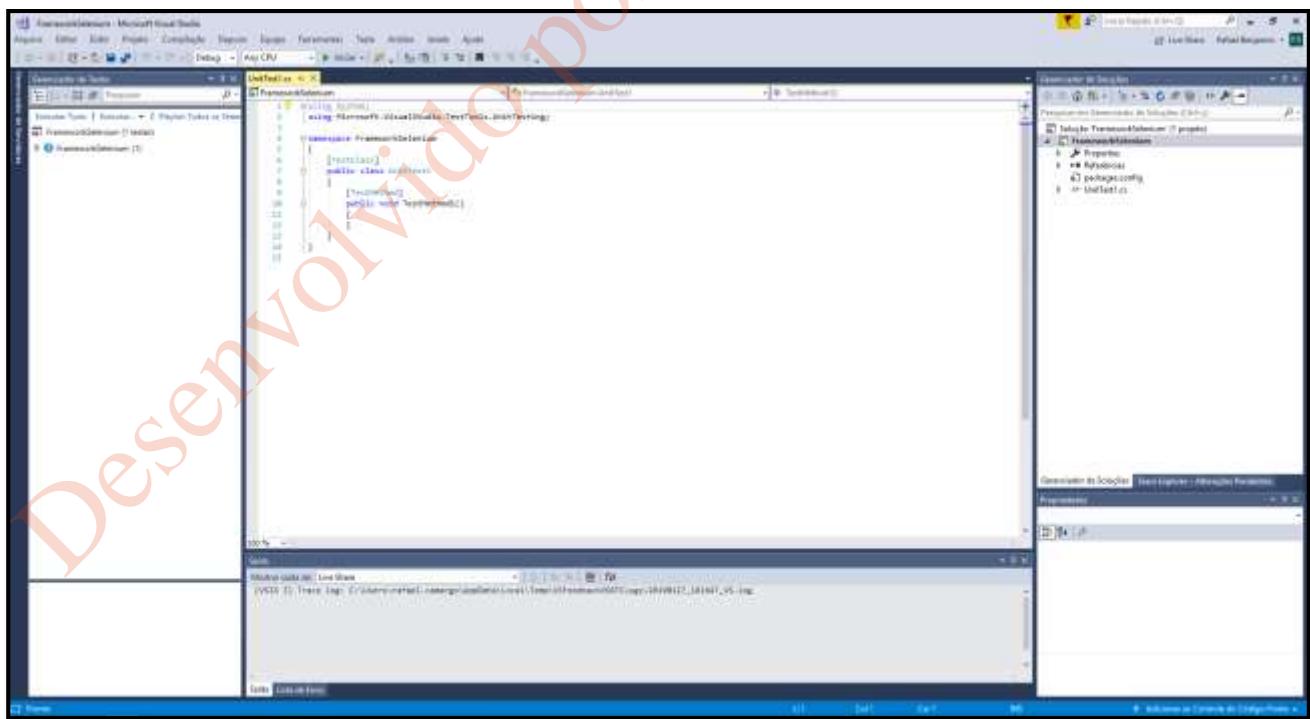


No menu lateral selecionar ‘Visual C#’, ‘Teste’, ‘Projeto de Teste de Unidade (.Net Framework)’

Preencha o nome e o diretório caso queira alterar e clique em **OK**.



O projeto será criado.



4.2 Alterando o framework de testes para MsTest Versão 1

Neste projeto usaremos a primeira versão do framework MsTest para que seja possível associar nossos testes automatizados aos cenários de testes criados no AzureDevOps ou TFS.

Fonte: <https://docs.microsoft.com/en-us/azure/devops/test/associate-automated-test-with-test-case?view=azdevops&viewFallbackFrom=vsts>

Ou <https://github.com/MicrosoftDocs/vsts-docs/blob/master/docs/test/associate-automated-test-with-test-case.md>

Atualmente este recurso de associação dos casos de teste ainda não está disponível para demais frameworks de testes em C#, mas está planejado para uma implementação futura.

Update : Em 09/01/2019 a documentação foi atualizada, o suporte para os demais frameworks está sendo realizado, porém ainda com algumas limitações:

Q: What types of tests are supported?

A: These are the limitations for each type of test:

- Coded UI test, Selenium tests, and unit tests written using Version 1 of the MSTest framework can be associated with a test case.
- Tests that use MSTest v2, NUnit, and xUnit frameworks can be associated with a test case workitem when using Visual Studio 15.9 Preview 2 or later. However, these tests cannot be run using Microsoft Test Manager and XAML builds.
- Tests that use the .NET core framework can be associated with a test case workitem when using Visual Studio 15.9 Preview 2 or later. To run the .NET core tests the appropriate target framework must be specified in a `runsettings` file. However, these tests cannot be run using Microsoft Test Manager and XAML builds.
- Tests that use other test frameworks such as Chutzpah (for JavaScript tests such as Mocha or QUnit), or Jest cannot be associated with a test case.
- Associating generic tests may work, but running these tests is not supported.

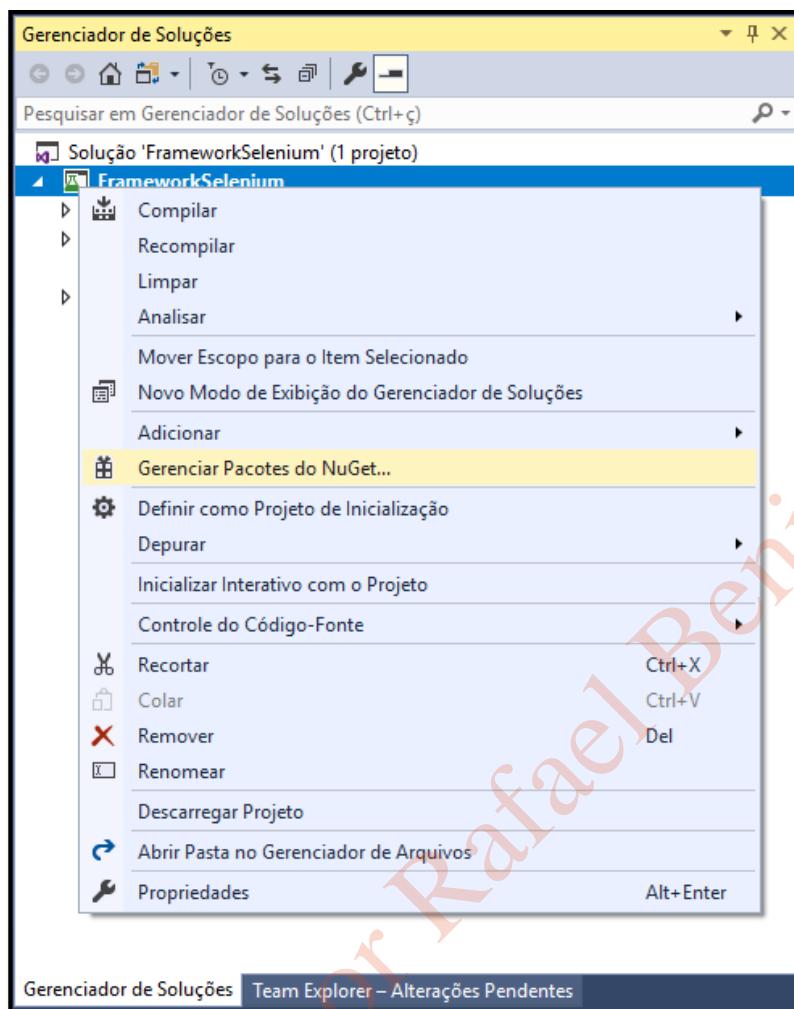
<https://github.com/MicrosoftDocs/vsts-docs/commit/60dd10551a2bae7d171c6608d0e5e362d9bb2654#diff-085d7b6f11d523ac749927ab6b42a014R100>

NOTE: At present this procedure is not supported for associating MSTest V2 tests or tests written in NUnit and XUnit. This capability is planned for a future release.

Update: Trecho removido em 13/02/2019 - <https://github.com/MicrosoftDocs/vsts-docs/commit/c9f241e26da67b5f6aaa89c86ae74372045b0343#diff-085d7b6f11d523ac749927ab6b42a014>

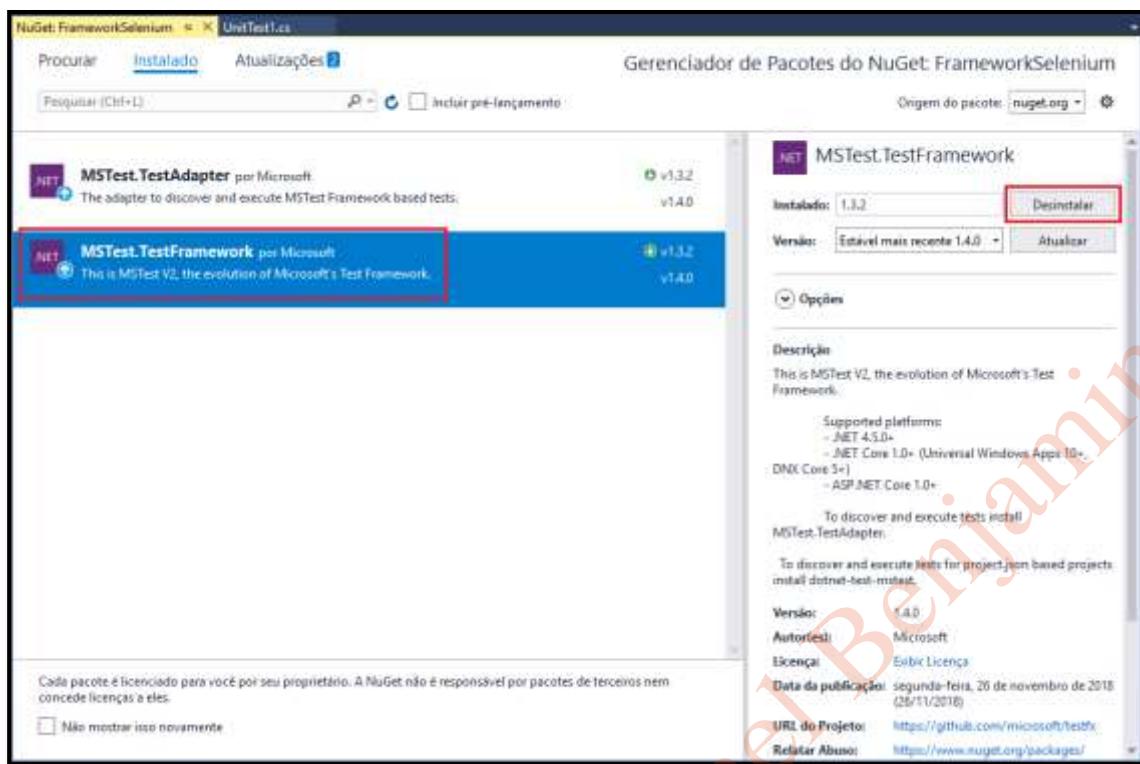
Os demais conceitos de testes que serão introduzidos neste curso poderão ser reaproveitados mesmo caso outros frameworks e linguagens sejam utilizados.

Para realizar a alteração devemos acessar o projeto dentro do gerenciador de soluções, clicar com o botão direito acima dele e selecionar a opção '**Gerenciar Pacotes do NuGet...**'

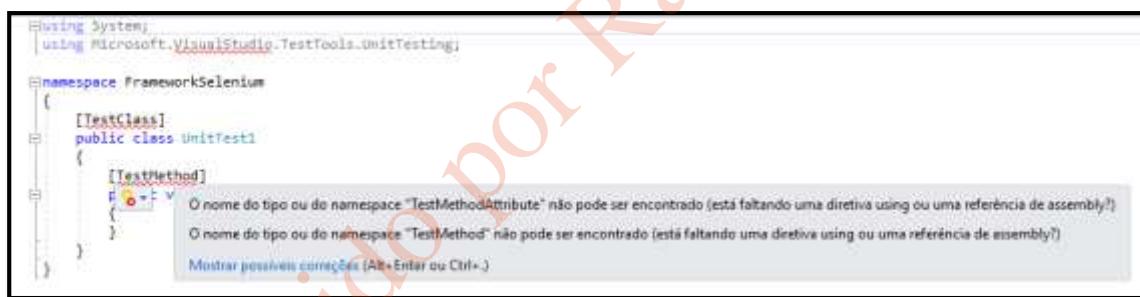


Dentro do gerenciador de pacotes, selecionar o item '**MSTest.Framework**', e clicar em **Desinstalar**.

Não remover o '**MSTest.TestAdapter**', é usado para a identificação dos testes.

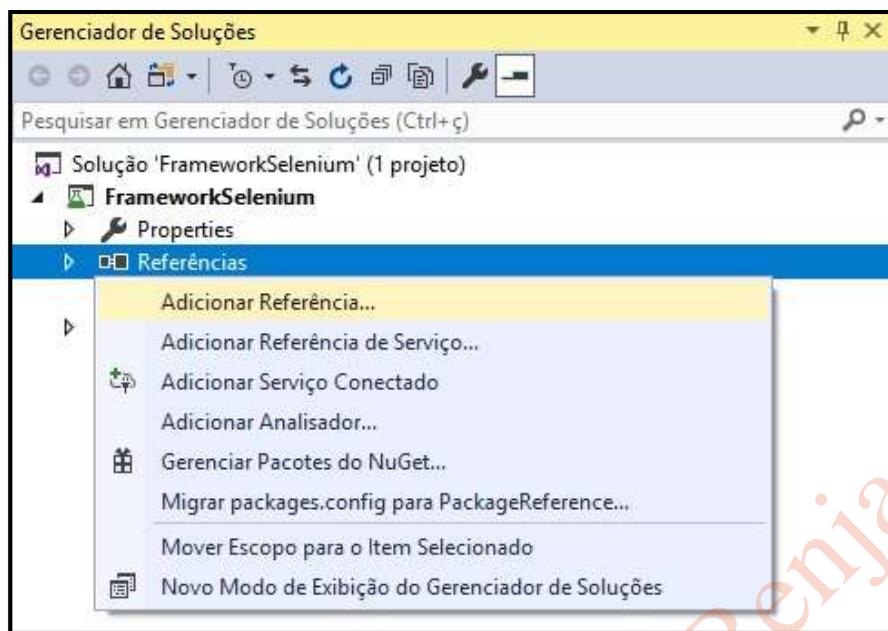


De volta à tela de código, serão exibidos erros indicando que o pacote de testes foi removido.

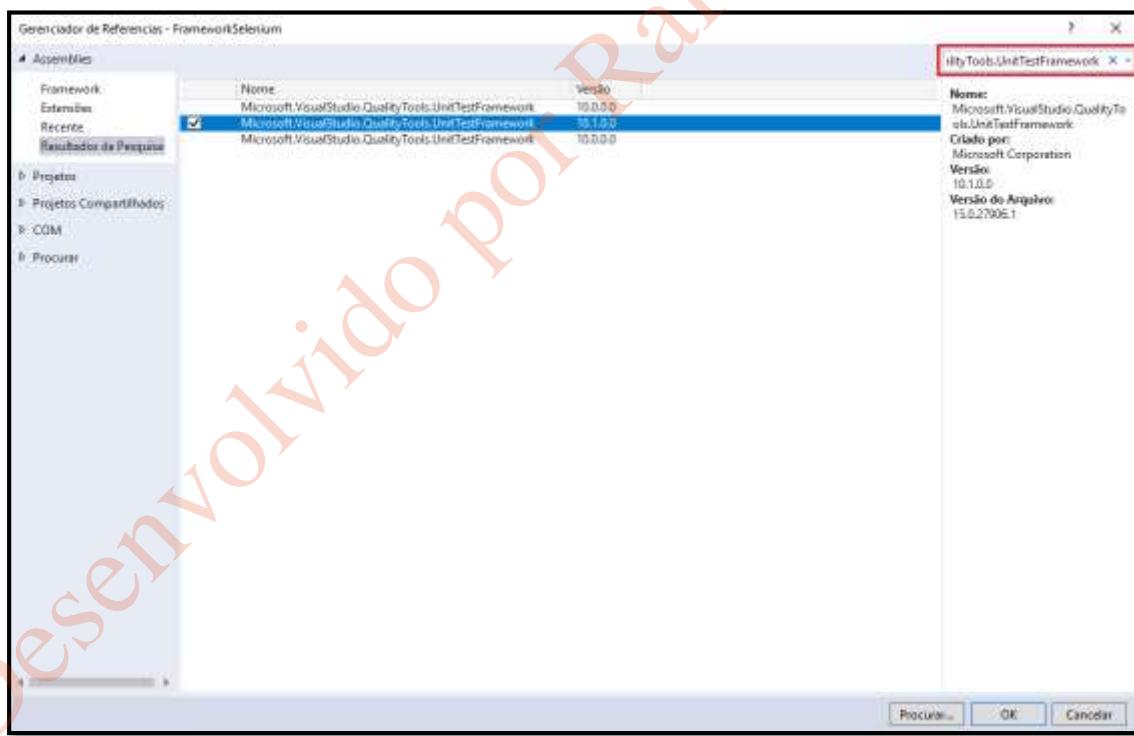


Para corrigi-los iremos incluir a referência à versão 1 do MsTest.

Para isso, iremos clicar com o botão direito em '**Referências**' dentro do nosso projeto e selecionar '**Adicionar Referência**'.



Dentro do Gerenciador de Referências, no canto superior direito busque por :
'Microsoft.VisualStudio.QualityTools.UnitTestingFramework'. Serão exibidas três opções, selecione apenas uma qualquer dentre as três e clique em OK.



Os erros deixarão de ser exibidos no projeto.

4.3 Primeiro Teste

Entendendo a estrutura criada pelo projeto:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace FrameworkQarti
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

using Microsoft.VisualStudio.TestTools.UnitTesting: As palavras chave using no início do arquivo são referências as bibliotecas importadas. Neste caso à biblioteca de testes MsTest.

namespace FrameworkQarti: É o nome dado ao agrupamento de classes, inicialmente usaremos o namespace padrão que é o nome do projeto criado. Futuramente com o crescimento do framework iremos agrupar as classes de acordo com suas características.

[TestClass]: Atributo que identifica a classe associada como uma classe de teste.

[TestMethod]: Atributo que identifica o método associado como um método de teste.

A documentação com os atributos disponíveis para o MsTest encontra-se em : <https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting?redirectedfrom=MSDN&view=mstest-net-1.2.0>

Antes de iniciar o primeiro teste, precisamos realizar o download das dependências necessárias.

Acesse o Gerenciador de Pacotes do NuGet e busque pelos seguintes pacotes:

- **Selenium.WebDriver**
- **Selenium.Chrome.WebDriver**

Estes contém as bibliotecas do Selenium e o executável para o driver do chrome respectivamente.

Renomeie o arquivo de ‘**UnitTest1**’ para ‘**PrimeiroTeste**’.

Com isso podemos escrever nosso primeiro teste:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;

namespace FrameworkQarti
{
    [TestClass]
    public class PrimeiroTeste
    {
        [TestMethod]
        public void TestMethod1()
        {
            IWebDriver driver = new ChromeDriver();
```

```

        driver.Navigate().GoToUrl("https://www.google.com.br");

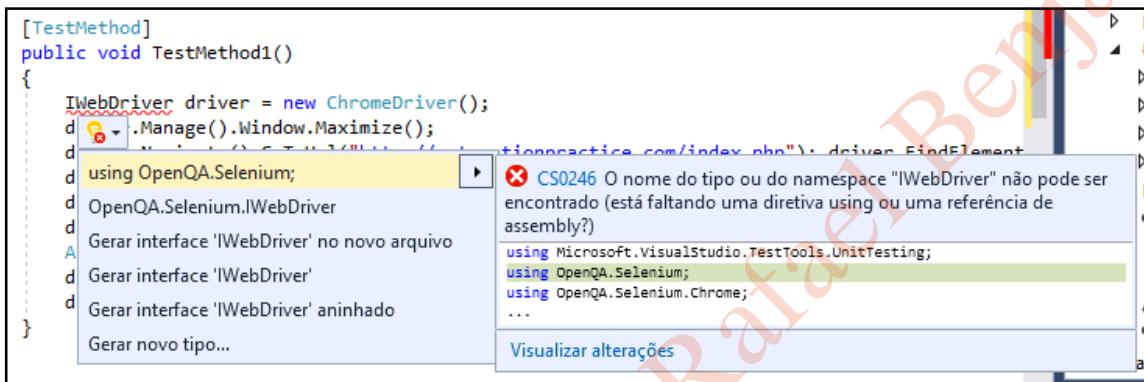
        string titulo = driver.Title;

        Assert.IsTrue(titulo == "Google", "Teste falhou, resultado exibido: " + titulo);

        driver.Quit();
    }
}
}

```

Se após a inclusão do código algum comando fique com um erro marcado, mantenha o cursor sobre o erro e clique em ‘Mostrar possíveis correções’, então serão exibidas as sugestões indicando um possível pacote que não foi importado.



Entendendo o código:

`IWebDriver driver = new ChromeDriver();` : Criamos uma nova instância do driver para o browser Chrome. Após este comando ser executado o navegador será aberto. Todos os comandos que serão efetuados devem ser feitos a partir desta instância do driver.

`driver.Navigate().GoToUrl("https://www.google.com.br")`: Navega para a url apontada.

`string titulo = driver.Title;`: Armazena o Título da aba atual dentro da variável ‘titulo’

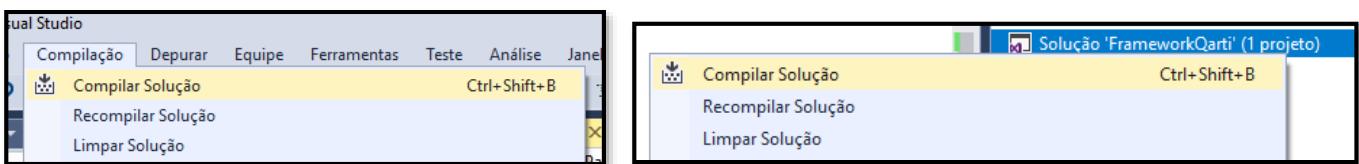
`Assert.IsTrue(titulo == "Google", "Teste falhou, resultado exibido: " + titulo);`): O Comando ‘Assert’ é usado para realizar as verificações (Aserções). Neste caso, verifica se a condição que compara o título armazenado anteriormente, é igual a “Google”. Caso a validação falhe é exibida a mensagem configurada.

`driver.Quit();`: Fecha todas as janelas abertas nesta instância do driver.

4.4 Executando o teste

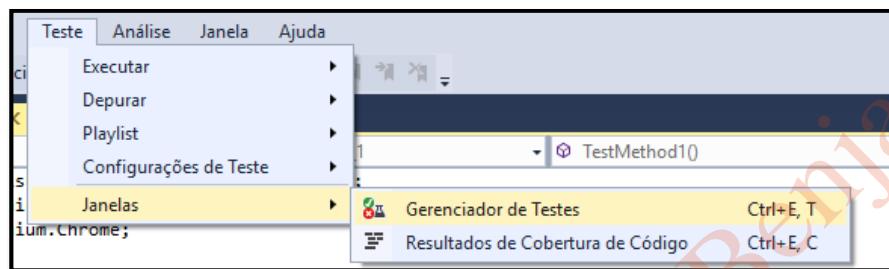
Para executar primeiro teste, devemos primeiro compilar a solução.

Clicando no menu ‘Compilação’>‘Compilar Solução’ ou clicando no projeto com o botão direito e selecionar ‘Compilar’.

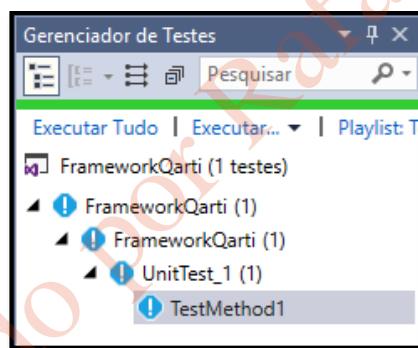


O teste deve ser exibido no painel ‘Gerenciador de Testes’.

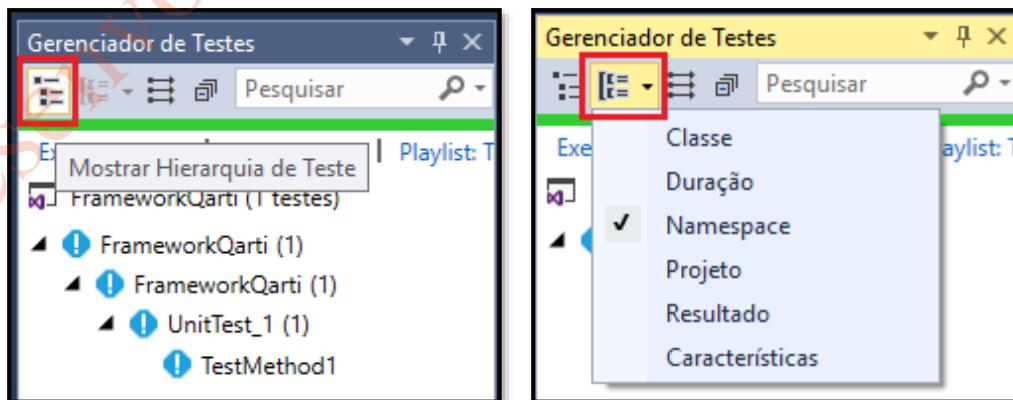
Caso não o painel ‘Gerenciador de Testes’ não esteja sendo exibido, acesse o menu ‘Teste’> ‘Janelas’ > ‘Gerenciador de Testes’.



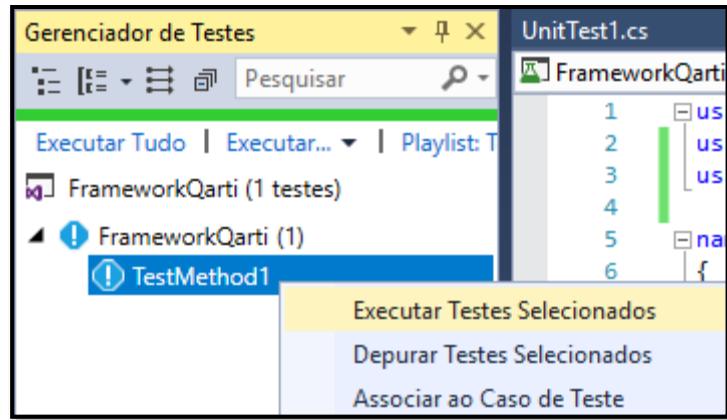
Caso os testes não estejam sendo exibidos, verifique se a visibilidade da classe e métodos de teste estão definidos como Public, e se possuem os atributos de teste definidos acima dos mesmos.



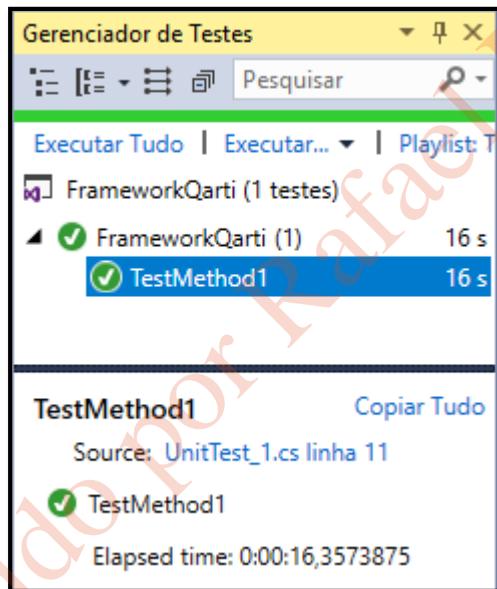
Também é possível alterar a exibição dos testes através dos botões ‘Mostrar Hierarquia de Teste’ e ‘Agrupar por...’ conforme ilustrado abaixo.



Para executar os testes, selecione os testes no ‘Gerenciador de Testes’, clique com o botão direito e selecione ‘Executar Testes Selecionados’.



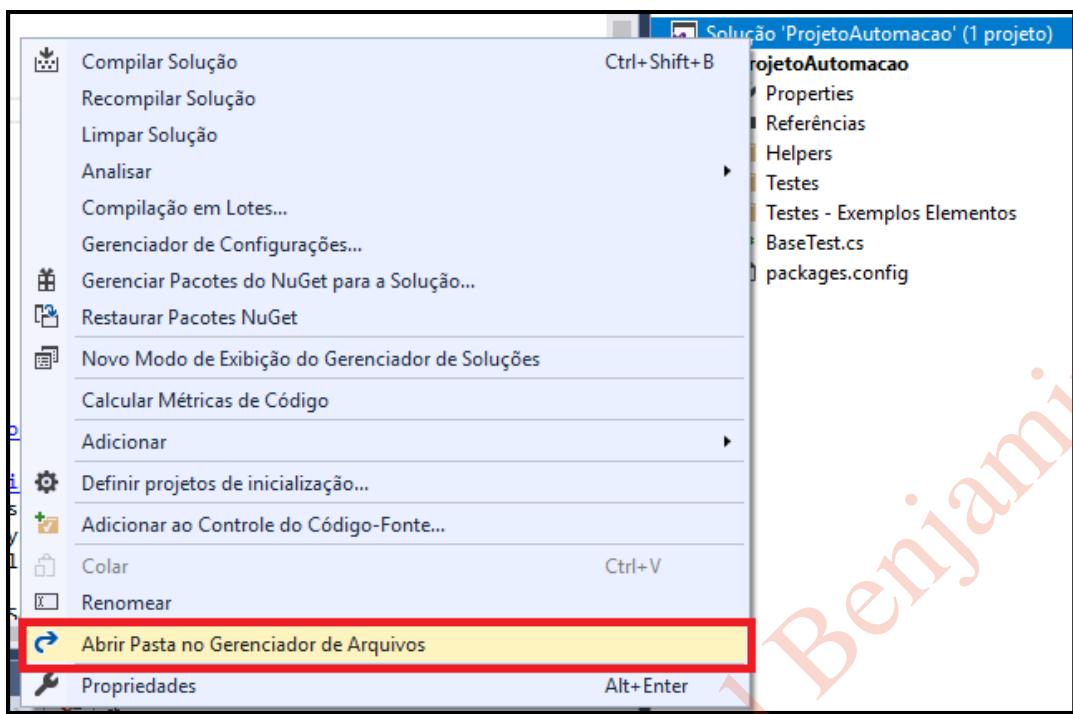
Após a execução o resultado será exibido no 'Gerenciador de Testes'



4.5 Estrutura do projeto em C#

Para demonstrar como funciona a estrutura do projeto, usarei um exemplo de um projeto que já está com uma estrutura um pouco maior.

Clicar com o botão direito na Solução e clicar em Abrir Pasta no Gerenciador de Arquivos



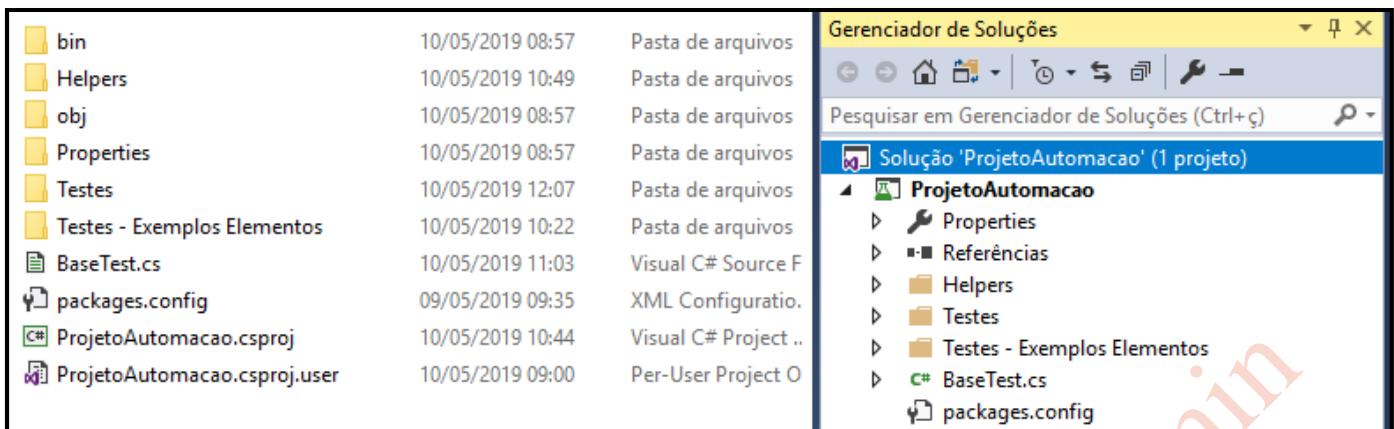
A pasta raiz da Solução será aberta pelo Windows:

Nome	Data de modificaç...	Tipo	Tamanho
.vs	12/05/2019 19:40	Pasta de arquivos	
packages	12/05/2019 19:41	Pasta de arquivos	
ProjetoAutomacao	10/05/2019 11:03	Pasta de arquivos	
TestResults	10/05/2019 12:09	Pasta de arquivos	
ProjetoAutomacao.sln	07/05/2019 14:11	Visual Studio Solu...	2 KB

Entendendo este nível da solução:

- **.vs**: Pasta oculta que armazena as informações sobre o estado atual do projeto, como breakpoints, quais arquivos estão abertos, etc...
- **Packages**: Armazena todos os pacotes referenciados em todos os projetos dentro da sua solução
- **ProjetoAutomação**: Pasta do projeto atual, lembrando que o nome varia de acordo com seu projeto. E uma solução pode ter vários projetos.
- **TestResults**: Pasta onde são gerados os resultados após cada execução de teste
- **ProjetoAutomacao.sln**: Arquivo solução, é por ele que você deve abrir o seu projeto.

Agora abra a pasta de seu projeto pelo Windows:



Acima temos a pasta do projeto aberta à esquerda, e a estrutura do projeto aberta no visual Studio à direita.

Entendendo este nível de projeto:

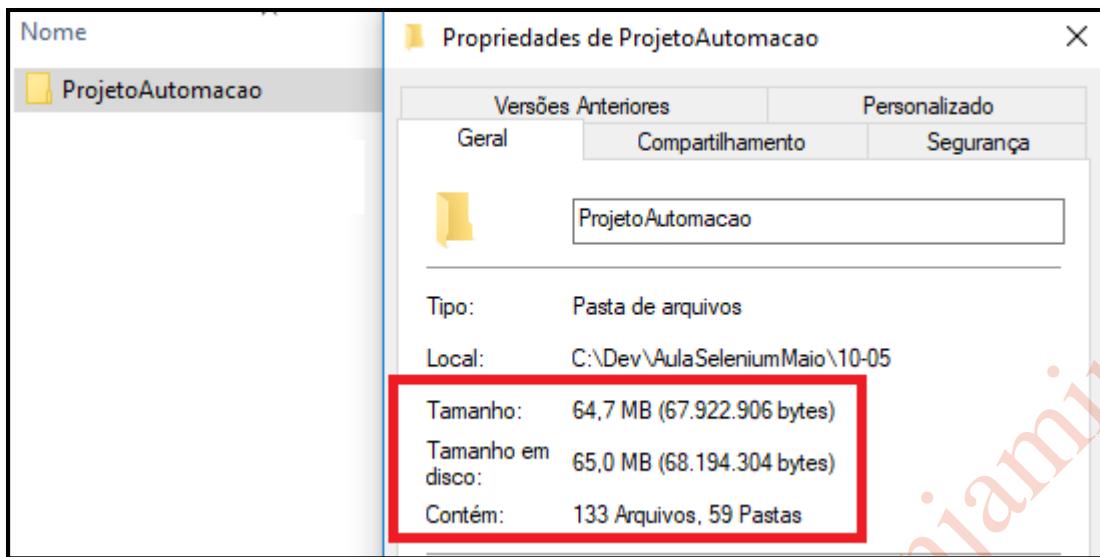
- **Bin:** Pasta onde são gerados os arquivos finais após a compilação junto com as bibliotecas e arquivos necessários para execução do projeto
- **Helpers:** Pasta criada neste projeto de exemplo, pode vista dentro do projeto pelo visual studio
- **Obj:** Pasta onde estão os arquivos fonte compilados individualmente
- **Properties:** Armazena os arquivos com as informações como autor, versão, descrição, etc...
- **Testes:** Pasta criada neste projeto de exemplo, pode vista dentro do projeto pelo visual studio
- **Testes – Exemplos Elementos:** Pasta criada neste projeto de exemplo, pode vista dentro do projeto pelo visual studio
- **BaseTest.cs:** Arquivo criado neste projeto de exemplo, pode vista dentro do projeto pelo visual studio
- **Packages.config:** Arquivo que lista quais pacotes estão sendo usados no projeto e suas versões
- **ProjetoAutomacao.csproj:** Arquivo com as informações do projeto (seu nome é correspondente ao projeto criado), o projeto pode ser aberto por ele. *Mas é recomendado abrir toda a solução e não apenas o projeto.*
- **ProjetoAutomacao.csproj.user:** Arquivo que armazena suas preferências pela IDE em relação a este projeto

4.5.1 Exportando a solução

Não é necessário nenhum comando através do visual Studio para exportar a solução, basta enviar sua pasta. Porém sua pasta possui vários arquivos gerados após a compilação e o que irei mostrar é a melhor forma de enviar um projeto para alguém ou mesmo enviar para uma ferramenta de controle de versão.

Ao voltar uma pasta acima da solução e verificar o tamanho do projeto, neste exemplo tenho 65Mb a ser enviado. No momento ainda é um projeto pequeno, mesmo assim podemos reduzir muito o seu tamanho removendo arquivos que são gerados apenas após a compilação.

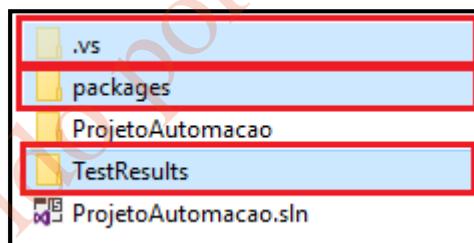
Estes arquivos não precisam ser verificados, já que as informações necessárias para gerá-los estão mantidas nos arquivos fonte do projeto.



Podemos apagar as seguintes pastas:

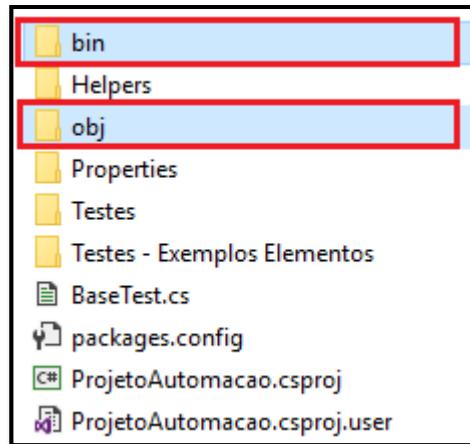
Em nível de solução:

- VS:** Quando você carregar o seu visual Studio novamente ele gera um novo (Lembre-se de ter fechado o projeto ao apagar esta pasta)
- Packages:** Os pacotes podem ser baixados novamente pelo gerenciador de pacotes do nuget. Também é a pasta mais pesada já que possui todas as dependências de todos os projetos dentro de sua solução.
- TestResults:** Inicialmente os resultados são apenas pastas vazias, não há necessidade de serem salvos

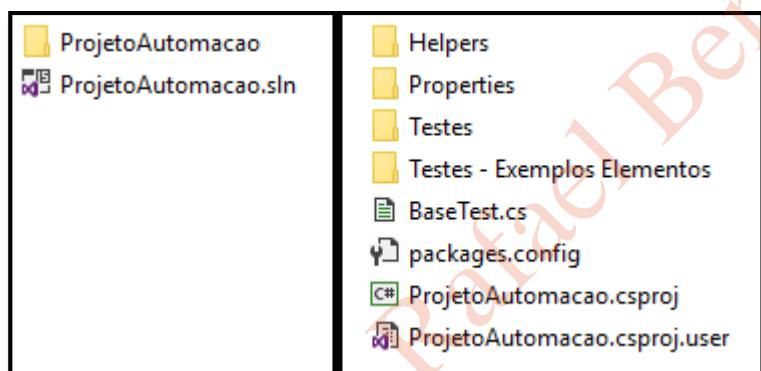


Em nível de projeto:

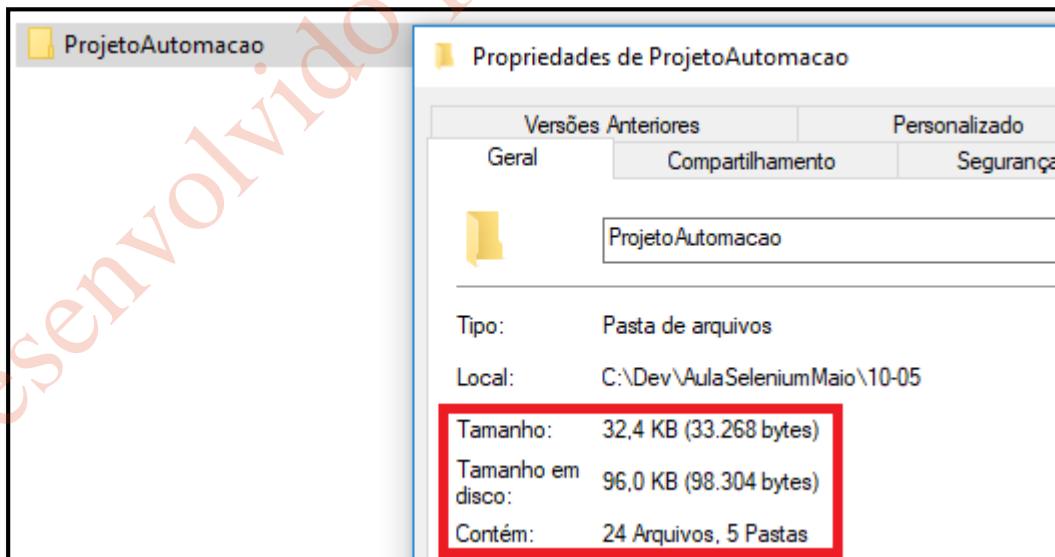
- Bin:** Após a compilação, os arquivos finais junto com as dependências que foram copiadas da pasta Packages são armazenados aqui para serem executados.
- Obj:** Após a compilação os arquivos parciais são gerados aqui.



Solução (esquerda) e projeto (direita) após exclusão:

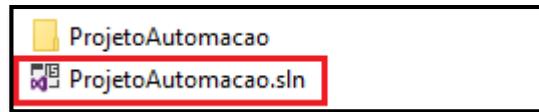


Após excluir estas pastas, o projeto é reduzido drasticamente para 96kb. Apenas os arquivos são mantidos, este serão compilados e suas dependências podem ser baixadas novamente.



4.5.2 Importando a solução

Novamente, não é preciso nenhum caminho especial pelo visual Studio para abrir uma solução, basta abrir o arquivo **.sln** dentro da pasta e todos os projetos serão abertos pelo visual studio.

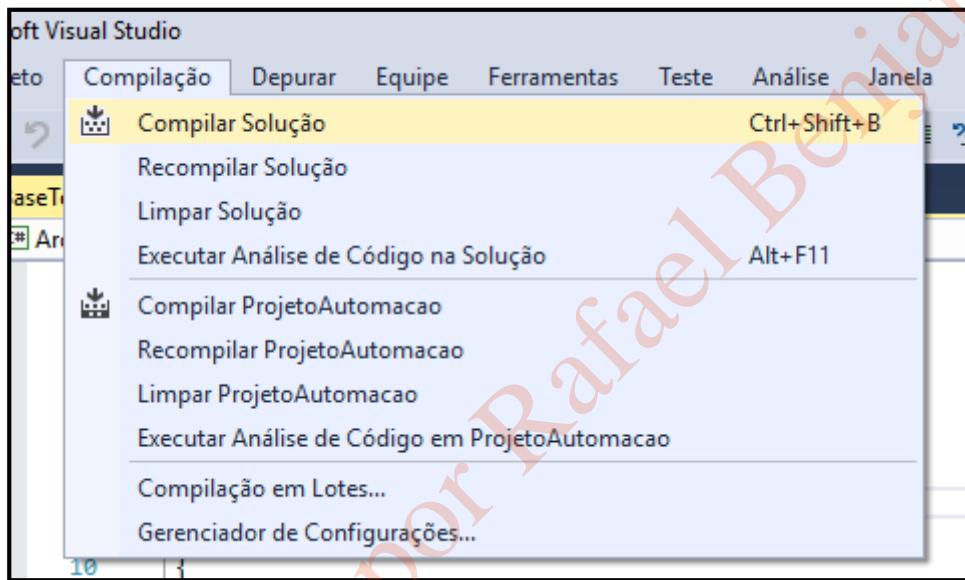


Seguindo o exemplo, vamos abrir no projeto que foi ‘compactado’.

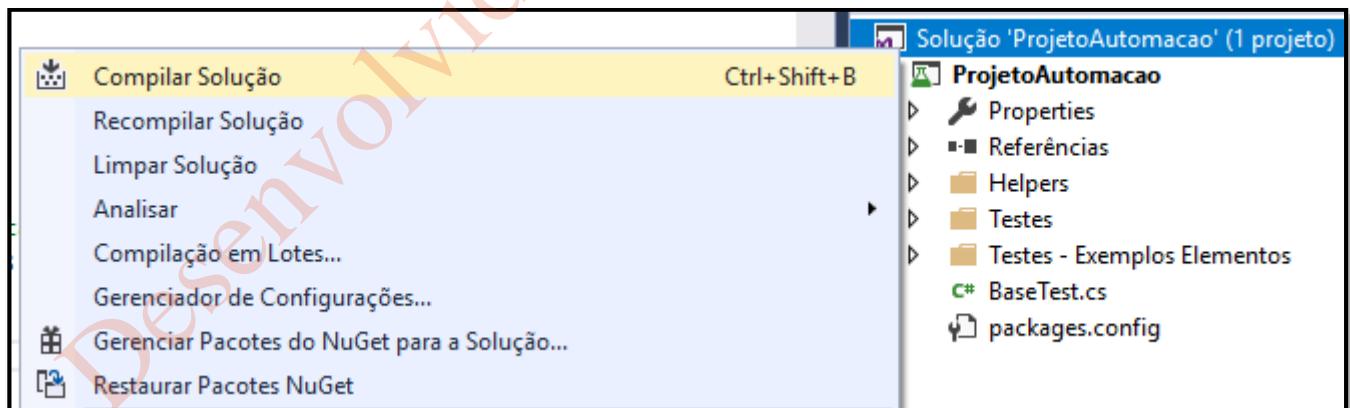
Logo após abrir o projeto repare que os testes não serão exibidos, pois o projeto ainda não foi compilado. Neste momento a pasta .vs é gerada novamente, junto com as pasta bin e obj, porém estas ainda ficarão vazias.

Para compilar:

Selecione no menu Compilação > Compilar Solução:



Ou botão direito acima da Solução e selecione Compilar Solução:

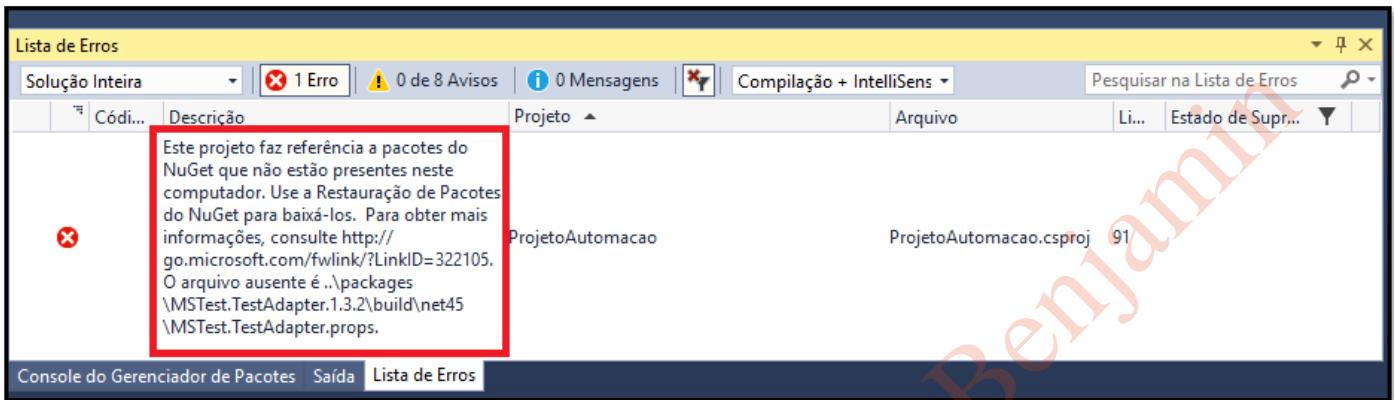


Durante este processo, os pacotes ausentes referenciados no arquivo *packages.config* serão baixados na pasta packages da solução, então após isto nosso projeto será compilado, as pasta Bin e obj serão populadas novamente e os testes devem ser exibidos novamente.

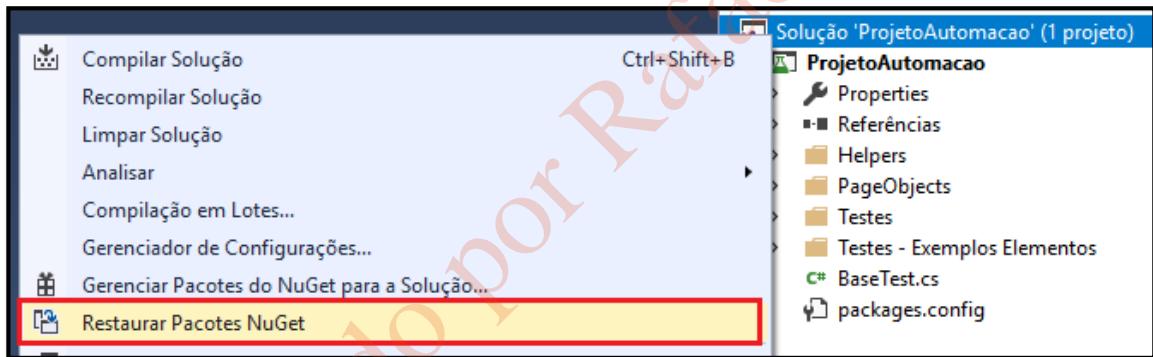
Agora nosso projeto volta a ficar maior e funcionando normalmente.

Obs: Caso seja exibida uma mensagem de erro ‘Este projeto faz referência a pacotes do NuGet que não estão presentes neste computador. Use a Restauração de Pacotes do NuGet para baixá-los.’, clique com o botão direito na solução e depois clique em ‘Restaurar Pacotes NuGet’ para primeiro baixar os pacotes novamente. Após isto compile sua solução.

Mensagem de erro:

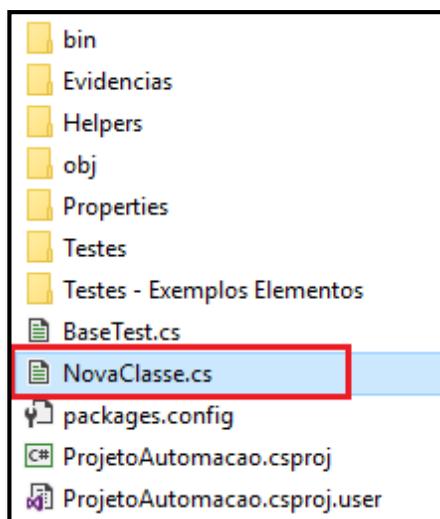


Restaurar Pacotes NuGet:

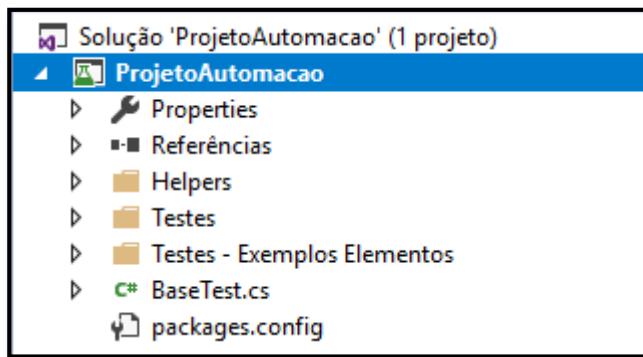


4.5.3 Copiando arquivos para a solução

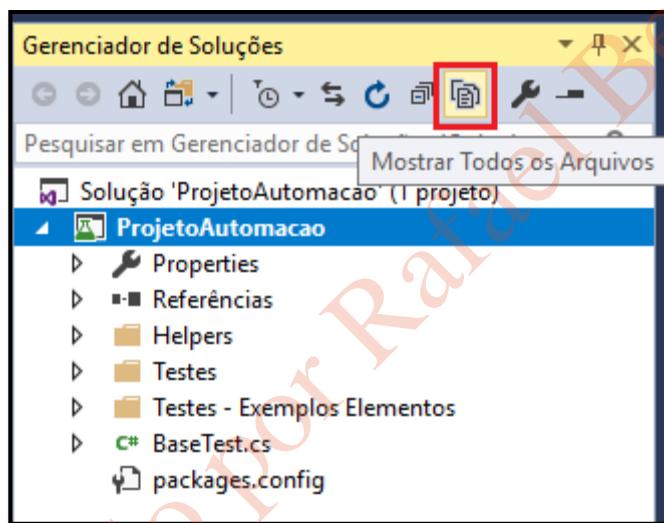
Caso queria incluir apenas alguns arquivos novos dentro de seu projeto e solução, é necessário move-los para a pasta que deseja pelo Windows. Como exemplo vou incluir o arquivo ‘NovaClasse.cs’ dentro do meu projeto:



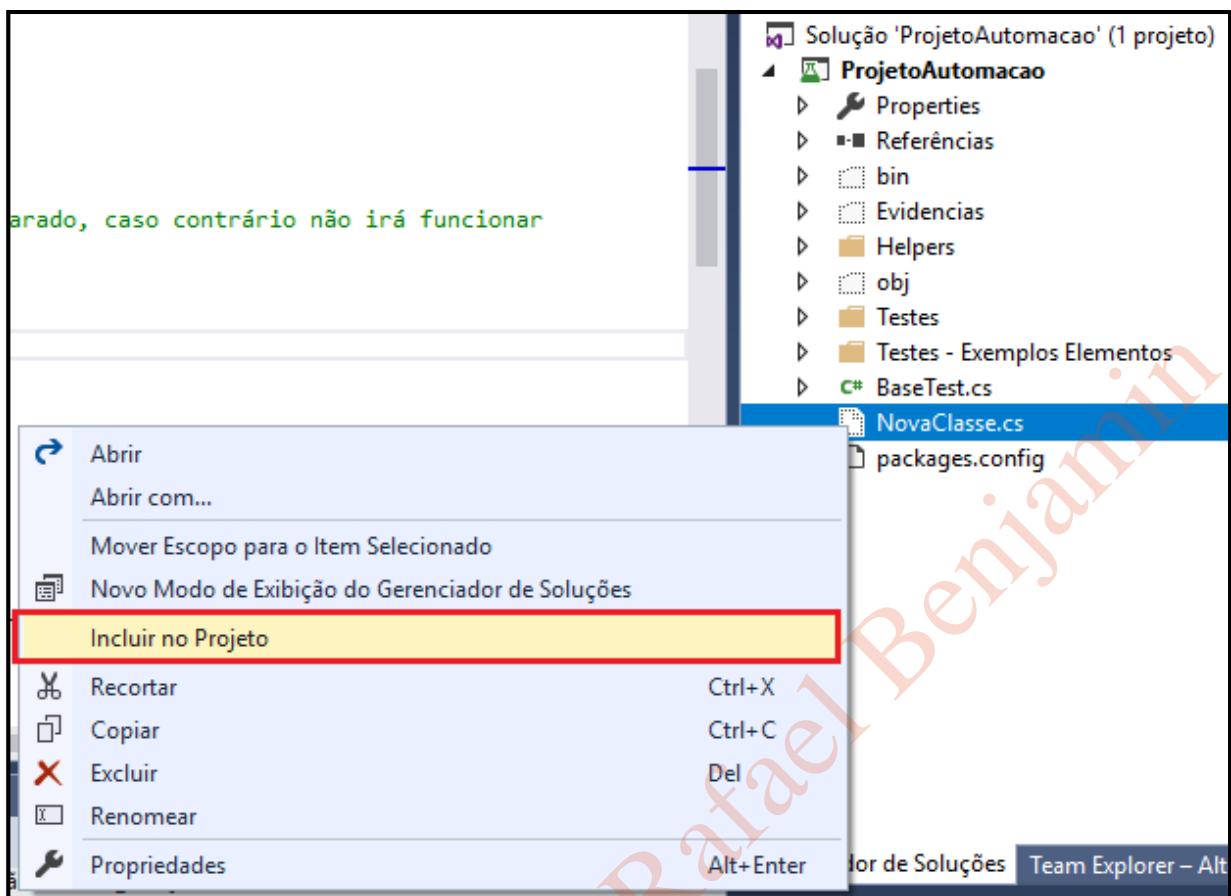
Porém essa alteração não é refletida dentro da solução pelo visual Studio:



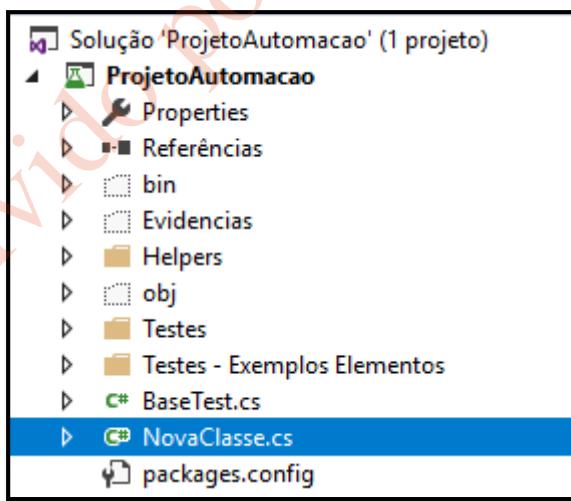
Para que o novo arquivo seja encontrado pela solução, temos que clicar no ícone ‘**Mostar Todos os arquivos**’ que se encontra na barra do Gerenciador de Soluções:



Após clicar, as pastas que e arquivos que não estão incluídas serão exibidas com um ícone pontilhado. Clique no(s) arquivo(s) que deseja incluir com o botão direito e selecione **Incluir no Projeto**.



Agora o arquivo é adicionado ao projeto:



Após isto, clique novamente no ícone ‘Mostrar Todos os Arquivos’ para ocultar os demais arquivos e evitar que algum arquivo desnecessário seja incluído ou até mesmo excluído no Windows.

Obs: Também tome cuidado com o Namespace, que varia de acordo com o nome do projeto do qual este arquivo pertence. Sempre que copiar algum código ou arquivo externo lembre-se de verificar se o namespace está de acordo com o seu projeto.

NovaClasse.cs

```

1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4  namespace ProjetoAutomacao
5  {
6      [TestClass]
7      public class NovaClasse
8      {
9      }
10 }
11
12

```

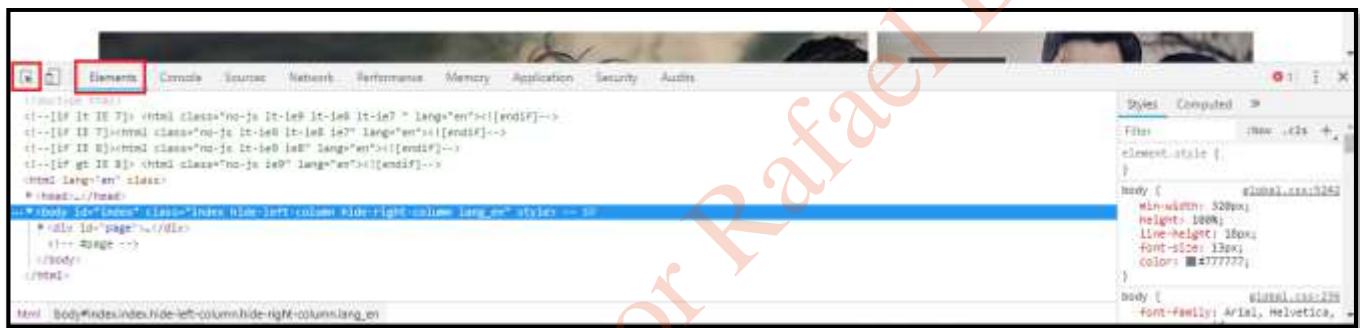
Gerenciador de Soluções

- Solução 'ProjetoAutomacao' (1 projeto)
 - ProjetoAutomacao
 - Properties
 - Referências
 - Helpers
 - Testes
 - Testes - Exemplos Elementos
 - BaseTest.cs
 - NovaClasse.cs
 - packages.config

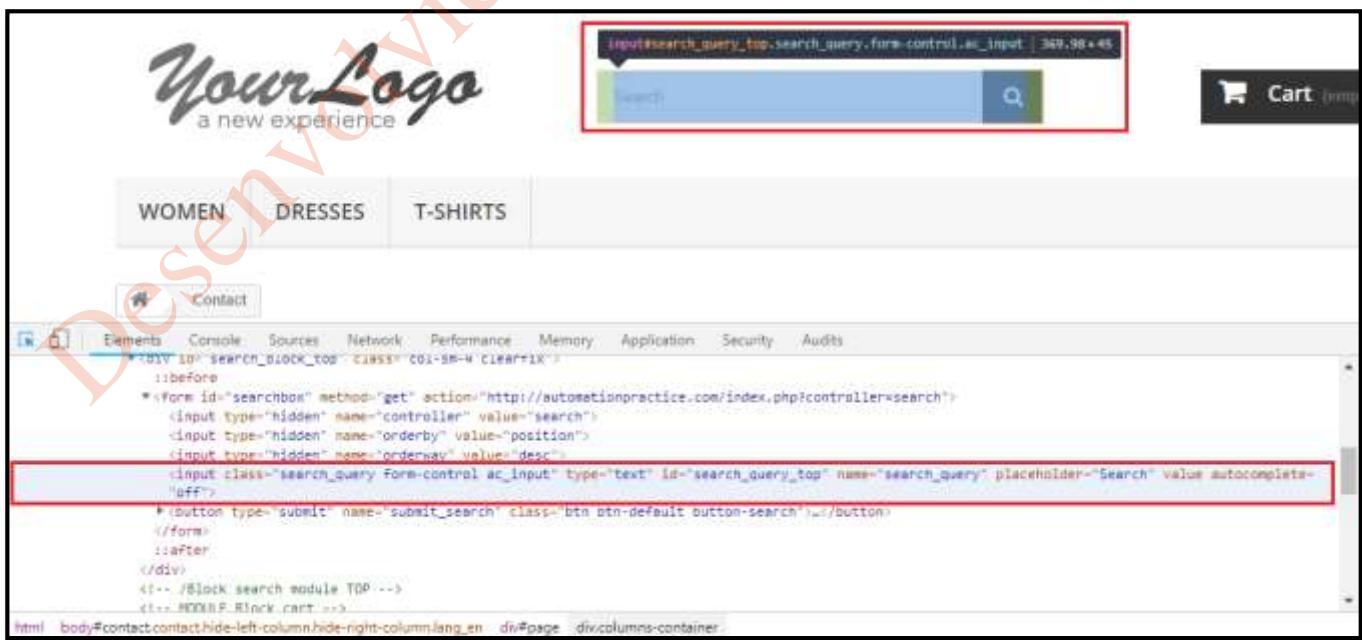
4.6 Inspecionando Elementos

Para trabalhar com automação funcional é necessário mapear os elementos da tela com os quais iremos interagir, faremos isso através das ferramentas de inspeção dos navegadores.

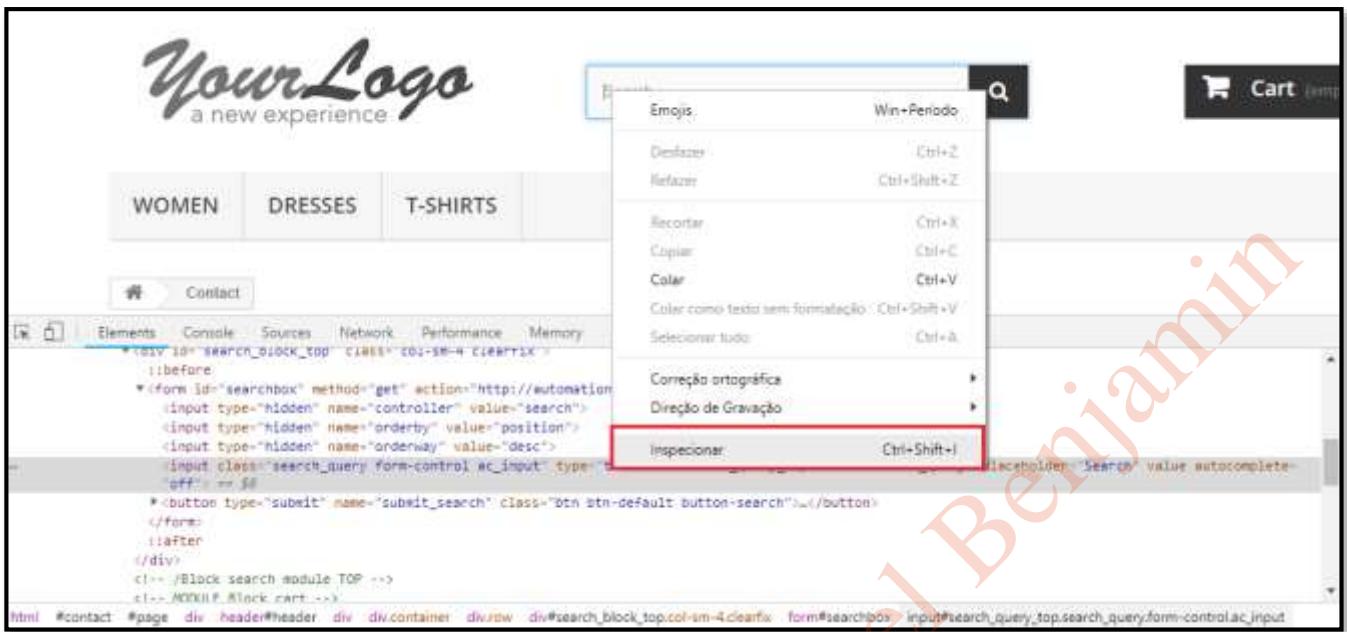
Abra o Chrome e aperte o botão F12. Será exibido uma console com as ferramentas de desenvolvedor. Usaremos a guia Elements para realizar a inspeção dos elementos.



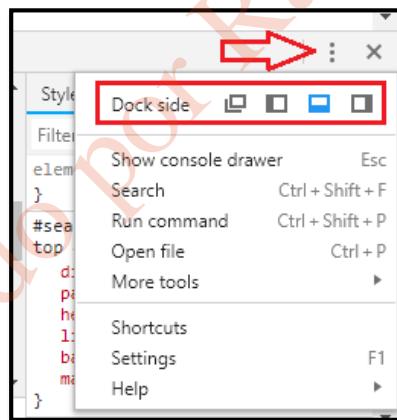
Clique na seta no canto superior esquerdo das ferramentas do desenvolvedor, agora é possível selecionar os elementos e ver suas propriedades. Clique em um elemento com essa opção, ele será apontado dentro do HTML exibido na guia Elements.



Também é possível realizar a inspeção posicionando o mouse em cima de algum elemento, clicar com o botão direito > Inspecionar.



Dependendo da resolução do navegador é necessário ajustar a posição de exibição desta ferramenta, esta pode ser configurada pelo menu “...” e selecionar a posição conforme figura abaixo:



O uso em demais navegadores (IE, Firefox, Safari) é bem semelhante.

4.6.1 Tipos de Identificadores

O comando utilizado pelo selenium para localizar os elementos é o **FindElement** para um Elemento e o **FindElements** para uma lista de elementos.

FindElement():

- Não encontrou elementos: lança a exceção NoSuchElementException
- Encontrou uma correspondência: retorna o WebElement encontrado
- Uma ou mais correspondências: retorna o primeiro WebElement encontrado no DOM

FindElements():

- Não encontrou elementos: retorna uma lista vazia

- Encontrou uma correspondência: retorna uma lista com apenas um WebElement
- Uma ou mais correspondências: retorna uma lista com todas as correspondências de WebElement

Estes comandos necessitam ainda de um identificador do tipo **By** que irei detalhar abaixo:

ID

A forma mais eficiente de localizar um elemento, já que pelas boas práticas de desenvolvimento web um ID deve ser único na página.

Exemplo:

```
<input class="search_query form-control ac_input" type="text" id="search_query_top" name="search_query" placeholder="Search" value autocomplete="off"> == $0
```

```
driver.FindElement(By.Id("search_query_top"))
```

Name

Localizado pela propriedade 'name', porém vários elementos podem possuir este atributo.

```
<input class="search_query form-control ac_input" type="text" id="search_query_top" name="search_query" placeholder="Search" value autocomplete="off"> == $0
```

```
driver.FindElement(By.Name("search_query"))
```

CssSelector

Localizado pela propriedade 'class', porém vários elementos podem possuir este atributo. Este é um atributo que indica qual o estilo adotado pelo elemento. Neste caso temos três estilos atribuídos para este elemento.

-search_query

-form-control

-ac_input

Veja que eles possuem um espaço entre si, o espaço é o separador entre os diferentes estilos.

```
<input class="search_query form-control ac_input" type="text" id="search_query_top" name="search_query" placeholder="Search" value autocomplete="off"> == $0
```

Porém na hora de mapear devemos utilizar um “.” antes do início de cada estilo, e remover os espaços entre eles.

```
driver.FindElement(By.CssSelector(".search_query.form-control.ac_input"))
```

ClassName

Semelhante ao CssSelector localizado pela propriedade 'class'. A diferença é que neste não podemos usar mais de uma propriedade.

```
<input class="search_query form-control ac_input" type="text" id="search_query_top" name="search_query" placeholder="Search" value autocomplete="off"> == $0
```

Na hora de mapear não é necessário utilizar o “.” antes de seu identificador, porém devemos escolher apenas um.

```
driver.FindElement(By.ClassName("ac_input"))
```

TagName

Utiliza a tag para identificação. É um identificador com menor utilização já que muitas tags são repetidas.

```
<input class="search_query form-control ac_input" type="text" id="search_query_top" name="search_query" placeholder="Search" value autocomplete="off"> == $0
```

```
driver.FindElement(By.TagName("input"))
```

LinkText

Buscar por elementos tipo link (ou seja com a tag HTML “a”), que possuam o texto exato especificado.

```
<a href="http://automationpractice.com/index.php?controller=contact" title="Contact Us">Contact us</a>
```

```
driver.FindElement(By.LinkText("Contact us"))
```

PartialLinkText

Buscar por elementos tipo link (ou seja com a tag HTML “a”), que possuam parte do texto especificado.

```
<a href="http://automationpractice.com/index.php?controller=contact" title="Contact Us">Contact us</a>
```

```
driver.FindElement(By.PartialLinkText("Contact"))
```

Xpath

Busca utilizando uma expressão própria. Pode encontrar elementos que não possuem identificadores claros como Id, Css, Name . Pode encontrar um elemento utilizando um atributo qualquer, ou uma referência próxima.

Neste exemplo é feita a busca utilizando o atributo ‘href’.

```
<a href="http://automationpractice.com/index.php?controller=contact" title="Contact Us">Contact us</a>
```

```
driver.FindElement(By.Xpath("//a[@href='http://automationpractice.com/index.php?controller=contact']))
```

Lembrando que devemos utilizar um localizador que retorne apenas o elemento desejado. Antes de testar no código podemos verificar se o identificador é único utilizando Css ou Xpath.

4.6.2 Css por Jquery

Através do Chrome, podemos utilizar o seu console e com um comando jquery através do seu css verificar se o identificador que pretendemos usar é único.

Como exemplo tentaremos mapear o seguinte botão que se encontra nesta página :

<https://automacaocombatista.herokuapp.com/buscaelementos/botoes>



Caso tente encontrar o botão usando apenas o cssselector do tipo 'waves-light'.

Acessar o console do Chrome através do F12.

Digitar:

`$('.waves-light')`

E dar Enter. (Lembrando que estamos passando como um css, logo as classes são representadas com “.” Antes de seu nome”)

```
> $('.waves-light')
<-- ▶ ft.fn.init(3) [a#teste.waves-light.btn, a.btn-floating.btn-large.waves-light.red, button.btn.waves-light, p]
  ▶ 0: a#teste.waves-light.btn
  ▶ 1: a.btn-floating.btn-large.waves-light.red
  ▶ 2: button.btn.waves-light
  ▶ context: document
  ▶ length: 3
  ▶ prevObject: ft.fn.init [document, context: document]
  ▶ selector: ".waves-light"
  ▶ __proto__: Object(0)
```

Ao expandir a linha resultante, veremos que são exibidos 3 itens. Então temos 3 botões que utilizam este seletor, assim não sendo a escolha correta. Ao passar o mouse sobre estes itens da lista, repare que são destacados na página indicando o que eles são.

Vamos testar com outro estilo, agora com o :

```
$('.btn-floating')
```

```
> $('.btn-floating')
<-- ▶ ft.fn.init [a.btn-floating.btn-large.waves-light.red, prevObject:
  ▶ 0: a.btn-floating.btn-large.waves-light.red
  ▶ context: document
  ▶ length: 1
  ▶ prevObject: ft.fn.init [document, context: document]
  ▶ selector: ".btn-floating"
  ▶ __proto__: Object(0)
```

Agora é retornado apenas um elemento, justamente o que precisamos.

É uma questão de tentativa e erro até encontrar um identificador único para nossos elementos.

4.6.3 Buscando por xpath

O xpath também pode ser usado para verificar se o elemento em questão é único ou quando temos dificuldade em mapeá-lo. A maneira mais comum de mapear um elemento difícil é da seguinte forma:

Inspecionar o elemento > Clicar em código HTML com o botão direito > Selecionar ‘Copy’ > Selecionar ‘Copy Xpath’

The screenshot shows the Chrome DevTools Elements tab with the DOM tree. A context menu is open over a red button labeled "Floating". The "Copy" submenu is highlighted with a red box, and the "Copy XPath" option is also highlighted with a red box. The full path to the button is visible in the status bar at the bottom.

Porém nem sempre será um resultado 'enxuto'. Neste caso o retorno foi:

`/html/body/div[2]/div[2]/div[4]/div[1]/div/a`

Esta é a representação de todo o 'caminho' dentro do HTML até chegar neste elemento.

Para testar, aperte F12 no Chrome, e na guia '*Elements*', aperte CTRL+F

Dentro desta barra que foi exibida no final, digite o xpath retornado acima.

The screenshot shows the Chrome DevTools Elements tab with the DOM tree. A yellow box highlights the element `<a>` tag. The status bar at the bottom shows the full xpath: `/html/body/div[2]/div[2]/div[4]/div[1]/div/a`. A red box highlights the status bar.

Funciona, temos o elemento destacado em amarelo e a indicação de que existe apenas 1 na direita.

O principal problema em manter a busca desta forma, é que como ele percorre este caminho específico, qualquer alteração fará com que este código pare de funcionar.

Uma maneira de melhorar esta expressão é utilizar algum outro atributo para encontrar o elemento.

Imagine que o xpath funciona como uma query, primeiro temos que usar o “//” para iniciar a busca pelo DOM.

Em seguida temos que digitar a tag do elemento que desejamos neste caso a tag “a”.

Substitua esse comando no lugar do xpath gerado anteriormente, veja que temos como resultado todos os links com tag “a” na página.

Agora iremos refinar para encontrar o botão.

Vamos usar o atributo `onclick="ativaedesativa2()"`, então nosso xpath fica da seguinte forma (troque as aspas duplas por aspas simples):

```
//a[@onclick='ativaedesativa2()']
```

Cole este xpath novamente na busca e veja que o elemento correto é retornado. Esta forma é preferível já que temos algum identificador para representar o elemento, removendo o risco de quebra da automação mesmo caso o HTML anterrior ao elemento mude. Neste caso o elemento só deixará de ser encontrado caso a tag onclick ou o seu tipo mudem.

A estrutura é a seguinte:

```
//tagHTML[@Atributo='Valor_do_atributo']
```

Este foi apenas um exemplo mais simples para demonstração do funcionamento. Abaixo vou deixar um link com uma ‘colinha’ que serve tanto para Css quanto para Xpath para mais tipos de filtros.

Cheatsheet Xpath:

<https://devhints.io/xpath>

Links com referências sobre Xpath e Css:

Seletores css:

https://www.w3schools.com/cssref/css_selectors.asp

Guia Xpath:

<https://medium.com/testbean/xpath-na-pr%C3%A1tica-em-5-minutos-957b76beba5>

Doc oficial selenium:

https://www.seleniumhq.org/docs/03_webdriver.jsp#selenium-webdriver-api-commands-and-operations

Referência:

<https://www.toolsqa.com/selenium-webdriver/c-sharp/findelement-and findelements-commands-in-c/>

4.7 Tipos de Elementos

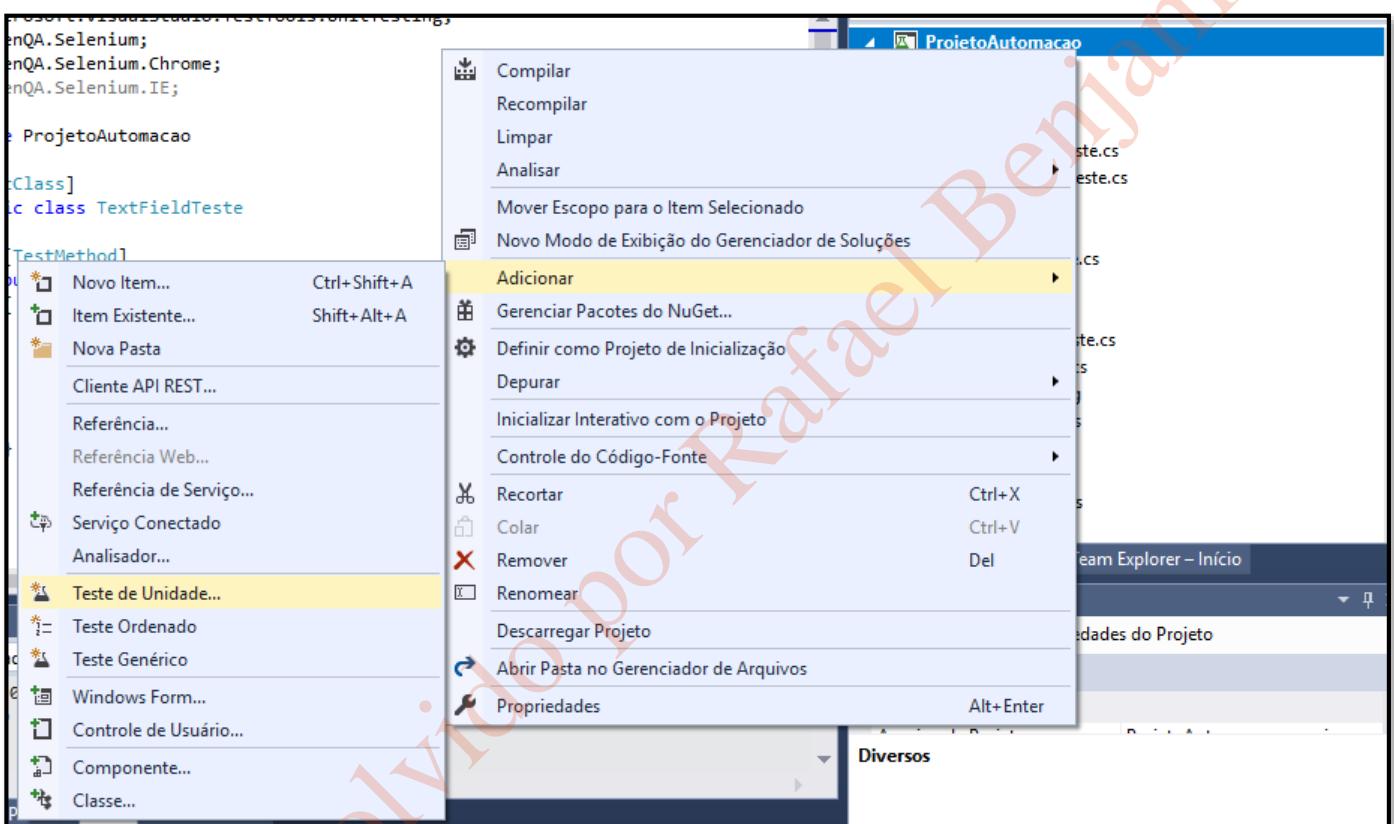
Agora para entender os principais tipos de elementos e como utilizar os identificadores vamos fazer um exemplo de cada tipo. A cada novo tipo sugiro que crie uma nova classe para organizar melhor os exemplos.

Para alguns testes precisaremos do Firefox, então baixe o seguinte pacote pelo **Gerenciador de Pacotes do NuGet**:

- **Selenium.Firefox.WebDriver**

4.7.1 Criando uma nova classe de teste

Para criar uma nova classe que já possua os atributos de teste, clique com o botão direito em seu Projeto, e selecione **Adicionar > Teste de Unidade**.



4.7.2 TextField

Crie uma nova classe chamada **TextFieldTest**:

```
[TestClass]
public class TextFieldTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

        driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/buscaelementos/inputsettextfield");
        driver.FindElement(By.Id("first_name")).SendKeys("Teste");
    }
}
```

```
}
```

4.7.3 Botões

Crie uma nova classe chamada BotoesTest:

```
[TestClass]
public class BotoesTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/buscaelementos/botoes");

        //Selecionando Botão pelo ID
        driver.FindElement(By.Id("teste")).Click();

        //Selecionando botão pela Classe (Apenas um identificador)
        //driver.FindElement(By.ClassName("btn-floating")).Click();

        //Selecionando botão pelo Css (vários identificadores iniciados por "." e sem espaços
entre si)
        driver.FindElement(By.CssSelector(".waves-light.btn-floating")).Click();

    }
}
```

4.7.4 Link

Crie uma nova classe chamada LinkTest:

```
[TestClass]
public class LinkTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/buscaelementos/links");

        //driver.FindElement(By.LinkText("Ok 200 - Sucess")).Click();

        //driver.FindElement(By.PartialLinkText("Erro 400")).Click();

        //driver.FindElement(By.TagName("a")).Click();

        driver.FindElement(By.XPath("//a[@href='/buscaelementos/notfound']")).Click();

    }
}
```

4.7.5 Radio

Crie uma nova classe chamada RadioTest:

```
[TestClass]
public class RadioTest
{
```

```

[TestMethod]
public void TestMethod1()
{
    IWebDriver driver = new ChromeDriver();

    driver.Navigate().GoToUrl("https://www.facebook.com/");

    bool selecionadoFeminino;
    bool selecionadoMasculino;

    //Radiobutton Feminino
    driver.FindElement(By.Id("u_0_9")).Click();
    Console.WriteLine("Click no Radiobutton Feminino");

    selecionadoFeminino = driver.FindElement(By.Id("u_0_9")).Selected;
    selecionadoMasculino = driver.FindElement(By.Id("u_0_a")).Selected;
    Console.WriteLine("Feminino : " + selecionadoFeminino + " Masculino : " +
selecionadoMasculino);

    //Radiobutton Masculino
    driver.FindElement(By.Id("u_0_a")).Click();
    Console.WriteLine("Click no Radiobutton Masculino");

    selecionadoFeminino = driver.FindElement(By.Id("u_0_9")).Selected;
    selecionadoMasculino = driver.FindElement(By.Id("u_0_a")).Selected;

    Console.WriteLine("Feminino : " + selecionadoFeminino + " Masculino : " +
selecionadoMasculino);
}
}

```

4.7.6 Checkbox / Frame / Scroll To Element

Crie uma nova classe chamada CheckTest:

```

[TestClass]
public class CheckTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

        driver.Navigate().GoToUrl("https://developer.mozilla.org/en-
US/docs/Web/HTML/Element/input/checkbox");

        bool selecionadoSubscribe;

        //Mudar foco para o frame em qual o elemento se encontra
        driver.SwitchTo().Frame(driver.FindElement(By.Id("frame_Value")));

        //Realiza o scroll até o elemento
        var element = driver.FindElement(By.Id("subscribeNews"));
        Actions actions = new Actions(driver);
        actions.MoveToElement(element);
        actions.Perform();

        //Verifica se esta selecionado
        selecionadoSubscribe = driver.FindElement(By.Id("subscribeNews")).Selected;
        Console.WriteLine("Selecionado : " + selecionadoSubscribe);
    }
}

```

```

    //Clica no elemento
    driver.FindElement(By.Id("subscribeNews")).Click();
    Console.WriteLine("Click no Checkbox");

    //Verifica se esta selecionado
    selecionadoSubscribe = driver.FindElement(By.Id("subscribeNews")).Selected;
    Console.WriteLine("Selecionado : " + selecionadoSubscribe);

    //Retorna o foco para a página principal
    driver.SwitchTo().DefaultContent();
}
}

```

4.7.7 Dropdown (Select)

Crie uma nova classe chamada DropDownTest:

```

[TestClass]
public class DropDownTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

        driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/buscaelementos/dropdownselect");
    }

    var combo = driver.FindElement(By.ClassName("browser-default"));
    //Funciona apenas quando a combo possui a tag 'Select'
    SelectElement select = new SelectElement(combo);
    //select.SelectByValue("3");
    select.SelectByText("Internet Explorer");

    }
}

```

4.7.8 Tabela

Crie uma nova classe chamada TabelaTest:

```

[TestClass]
public class TabelaTest
{
    [TestMethod]
    public void TestMethod1()
    {

        IWebDriver driver = new ChromeDriver();

        driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/buscaelementos/table");

        //Armazena a tabela
        var table = driver.FindElement(By.TagName("table"));

        //Realiza uma busca encadeada a partir da tabela (table.findElements), onde são buscadas
        as linhas da tabela
        //Utilizando o FindElements que retorna uma lista de objetos
        //No FindElements caso nenhum elemento seja encontrado retorna uma lista vazia sem dar
        erro
    }
}

```

```

var rows = table.FindElements(By.TagName("tr"));
//Percorre pelas linhas encontradas
foreach (var row in rows)
{
    //Realiza uma busca encadeada apartir da linha atual, onde são buscadas as células
individualmente
    var rowTds = row.FindElements(By.TagName("td"));
    //Itera por cada célula imprimindo seu conteúdo
    foreach (var td in rowTds)
    {
        Console.Write(" | " + td.Text);
    }
    //Quebra de linha no output antes de buscar uma nova linha da tabela.
    Console.WriteLine("\n");
}
}
}

```

4.7.9 Alert

Crie uma nova classe chamada AlertTest:

```

[TestClass]
public class AlertTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new FirefoxDriver();
        //IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/mudancadefoco/alert");

        driver.FindElement(By.XPath("//button[@onclick='jsAlert()']")).Click();

        //driver.SwitchTo().Alert().Accept();

        //Ler texto do Alert
        var texto = driver.SwitchTo().Alert().Text;

        driver.SwitchTo().Alert().Dismiss();

        Thread.Sleep(2000);

        driver.FindElement(By.XPath("//button[@onclick='jsConfirm()']")).Click();

        driver.SwitchTo().Alert().Accept();

        //driver.SwitchTo().Alert().Dismiss();

        Thread.Sleep(2000);

        //Não funciona no Chrome
        driver.FindElement(By.XPath("//button[@onclick='jsPrompt()']")).Click();

        driver.SwitchTo().Alert().SendKeys("Wheee");
        driver.SwitchTo().Alert().Accept();

    }
}

```

4.7.10 Janela/Aba

Crie uma nova classe chamada JanelaTest:

```
[TestClass]
public class JanelaTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/mudancadefoco/janela");

        Console.WriteLine("Abas Abertas: " + driver.WindowHandles.Count);
        Console.WriteLine("Url da janela atual: " + driver.Url);

        //Clico no botão para abrir uma nova aba
        driver.FindElement(By.CssSelector(".btn.waves-light.red")).Click();
        Console.WriteLine("Abas Abertas: " + driver.WindowHandles.Count);

        //Armazena o "ID" da ultima janela através do WindowHandle
        string newWindowHandle = driver.WindowHandles.Last();
        var newWindow = driver.SwitchTo().Window(newWindowHandle);

        Console.WriteLine("Url da janela atual: " + driver.Url);
    }
}
```

4.7.11 Modal

Crie uma nova classe chamada ModalTest:

```
[TestClass]
public class ModalTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/mudancadefoco/modal");

        driver.FindElement(By.CssSelector(".waves-light.btn.modal-trigger")).Click();

        //Talvez seja necessário aumentar este tempo em sua máquina
        //Comente este tempo e execute o teste N vezes
        //Repare que em algumas o texto será não exibido no output
        //Temos que descomentar esse trecho que faz com que aguarde X segundos,
        //dando tempo do texto ser exibido e lido abaixo
        //Thread.Sleep(2000);

        //Lê o texto da modal
        var texto = driver.FindElement(By.CssSelector(".modal-content")).Text;

        //É necessário aguardar para que o texto fique visivel
        //Caso contrário a mensagem não será impressa no output ao final do teste
        //O teste não quebra porque estes elementos estão sempre no html, porém ocultos
        //Mas para que o texto seja de fato lido, é necessário que ele seja de fato exibido na
tela do navegador
        Console.WriteLine(texto);
    }
}
```

```
    }  
}
```

4.7.12 Autocomplete

Crie uma nova classe chamada AutocompleteTest:

```
[TestClass]  
public class AutocompleteTest  
{  
    [TestMethod]  
    public void ExemploGoogleAutocomplete()  
    {  
        IWebDriver driver = new ChromeDriver();  
  
        driver.Navigate().GoToUrl("https://www.google.com/");  
  
        Thread.Sleep(4000);  
  
        driver.FindElement(By.Name("q")).SendKeys("Estado Sa");  
  
        //Necessário esperar as sugestões serem carregadas  
        //Não é o ideal, mas apenas para fins de aprendizado neste momento usaremos o  
Thread.Sleep  
        Thread.Sleep(4000);  
  
        //Armazena a lista de sugestões  
        var sugestoes =  
driver.FindElements(By.XPath("//li[@jsaction='click:.CLIENT;mouseover:.CLIENT']"));  
  
        foreach (var item in sugestoes)  
        {  
            if (item.Text == "estado santa catarina")  
            {  
                item.Click();  
                //Já que a lista não exisitirá após o clique  
                //Usamos o break para sair, caso tente iterar pelo próximo item da lista que não  
está mais sendo exibida  
                // é lançado o erro StaleElementException,  
                break;  
            }  
        }  
    }  
  
    [TestMethod]  
    public void ExemploBatistaMeiaBoca()  
    {  
        IWebDriver driver = new ChromeDriver();  
  
driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/widgets/autocomplete");  
  
        //Elemento possui ID dinâmico, onde apenas o início é mantido como padrão.  
        //Assim usamos um xpath que usa a condição de busca 'starts-with'  
        driver.FindElement(By.XPath("//input[@data-target[starts-with(., 'autocomplete-options-')]]")).SendKeys("Sa");  
  
        //Necessário esperar as sugestões serem carregadas  
        //Não é o ideal, mas apenas para fins de aprendizado neste momento usaremos o  
Thread.Sleep  
        Thread.Sleep(2000);  
    }  
}
```

```

var fonte = driver.PageSource;

//Armazena a lista de sugestões
var sugestoes = driver.FindElements(By.ClassName("highlight"));

foreach (var item in sugestoes)
{
    Console.WriteLine(item.Text);
    //O texto é cortado devido a forma como foi implementado no site =/

    if (item.Text == "santa catarina")
    {
        item.Click();
        //Já que a lista não exisitirá após o clique
        //Usamos o break para sair, caso tente iterar pelo próximo item da lista que não
        está mais sendo exibida
        // é lançado o erro StaleElementException,
        break;
    }
}

}

```

4.7.13 Mouseover

Crie uma nova classe chamada MousehoverTest:

```

[TestClass]
public class MouseHoverTest
{
    [TestMethod]
    public void TestMethod1()
    {
        IWebDriver driver = new ChromeDriver();

        driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/iteracoes/mouseover");

        //Elemento sobre o qual o mouse deve ficar em cima
        var elemento = driver.FindElement(By.ClassName("activator"));

        //Realizar ação de mover o mouse para cima do elemento especificado
        Actions action = new Actions(driver);
        action.MoveToElement(elemento).Perform();

        //Aguardar para o texto ser exibido
        // Não é o ideal, mas apenas para fins de aprendizado neste momento usaremos o
        Thread.Sleep
        Thread.Sleep(2000);

        //Imprimimos o texto exibido após a ação do mouse
        Console.WriteLine(driver.FindElement(By.ClassName("card-reveal")).Text );
    }
}

```

4.7.14 Upload de arquivo

Crie uma nova classe chamada UploadTest:

```

[TestClass]
public class UploadTest
{

```

```

[TestMethod]
public void TestMethod1()
{
    IWebDriver driver = new ChromeDriver();

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/outros/uploaddearquivos");

    //Mapear o campo geralmente do tipo input que corresponda ao upload
    //Passar o caminho com extensão do arquivo a ser enviado

driver.FindElement(By.Id("upload")).SendKeys(@"C:\Users\rafael.camargo\Pictures\teste.png");
}

}

```

4.7.15 Autenticação

Crie uma nova classe chamada AutenticacaoTest:

```

[TestClass]
public class AutenticacaoTest
{
    [TestMethod]
    public void AutenticacaoAlert()
    {
        IWebDriver driver = new FirefoxDriver();

        driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/basicauth/home");
        //Não funciona no Chrome
        IAlert alert = driver.SwitchTo().Alert();
        alert.SendKeys("admin" + Keys.Tab + "admin");
        alert.Accept();
    }

    [TestMethod]
    public void AutenticacaoUrl()
    {
        IWebDriver driver = new ChromeDriver();

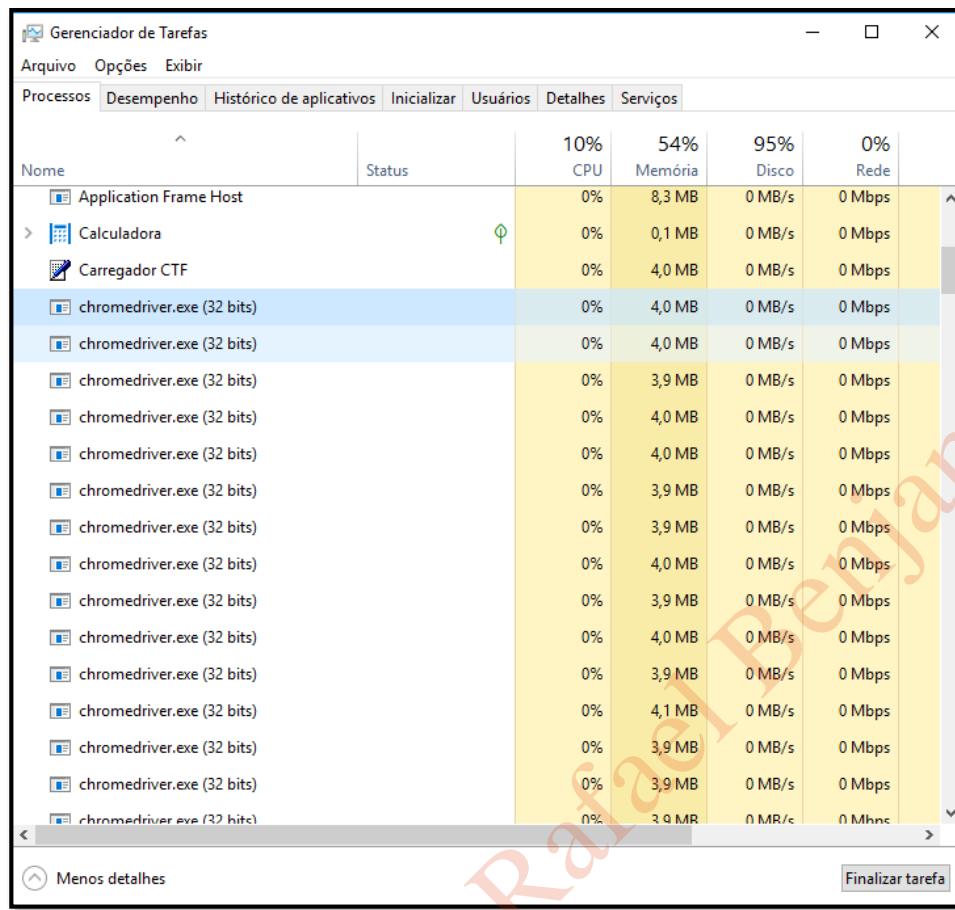
        //Alternativa quando o usuário e senha são passados diretamente pela url:
        //formato da url: https://username:password@www.example.com

driver.Navigate().GoToUrl("https://admin:admin@automacaocombatista.herokuapp.com/basicauth/home");
    }
}

```

4.7.16 Encerrando múltiplos processos de drivers

Caso execute vários testes sem fechar o navegador usando quit ou em casos em que o teste falhou e o navegador não foi fechado, note que vários processos com o navegador executado ficarão em aberto no seu Gerenciador de Tarefas do Windows:



Uma maneira mais rápida de fechar as diversas instâncias é usando o prompt de comando com o seguinte comando:

Taskkill /IM <nome_do_processo> /F

```

Windows [versão 10.0.17134.648]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\rafael.camargo>Taskkill /IM chromedriver.exe /F
ÊXITO: o processo "chromedriver.exe" com PID 13128 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 8428 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 1916 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 12052 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 14972 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 14968 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 4332 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 16348 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 13684 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 17856 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 18292 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 19144 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 19808 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 9052 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 20988 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 22428 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 20948 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 20680 foi finalizado.
ÊXITO: o processo "chromedriver.exe" com PID 12392 foi finalizado.

C:\Users\rafael.camargo>

```

Para encerrar processos do chrome driver:

Taskkill /IM chromedriver.exe /F

Para encerrar processos do Firefox driver:

```
Taskkill /IM geckodriver.exe /F
```

Para encerrar processos do Internet Explorer driver:

```
Taskkill /IM IEDriverServer.exe /F
```

4.8 Esperas

Até o momento vimos alguns casos em que tivemos que forçar uma espera para que nosso código pudesse identificar o elemento e prosseguir com sua interação, geralmente ocorre com elementos Ajax (*Asynchronous JavaScript & XML*), são basicamente elementos carregados internamente na página sem a necessidade de recarregar toda a página.

Usamos a pior forma possível, o comando **Thread.Sleep()**, que faz com que nosso programa seja interrompido pelo tempo que configuramos dentro desta função. Esta é a pior forma, pois o tempo que definimos SEMPRE será aguardado em sua totalidade. Se configurarmos um sleep de 10 segundos, então o teste será pausado por 10 segundo independente se o elemento pelo qual estamos procurando foi exibido ou não.

4.8.1 Espera Implícita

A espera implícita basicamente aguarda até um determinado tempo caso o elemento não seja encontrado. Em casos mais simples aonde a página possui uma lentidão este tipo de espera pode funcionar, apesar de ainda não ser a forma recomendada.

Exemplo:

```
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
```

Neste exemplo acima, configuramos um tempo de espera de até 10 segundos para que o nosso driver tente encontrar um elemento. Caso ele encontre o elemento antes desse tempo, o fluxo segue normalmente sem ter que aguardar o tempo restante esgotar. Mas caso o elemento não seja encontrado após o timeout ser atingido, a exceção para timeout é lançada.

Esta espera é configurada para a instância do driver atual, ou seja, realizamos sua configuração apenas uma vez. Para todo o restante do código o valor configurado será utilizado, ou até que uma nova instrução deste mesmo comando sobrescreva este valor (não recomendado).

4.8.2 Espera Explícita

A espera explícita é mais refinada e utilizada a cada elemento. É necessário criar uma instância do tipo `WebDriverWait`, informar o tempo limite desejado para este elemento e em seguida definir o tipo de condição que será utilizada para aguardar este elemento, como por exemplo aguardar que o elemento seja visível, ou clicável, ou mesmo que aguardar que fique invisível.

Este tipo de espera possui um nível maior de detalhamento para definir a forma com que o teste prosseguirá. Mas seu funcionamento é semelhante, caso o elemento seja encontrado antes do tempo limite definido o teste segue adiante, caso o tempo seja excedido uma exceção é lançada.

Como criamos uma espera a cada elemento, podemos custmoizar mais a forma com que trabalhamos. Se um determinado ponto possui uma lentidão maior, podemos customizar um timeout maior apenas naquele ponto. Ao contrário da espera implícita, onde definimos um tempo único para todos os elementos.

Para utilizar as esperas explícitas baixe o seguitne pacote NuGet:

- DotNetSeleniumExtras.WaitHelpers

Exemplo:

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("Masc")));
```

Onde:

`WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));` : Cria uma nova instância do tipo WebDriverWait, que tem como parâmetros o driver atual, e tempo limite a ser aguardado.

`wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("Masc")));` : A partir da instância de WebDriverWait criada, chamamos a função Until. Esta por sua vez realizar o tipo de aguardo, neste exemplo usa o ExpectedConditions.ElementIsVisible, que aguardar até o elemento seja visível.

Dentro das funções disponíveis em ExpectedConditions, temos vários tipos de condições, como por exemplo aguardar que o elemento seja clicável, ou que esteja selecionado, ou que fique invisível, ou aguardar que seu texto seja algum específico. Então sempre que selecionar alguma, leia com atenção os parâmetros que cada uma pede, neste caso pediu apenas o identificador do elemento (By).

Também repare que precisei usar o SeleniumExtras.WaitHelpers antes do expected conditions, é que esta função existe em outra biblioteca auxiliar do selenium causando ambiguidade, por isso é necessário especificar que ela será lida da SeleniumExtras.WaitHelpers.

Importante:

A espera implícita é executada no lado ‘remoto’ do Webdriver, nos drivers. Onde cada driver pode possuir uma implementação interna diferente dos demais. Já a espera explícita é executada no lado ‘local’ do selenium, sendo que usa a mesma implementação e esta implementação é por fim enviada aos navegadores. O uso misto destes tipos de espera pode causar requisições com comportamentos indefinidos.

Alerta: Não misture o uso de esperar implícitas e explícitas! Fazer isso pode causar tempos de espera imprevisíveis. Por exemplo, configurar uma espera implícita de 10 segundos e uma explícita de 15 segundos pode causar um timeout depois de 20 segundos.

Trecho retirado de: https://docs.seleniumhq.org/docs/04_webdriver_advanced.jsp#explicit-and-implicit-waits

Mais sobre:

- <http://www.eliasnogueira.com/selenium-nao-misture-esperas-implicitas-com-explicitas/>
- <https://stackoverflow.com/questions/15164742/combining-implicit-wait-and-explicit-wait-together-results-in-unexpected-wait-ti/15174978#15174978>

Para exemplificar o uso da espera explícita, vamos voltar no teste da [modal](#) que foi feito anteriormente.

Estavámos usando o Thread.Sleep(), mas como mencionado acima não é recomendado. Já que pode variar de acordo com a máquina de cada um e o teste sempre ficará parado por este tempo determinado.

Se caso tentarmos usar a espera implícita, esta opção não nos ajudará agora. A espera implícita apenas aguarda até que o elemento seja encontrado, neste caso estamos aguardando um texto ficar visível, mas o elemento aonde este texto se encontra já existe na tela e sempre será encontrado, mesmo ainda não estando visível na tela.

Repare que os elementos que buscamos já existem no HTML, mesmo com a modal fechada:

```
<div class="modal-content">
  <h4>Model Teste</h4>
  <p>Pensando mais a longo prazo, a execução dos pontos do programa garante a contribuição de um grupo importante na determinação da gestão inovadora de qual fazemos parte. Ainda assim, existem dúvidas e respeito de como o surgimento do comércio virtual desafia a capacidade de equalização das diretrizes de desenvolvimento para o futuro. A certificação de metodologias que nos auxiliam a lidar com o julgamento imparcial das eventualidades talvez venha a ressaltar a relevância do impacto na agilidade decisória. Acima de tudo, é fundamental ressaltar que a percepção das dificuldades auxilia a preparação e a composição das novas proposições. </p>
```

Removendo o Thread.Sleep() e realizando diversas execuções, repare que em algumas, o texto não é exibido no output do resultado final:

Test Name: TestMethod1
Test Outcome: Aprovado

Standard Output

TestMethod1 Copiar Tudo
Source: ModalTeste.cs linha 14
TestMethod1
Elapsed time: 0:00:05,3845716
Output

Apenas caso o elemento contendo o texto não existisse no HTML, e fosse criado apenas após o clique no botão, então neste caso a espera implícita funcionaria corretamente, aguardando o elemento criado e consequentemente seu texto exibido.

Ao executar este teste sem nenhum tipo de espera, ele funciona as vezes sim e as vezes não. Sempre devemos garantir que nossos testes estão passando em todas as execuções, por isso a importância de estressar o teste e executá-lo diversas vezes e verificar se não é necessário mais algum ajuste para que o teste funcione em todas as situações.

Para que este teste sempre funcione, inclua a espera explícita para a modal após o clique como no código abaixo:

```

[TestMethod]
public void TestMethod1()
{
    IWebDriver driver = new ChromeDriver();

    //Não funciona, o elemento sempre existirá no html mesmo não estando visível na tela
    //Faça o teste descomentando esta linha abaixo
    //driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);

driver.Navigate().GoToUrl("https://automacaocombatista.herokuapp.com/mudancadefoco/modal");

    driver.FindElement(By.CssSelector(".waves-light.btn.modal-trigger")).Click();

    //Talvez seja necessário aumentar este tempo em sua máquina
    //Comente este tempo e execute o teste N vezes
    //Repare que em algumas o texto será não exibido no output
    //Temos que descomentar esse trecho que faz com que aguarde X segundos,
    //dando tempo do texto ser exibido e lido abaixo
    //Thread.Sleep(2000);

    //Com o uso da espera explícita
    //Aguardará que o elemento fique visível antes de prosseguir
    //Comente a linha de espera implícita antes de usar esta
    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(5));
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.CssSelector(".modal-content")));

    //Lê o texto da modal
    var texto = driver.FindElement(By.CssSelector(".modal-content")).Text;

    //É necessário aguardar para que o texto fique visível
    //Caso contrário a mensagem não será impressa aqui
    //O teste não quebra porque estes elementos estão sempre no html, porém ocultos
    //Mas para que o texto seja de fato lido, é necessário que ele seja de fato exibido na
    //tela do navegador
    Console.WriteLine(texto);

    driver.Quit();
}

```

Execute o teste diversas vezes novamente, repare que agora em todas as vezes texto será exibido já que sempre aguarda o texto ficar visível antes de prosseguir (Lembrando que para esta situação a condição de ficar visível também funciona, mas nem sempre esta será a correta. Varia de acordo com o que você deseja aguardar.)

Incluimos uma espera apenas no elemento da modal, os demais elementos continuam sem um timeout definido (assumindo que não foi incluída a espera implícita). Caso fosse um teste maior e algum outro elemento necessitasse de uma espera, iríamos configurar outra espera explícita para o elemento em questão.

<p>Test Name: TestMethod1</p> <p>Test Outcome: Aprovado</p> <p>Standard Output</p> <p>Modal Teste</p> <p>Pensando mais a longo prazo, a execução dos pontos do programa garante a contribuição de um grupo importante na determinação da gestão inovadora da qual fazemos parte. Ainda assim, existem dúvidas a respeito de como o surgimento do comércio virtual desafia a capacidade de equalização das diretrizes de desenvolvimento para o futuro. A certificação de metodologias que nos auxiliam a lidar com o julgamento imparcial das eventualidades talvez venha a ressaltar a relatividade do impacto na agilidade decisória. Acima de tudo, é fundamental ressaltar que a percepção das dificuldades auxilia a preparação e a composição das novas proposições.</p>
--

Para visualizar uma possível falha, altere o localizador dentro da ExpectedCondition para algo inexistente, veja a exceção de timeout sendo lançada pois não encontrou o elemento dentro do tempo esperado:



The screenshot shows an error message from a Selenium WebDriver. The message is as follows:

```
TestMethod1
Source: ModalTeste.cs linha 14
X TestMethod1
Message: Test method
ProjetoAutomacao.Testes_Exemplos_Elementos.ModalTeste.TestMethod1 threw
exception:
OpenQA.Selenium.WebDriverTimeoutException: Timed out after 5 seconds --->
OpenQA.Selenium.NoSuchElementException: no such element: Unable to locate
element: {"method":"css selector", "selector":".masodal-content"}
(Session info: chrome=75.0.3770.80)
(Driver info: chromedriver=74.0.3729.6
(255758eccf3d244491b8a1317aa76e1ce10d57e9-refs/branch-heads/3729@(#29)),platform=Windows NT 10.0.17134 x86_64)
Elapsed time: 0:00:12,4369458
▲ StackTrace:
  RemoteWebDriver.UnpackAndThrowOnError(Response errorResponse)
  RemoteWebDriver.Execute(String driverCommandToExecute, Dictionary`2 parameters)
  RemoteWebDriver.FindElement(String mechanism, String value)
  RemoteWebDriver.FindElementByCssSelector(String cssSelector)
```

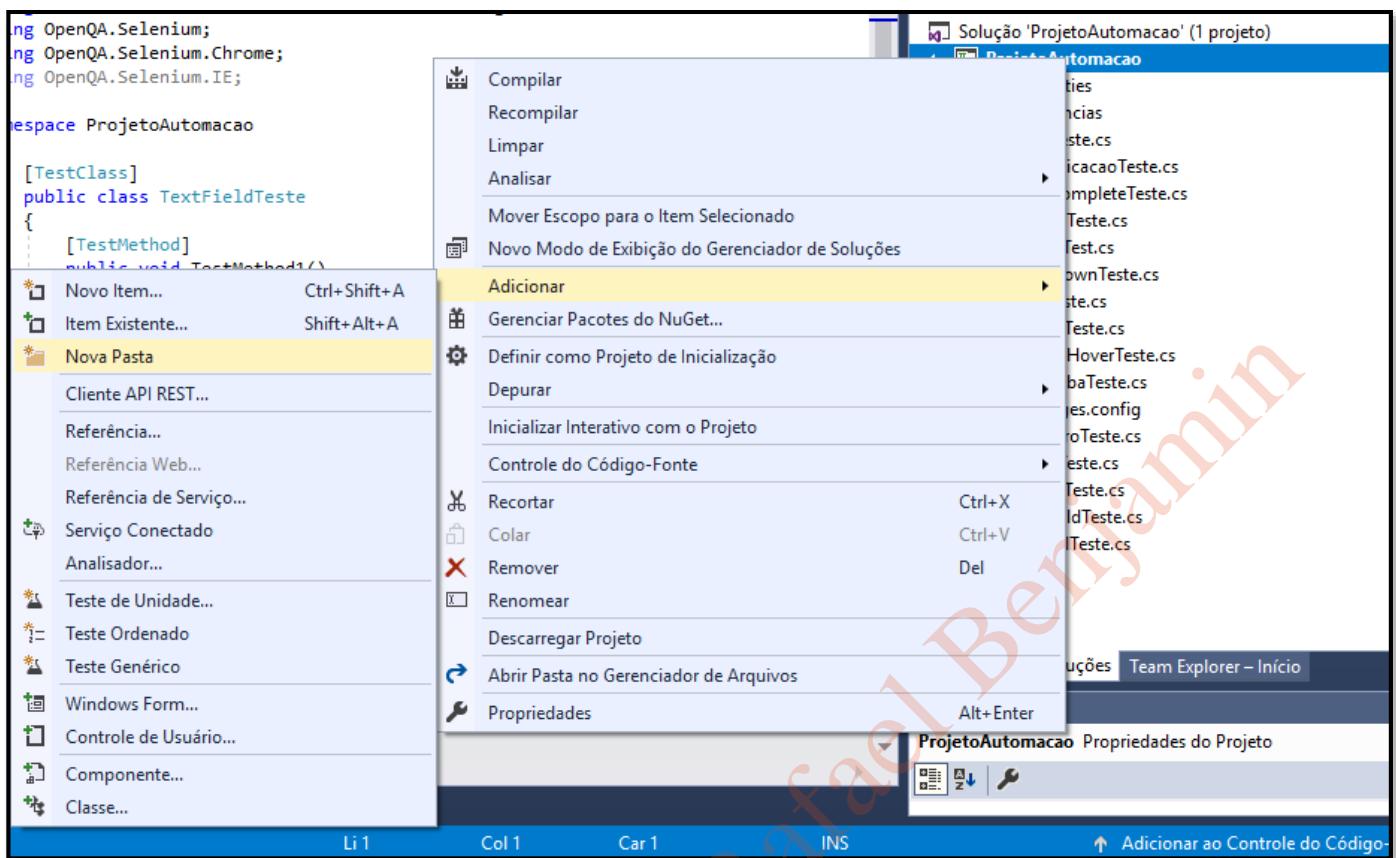
4.9 Exemplo Login

Agora que já vimos os principais elementos, vamos iniciar escrever os primeiros testes e dar início a estruturação do framework.

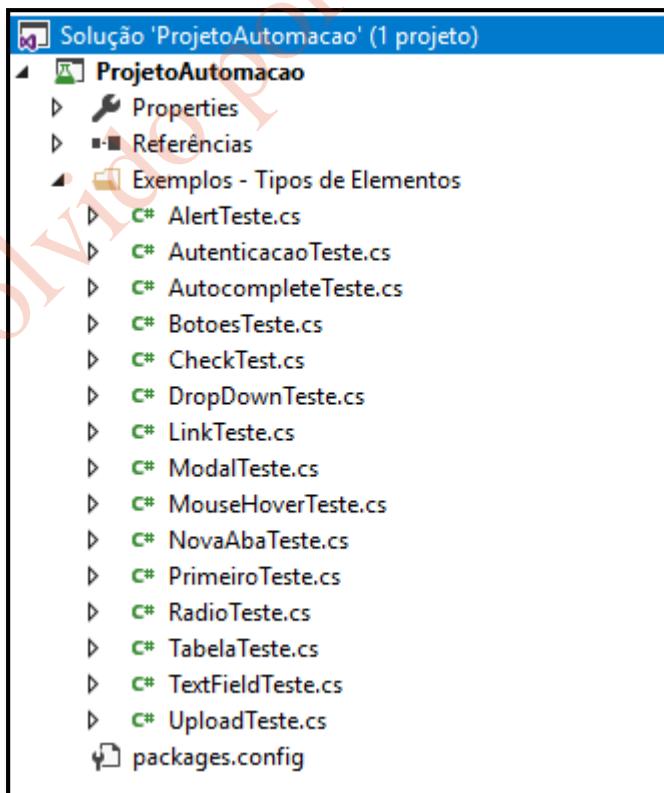
Para melhor organização, crie um novo projeto (dentro da mesma solução) ou apenas salve as classes criadas até o momento dentro de uma pasta a parte em seu projeto.

Em meu projeto apenas irei mover as classes dentro de uma pasta.

Para criar uma nova pasta, **clique no projeto com o botão direito > Adicionar > Nova Pasta**



Após criada, renomeie a pasta e mova apenas os arquivos de classes que criamos.



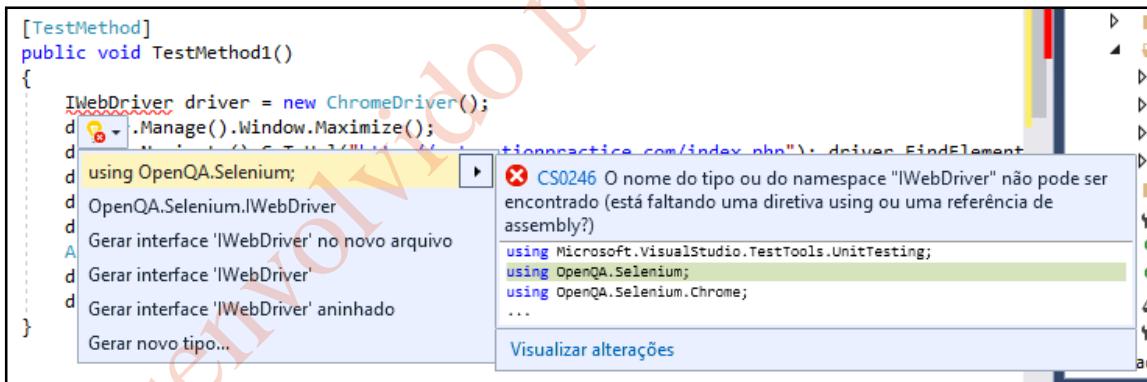
Após organizado, crie uma nova classe de testes e renomeie o para '**LoginTest**'.

Com isso podemos escrever nosso primeiro teste:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;

namespace FrameworkQarti
{
    [TestClass]
    public class LoginTest
    {
        [TestMethod]
        public void LoginValido()
        {
            IWebDriver driver = new ChromeDriver();
            driver.Manage().Window.Maximize();
            driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
            driver.FindElement(By.XPath("//a[@href='http://automationpractice.com/index.php?controller=my-account']")).Click();
            driver.FindElement(By.Id("email")).SendKeys("teste@testing.com");
            driver.FindElement(By.Id("passwd")).SendKeys("1234qwer");
            driver.FindElement(By.Id("SubmitLogin")).Click();
            Assert.IsNotNull(driver.FindElement(By.CssSelector(".account")));
            driver.Quit();
            driver.Dispose();
        }
    }
}
```

Se após a inclusão do código algum comando fique com um erro marcado, mantenha o cursor sobre o erro e clique em ‘Mostrar possíveis correções’, então serão exibidas as sugestões indicando um possível pacote que não foi importado.



Também tome cuidado com o Namespace, que varia de acordo com o nome de seu projeto. Sempre que copiar algum código de exemplo lembre-se de verificar se o namespace está de acordo com o seu projeto.

Entendendo o código:

IWebDriver driver = new ChromeDriver(); : Criamos uma nova instância do driver para o browser Chrome. Após este comando ser executado o navegador será aberto. Todos os comandos que serão efetuados devem ser feitos a partir desta instância do driver.

driver.Manage().Window.Maximize(); : Maximiza a janela do navegador.

driver.Navigate().GoToUrl("http://automationpractice.com/index.php"); : Navega para a url apontada.

```

driver.FindElement(By.XPath("//a[@href='http://automationpractice.com/index.php?controller=my-account']")).Click(); : Clicar no link 'Sign In' direcionando para a página de login.
O comando driver.FindElement(By.) é utilizado para encontrar elementos da tela. O identificador usado aqui é o Xpath, que realiza uma busca por um link que contenha a url 'http://automationpractice.com/index.php?controller=my-account' e realizar um comando de clique pelo método .Click();

driver.FindElement(By.Id("email")).SendKeys("teste@testing.com");
driver.FindElement(By.Id("passwd")).SendKeys("1234qwer"); : Buscam os campos para preencher o Email e Senha e enviam os valores a serem preenchidos através do método .SendKeys();

driver.FindElement(By.Id("SubmitLogin")).Click();: Clique no botão 'Sign In' para efetuar o login.

Assert.IsNotNull(driver.FindElement(By.CssSelector(".account"))); : O Comando 'Assert' é usado para realizar as verificações (Aserções). Neste caso, verifica que o elemento contendo o nome do usuário foi encontrado e garante que o mesmo encontra-se na área logada do site.

driver.Quit(); : Fecha todas as janelas abertas nesta instância do driver.

driver.Dispose(); : Encerra os processos iniciados pelo webdriver na memória.

```

4.9.1 Exercício:

Crie uma nova classe de testes com o nome de '**ContactUsTest**' e Inclua um novo teste para envio de mensagem válida através da área de atendimento do site.

Passos do teste:

- Acessar o site: <http://automationpractice.com/index.php>
- Clicar no link 'Contact Us'
- Preencha a combo 'Subject Heading' com qualquer opção
- Preencha o campo 'Email address' com qualquer email válido
- Preencha o campo 'Order reference' com um número qualquer
- Insira um anexo no campo 'Attach File'
- Preencha o campo 'Message' com uma mensagem qualquer
- Clique em enviar
- Valide a mensagem de sucesso

Sugestão de resolução:

```

[TestMethod]
public void FaleConoscoValido()
{
    IWebDriver driver = new ChromeDriver();
    driver.Manage().Window.Maximize();
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    driver.FindElement(By.Id("contact-link")).Click();
    SelectElement combo = new SelectElement(driver.FindElement(By.Id("id_contact")));
    combo.SelectByText("Customer service");
    driver.FindElement(By.Id("email")).SendKeys("teste@teste.com");
    driver.FindElement(By.Id("id_order")).SendKeys("#123456789");
    driver.FindElement(By.Id("fileUpload")).SendKeys(@"C:\Dev\Arquivo.txt");
    driver.FindElement(By.Id("message")).SendKeys("Mensagem de Teste");
    driver.FindElement(By.Id("submitMessage")).Click();
    string textoRetornado = driver.FindElement(By.CssSelector(".alert.alert-success")).Text;
    Assert.AreEqual("Your message has been successfully sent to our team.", textoRetornado);
    Console.WriteLine(textoRetornado);
}

```

```

        driver.Quit();
        driver.Dispose();
    }
}

```

Notas: A idéia é validar algo que possa confirmar o teste, sempre buscando algum identificador único que possa garantir o sucesso do teste. Neste caso a validação da mensagem de confirmação. Lembrando de sempre verificar se o teste realmente está funcionando, por exemplo alterando a mensagem de sucesso esperada para outro texto, ou buscando um elemento inválido devem exibir a falha nos testes.

Para o preenchimento da combo foi necessário realizar o download do pacote **Selenium.Support**

O Arquivo enviado no fileUpload deve apontar para um arquivo válido dentro do seu computador.

4.9.2 Exercício:

Passos do teste:

- Acessar o site: <http://automationpractice.com/index.php>
- Clicar no link ‘Contact Us’
- Preencha a combo ‘Subject Heading’ com qualquer opção
- Preencha o campo ‘Email address’ com qualquer email em formato inválido
- Preencha o campo ‘Order reference’ com um número qualquer
- Insira um anexo no campo ‘Attach File’
- Preencha o campo ‘Message’ com uma mensagem qualquer
- Clique em enviar
- Valide a mensagem de alerta sobre o email inválido

Sugestão de resolução:

```

[TestMethod]
public void FaleConoscoEmailInvalido()
{
    IWebDriver driver = new ChromeDriver();
    driver.Manage().Window.Maximize();
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    driver.FindElement(By.Id("contact-link")).Click();
    SelectElement combo = new SelectElement(driver.FindElement(By.Id("id_contact")));
    combo.SelectByText("Customer service");
    driver.FindElement(By.Id("email")).SendKeys("emailInvalido");
    driver.FindElement(By.Id("id_order")).SendKeys("#123456789");
    driver.FindElement(By.Id("fileUpload")).SendKeys(@"C:\Dev\Arquivo.txt");
    driver.FindElement(By.Id("message")).SendKeys("Mensagem de Teste");
    driver.FindElement(By.Id("submitMessage")).Click();
    string textoRetornado = driver.FindElement(By.CssSelector(".alert.alert-danger")).Text;
    textoRetornado = textoRetornado.Replace("\r\n", " ");
    Assert.AreEqual("There is 1 error Invalid email address.", textoRetornado);
    driver.Quit();
    driver.Dispose();
}
}

```

Notas: Ao inspecionar o texto retornado pela mensagem o seguinte valor é exibido: “There is 1 error\r\nInvalid email address.” Os caracteres \r\n representam: \r – carriage return, retorna o texto para o inicio da linha. \n – representa uma quebra de linha.

Temos que normalizar o texto removendo estes caracteres para poder realizar a validação do texto.

O texto exibido no site é o seguinte:

There is 1 error

1. Invalid email address.

O número “1.” Antes de “Invalid email address” não é exibido pois em seu HTML ele não é um número. Foi usada uma lista ordenada (representado pela tag ``) e o número é gerado apenas na tela, assim não podendo ser validado como parte do texto.

The screenshot shows a browser's developer tools with the 'Elements' tab selected. An alert box is displayed with the message 'There is 1 error' and the list item '1. Invalid email address.'. Below the alert, the browser's DOM structure is shown in the Elements panel:

```
<div class="alert alert-danger">
  ::before
  <p>There is 1 error</p>
  <ol>
    <li>Invalid email address.</li>
  </ol>
</div>
```

A red arrow points to the opening tag of the `` element.

4.10 Atributos de Inicialização e Finalização dos testes

São atributos que configuram a execução de um trecho de código em determinado momento do fluxo de execução.

- **[AssemblyInitialize]**
Executado uma vez antes na inicialização antes da execução de todos os testes.
Não pode ser herdado.
- **[ClassInitialize]**
Executado uma vez antes da inicialização da classe de teste.
Não pode ser herdado.
- **[TestInitialize]**
Executado antes de cada teste.
Pode ser herdado.
- **[AssemblyCleanup]**
Executado uma vez após a execução de todos os testes.
Não pode ser herdado.
- **[ClassCleanup]**
Executado uma vez após todos os testes desta classe serem executados.
Não pode ser herdado.
Não é garantido que seja executado logo após todos os testes de sua classe.
- **[TestCleanup]**
Executado após cada teste finalizado.
Pode ser herdado.

Para verificar na prática como é funcionamento deste fluxo, vamos testar algumas variações.

Cenário 1: Todos os atributos definidos em uma única classe.

Crie uma nova pasta chamada ‘**TesteAtributos**’ para que os testes a seguir não se misturem com nossos testes principais. Dentro da pasta crie uma classe de testes chamada ‘**TesteDeAtributos1**’ e inclua o código abaixo.

```
[TestClass]
```

```

public class TesteDeAtributos1
{
    [AssemblyInitialize]
    public static void AssemblyInitA(TestContext context)
    {
        //Executado uma vez antes na inicialização antes da execução de todos os testes.
        Console.WriteLine("Executando o AssemblyInitialize Definido na classe 1");
    }

    [AssemblyCleanup]
    public static void AssemblyCleanupA()
    {
        //Executado uma vez após a execução de todos os testes.
        Console.WriteLine("Executando o AssemblyCleanup Definido na classe 1");
    }

    [ClassInitialize]
    public static void TestFixtureSetupA(TestContext context)
    {
        //Executado uma vez antes da inicialização da classe de teste.
        Console.WriteLine("Executando o ClassInitialize Definido na classe 1");
    }

    [TestInitialize]
    public void SetupA()
    {
        //Executado antes de cada teste.
        Console.WriteLine("Executando o TestInitialize Definido na classe 1");
    }

    [ClassCleanup]
    public static void TestFixtureTearDownA()
    {
        //Executado uma vez após todos os testes desta classe serem executados.
        Console.WriteLine("Executando o ClassCleanup Definido na classe 1");
    }

    [TestCleanup]
    public void TearDownA()
    {
        //Executado após cada teste finalizado.
        Console.WriteLine("Executando o TestCleanup Definido na classe 1");
    }

    [TestMethod]
    public void MetodoDeTesteUm()
    {
        //Teste a ser executado
        Console.WriteLine("Metodo de Teste Um foi executado");
    }

    [TestMethod]
    public void MetodoDeTesteDois()
    {
        //Teste a ser executado
        Console.WriteLine("Metodo de Teste Dois foi executado");
    }
}

```

Após a execução veremos o seguinte resultado.

Test Name: MetodoDeTesteUm

Test Outcome: Aprovado

Standard Output

```
Executando o AssemblyInitialize Definido na classe 1
Executando o ClassInitialize Definido na classe 1
Executando o TestInitialize Definido na classe 1
Metodo de Teste Um foi executado
Executando o TestCleanup Definido na classe 1
Executando o ClassCleanup Definido na classe 1
Executando o AssemblyCleanup Definido na classe 1
```

Todas as fases serão exibidas como esperado.

Inclua mais um teste dentro desta classe.

```
[TestMethod]
public void MetodoDeTesteDois()
{
    //Teste a ser executado
    Console.WriteLine("Metodo de Teste Dois foi executado");
}
```

O comportamento do TestInitialize/TestCleanup ficará mais claro agora que temos dois testes distintos.

Os atributos AssemblyInitialize/AssemblyCleanup serão exibidos apenas no início e fim dos testes. O

ClassInitialize/ClassCleanup neste caso também serão exibidos apenas no final dos testes já que os dois testes estão definidos na mesma classe.

A ordem exibição para os atributos AssemblyInitialize/AssemblyCleanup e ClassInitialize/ClassCleanup pode variar dependendo da ordem de execução, mas ainda sim serão executados apenas uma vez.

Test Name: MetodoDeTesteUm

Test Outcome: Aprovado

Standard Output

```
Executando o AssemblyInitialize Definido na classe 1
Executando o ClassInitialize Definido na classe 1
Executando o TestInitialize Definido na classe 1
Metodo de Teste Um foi executado
Executando o TestCleanup Definido na classe 1
```

Test Name: MetodoDeTesteDois

Test Outcome: Aprovado

Standard Output

```
Executando o TestInitialize Definido na classe 1
Metodo de Teste Dois foi executado
Executando o TestCleanup Definido na classe 1
Executando o ClassCleanup Definido na classe 1
Executando o AssemblyCleanup Definido na classe 1
```

Cenário 2: Criar uma nova classe de teste ‘**TesteDeAtributos2**’, mantendo a primeira classe com dois testes que criamos e a segunda apenas com um teste.

```
[TestClass]
```

```

public class TesteDeAtributos2
{
    [AssemblyInitialize]
    public static void AssemblyInitB(TestContext context)
    {
        //Executado uma vez antes na inicialização antes da execução de todos os testes.
        Console.WriteLine("Executando o AssemblyInitialize Definido na classe 1");
    }

    [AssemblyCleanup]
    public static void AssemblyCleanupB()
    {
        //Executado uma vez após a execução de todos os testes.(Optional)
        Console.WriteLine("Executando o AssemblyCleanup Definido na classe 1");
    }

    [ClassInitialize]
    public static void TestFixtureSetupB(TestContext context)
    {
        //Executado uma vez antes da inicialização da classe de teste.
        Console.WriteLine("Executando o ClassInitialize Definido na classe 2");
    }

    [TestInitialize]
    public void SetupB()
    {
        //Executado antes de cada teste.
        Console.WriteLine("Executando o TestInitialize Definido na classe 2");
    }

    [ClassCleanup]
    public static void TestFixtureTearDownB()
    {
        //Executado uma vez após todos os testes desta classe serem executados.
        Console.WriteLine("Executando o ClassCleanup Definido na classe 2");
    }

    [TestCleanup]
    public void TearDownB()
    {
        //Executado após cada teste finalizado.
        Console.WriteLine("Executando o TestCleanup Definido na classe 2");
    }

    [TestMethod]
    public void MetodoDeTesteTres()
    {
        //Teste a ser executado
        Console.WriteLine("Metodo de Teste Tres foi executado");
    }
}

```

Ao tentarmos executar apenas esta nova classe, será exibido o seguinte erro:

MetodoDeTesteTres Source: TesteDeAtributos2.cs linha 52 MetodoDeTesteTres Message: UTA013: FrameworkQarti.Ignorar.TesteDeAtributos2: Cannot define more than one method with the AssemblyInitialize attribute inside an assembly. Elapsed time: 0:00:00	Copiar Tudo
--	-----------------------------

A definição de Assembly é válida para o projeto, mesmo estando em classes diferentes sua definição é feita apenas uma vez.

Apague os atributos de AssemblyInitialize/AssemblyCleanup definidos nesta classe ‘TesteDeAtributos2’ e execute apenas o teste Tres novamente.

O Resultado será o seguinte:

Test Name: MetodoDeTesteTres Test Outcome: Aprovado Standard Output Executando o AssemblyInitialize Definido na classe 1 Executando o ClassInitialize Definido na classe 2 Executando o TestInitialize Definido na classe 2 Metodo de Teste Tres foi executado Executando o TestCleanup Definido na classe 2 Executando o ClassCleanup Definido na classe 2 Executando o AssemblyCleanup Definido na classe 1

As duas classes não possuem nenhum tipo de relação, logo os atributos ClassInitialize/ClassCleanup e TestInitialize/TestCleanup serão executados conforme a classe aonde estão declarados.

Porém neste caso, o AssemblyInitialize/AssemblyCleanup declarado na classe ‘TesteDeAtributos1’ foi executado na ‘TesteDeAtributos2’ aonde está o teste Tres. Mesmo não executando os testes de onde o AssemblyInitialize/AssemblyCleanup está declarado, sua chamada é feita da mesma forma, e apenas uma vez independentemente do número de testes no projeto.

Cenário 3: As classes de teste herdam o TestInitialize/TestCleanup de uma classe pai

Crie uma nova classe base chamada ‘BaseTesteAtributos’ e inclua o seguinte código.

```
public class BaseTesteAtributos
{
  [TestInitialize]
  public void Setup()
  {
    //Executado antes de cada teste.
    Console.WriteLine("Executando o TestInitialize Definido na classe Pai");
  }

  [TestCleanup]
  public void TearDown()
  {
    //Executado após cada teste finalizado.
    Console.WriteLine("Executando o TestCleanup Definido na classe Pai");
  }
}
```

Mantenha as duas classes com o mesmo código, apenas faça com que herdem de '**BaseTesteAtributos**'.

Após a execução veremos o resultado

```
Test Name: MetodoDeTesteUm
Test Outcome: ✅ Aprovado
Standard Output
Executando o AssemblyInitialize Definido na classe 1
Executando o ClassInitialize Definido na classe 1
Executando o TestInitialize Definido na classe Pai
Executando o TestInitialize Definido na classe 1
Metodo de Teste Um foi executado
Executando o TestCleanup Definido na classe 1
Executando o TestCleanup Definido na classe Pai
```

```
Test Name: MetodoDeTesteDois
Test Outcome: ✅ Aprovado
Standard Output
Executando o TestInitialize Definido na classe Pai
Executando o TestInitialize Definido na classe 1
Metodo de Teste Dois foi executado
Executando o TestCleanup Definido na classe 1
Executando o TestCleanup Definido na classe Pai
Executando o ClassCleanup Definido na classe 1
Executando o ClassCleanup Definido na classe 2
Executando o AssemblyCleanup Definido na classe 1
```

```
Test Name: MetodoDeTesteTres
Test Outcome: ✅ Aprovado
Standard Output
Executando o ClassInitialize Definido na classe 2
Executando o TestInitialize Definido na classe Pai
Executando o TestInitialize Definido na classe 2
Metodo de Teste Tres foi executado
Executando o TestCleanup Definido na classe 2
Executando o TestCleanup Definido na classe Pai
```

Sendo os únicos atributos que permitem herança, o texto previsto em `TestInitialize/TestCleanup` na classe Pai foi executado em suas classes filha, junto com os que já estão definidos em cada um.

As ações definidas em cada classe não serão sobrescritas, apenas complementadas. Porém os nomes dos métodos configurados com os atributos `TestInitialize/TestCleanup` devem ser diferentes nas classes pai e filho ou não serão executados.

Obs: Note que neste caso, o `ClassCleanup` da Classe 2, só foi executado no teste Dois. Assim como já apresentado no início, na descrição dos atributos, o `ClassCleanup` não possui garantia de que seja executado logo após todos os testes de sua classe.

Lembrando que ordem de execução é aleatória e seus resultados podem estar diferentes.

Neste exemplo a ordem de execução foi a seguinte

1. `TesteUm` (Onde o `AssemblyInitialize` foi executado)
2. `TesteTres`

3. TesteDois (Onde apenas após o ClassCleanup da classe1 o ClassCleanup da classe2 foi executado. E para finalizar a execução o AssemblyCleanup)

Após realizar estes testes comente estas classes para que as saídas não sejam exibidas nos demais testes.

A definição da organização destes atributos, e do que eles executarão é muito importante.

Geralmente os atributos de inicialização, são usados para configuração dos testes como busca de dados, preparação do ambiente e criação de objetos auxiliares. E os atributos de finalização, geralmente usados para o encerramento destes objetos e restauração do ambiente após sua utilização.

Alterar essas estruturas no meio de um projeto em andamento pode exigir muito trabalho, por isso é ideal levantar todos os requisitos e necessidades e realizar sua documentação prévia para que não ocorram imprevistos e possíveis atrasos.

4.11 Centralizando o driver

Para nosso projeto usaremos uma estrutura semelhante ao *cenário 3*, onde o browser é aberto e fechado depois de cada teste.

Para organizar nosso projeto, vamos criar uma pasta ‘**Testes**’.

Mova todos os seus arquivos que contém testes para esta pasta. No caso ‘**LoginTest**’ e ‘**ContactUsTest**’

Dentro desta pasta criar um arquivo com nome ‘**BaseTest**’

Estaremos incluindo a opção de dois novos navegadores, será necessário realizar o download dos seguintes novos pacotes pelo **Gerenciador de Pacotes do NuGet**:

- **Selenium.Firefox.WebDriver**
- **Selenium.WebDriver.IEDriver** (É a versão 32 bits , não a 64)

Após o download inclua o seguinte código na nova classe ‘**BaseTest**’:

```
[TestClass]
public class BaseTest
{
    protected IWebDriver driver;
    //Não alterar a forma como TestContext está declarado, caso contrário não irá funcionar
    public TestContext TestContext { get; set; }

    [TestInitialize]
    public void TestInitialize()
    {
        driver = CriaDriver("CHROME");
        driver.Manage().Window.Maximize();
    }

    [TestCleanup]
    public void TestCleanup()
    {
        driver.Quit();
        driver.Dispose();
    }

    public IWebDriver CriaDriver(string browser)
```

```

    {
        switch (browser)
        {
            case "FIREFOX":
                driver = new FirefoxDriver();
                return driver;
            case "CHROMEHEADLESS":
                var options = new ChromeOptions();
                options.AddArgument("headless");
                driver = new ChromeDriver(options);
                return driver;
            case "CHROME":
                driver = new ChromeDriver();
                return driver;
            case "IE":
                driver = new InternetExplorerDriver();
                return driver;
            default:
                throw new AssertFailedException("Driver informado: " + browser + " Informe um
driver valido");
        }
    }
}

```

Entendo o código:

protected IWebDriver driver; : Como iremos centralizar a instância do driver nesta classe pai, vamos manter o driver como protected. (Protected = sua visibilidade é apenas para a classe pai e as classes que herdam dela)

public TestContext TestContext { get; set; } : O TestContext é uma classe que contém todas as informações do teste. Ela é setada em todo novo método de teste que é executado com novas informações referentes ao teste atual. <https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting.testcontext?view=mstest-net-1.2.0>

[TestInitialize]
public void TestInitialize()
{
 driver = CriaDriver("CHROME");
 driver.Manage().Window.Maximize();
} : Como apresentado na seção anterior, o TestInitialize executa um código antes de um método de teste ser executado. Para nosso projeto iremos usa-lo para abertura de um novo browser antes de cada teste. Porém agora iremos incluir a opção de escolher qual navegador será utilizado, para simplificar o código foi criado uma função que faz a escolha do navegador.

[TestCleanup]
public void TestCleanup()
{
 driver.Quit();
 driver.Dispose();
} : O TestCleanup responsável por executar um código depois da execução de um teste, irá fechar e encerrar o driver em execução.

public IWebDriver CriaDriver(string browser)
{
 switch (browser)
{
 case "FIREFOX":
 driver = new FirefoxDriver();
 return driver;
 case "CHROMEHEADLESS":

```

var options = new ChromeOptions();
options.AddArgument("headless");
driver = new ChromeDriver(options);
return driver;
case "CHROME":
driver = new ChromeDriver();
return driver;
case "IE":
driver = new InternetExplorerDriver();
return driver;
default:
throw new AssertFailedException("Driver informado: " + browser + " Informe um
driver valido");
}

```

} : Função que contém a escolha de navegador. A string recebida define qual navegador será retornado, caso nenhuma nome válido seja informado um erro é exibido e o teste não é iniciado. Também incluímos a opção de executar o chrome em modo headless, a execução é realizada em background sem carregar o navegador.

```

var options = new ChromeOptions();
options.AddArgument("headless");
driver = new ChromeDriver(options); : Para iniciar o chrome em modo headless foi necessário criar uma
instância de ChromeOptions(); onde podemos incluir argumentos ou parametros de inicialização. Os
demais navegadores também possuem suporte a este tipo de incialização. Mais detalhes em:
https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities
O modo headless serve para iniciar o navegador sem a exibição de sua interface gráfica na tela, o
navegador é executado em background.

```

Após criar a classe base, devemos remover as chamadas de inicialização das classes de teste já que estarão centralizadas no arquivo base. Exclua apenas o código em vermelho dos seus testes.

```

IWebDriver driver = new ChromeDriver();
driver.Manage().Window.Maximize();
//Manter o Código com ações do teste
driver.Quit();
driver.Dispose();

```

Agora faça com que as suas classes de teste herdem de BaseTest.

```

public class ContactUsTest: BaseTest

public class LoginTest : BaseTest

```

Execute os testes novamente, tente executar em todos navegadores alterando a opção de qual deseja executar em BaseTest.

4.11.1 Exercício

Dentro da classe login test, crie um novo teste para realizar um login inválido.

Passos do teste:

- Acessar o site: <http://automationpractice.com/index.php>
- Clicar no link 'Sign In'
- Preencha o campo 'Email address' com qualquer email em formato válido
- Preencha o campo 'Password' com uma senha aleatória qualquer
- Clique no botão 'Sign In'
- Valide a mensagem de falha na autenticação

Sugestão de resolução:

```
[TestMethod]
public void LoginInvalido()
{
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    driver.FindElement(By.XPath
        ("//a[@href='http://automationpractice.com/index.php?controller=my-account']")).Click();
    driver.FindElement(By.Id("email")).SendKeys("test@a.com");
    driver.FindElement(By.Id("passwd")).SendKeys("notapassword");
    driver.FindElement(By.Id("SubmitLogin")).Click();

    string textoRetornado = driver.FindElement(By.CssSelector(".alert.alert-danger")).Text;
    textoRetornado = textoRetornado.Replace("\r\n", " ");
    Assert.AreEqual("There is 1 error Authentication failed.", textoRetornado);
}
```

A resolução utilizada foi a mesma do exercício para validação da mensagem de fale conosco inválido. A estrutura apresentada na mensagem de erro é a mesma. E como agora estamos herdando o driver de BaseTest não precisamos abrir ou fechar a instância do driver dentro do teste.

4.12 Screenshots

Agora que temos a possibilidade de executar os testes de forma headless, precisamos comprovar se o teste realmente passou, e em caso de falha poderemos visualizar o ponto de erro.

Crie uma pasta ‘**Helpers**’ e inclua uma nova classe (Uma classe normal, não uma de testes) ‘**ScreenshotHelper**’ com o seguinte código.

Repare que o namespace agora é frameworkQarti.Helper ou <nomeDoSeuProjeto>.Helpers . Mantenha desta forma para facilitar a identificação do tipo de arquivo.

Dentro da pasta Helpers iremos incluir classes que contenham funções de auxílio para a execução dos testes.

```
public class ScreenshotHelper
{
    public static string TiraPrint(string nomeDaImagem, IWebDriver driver)
    {
        string screenshotsPasta =
            @"C:\Users\rafael.camargo\source\repos\FrameworkQarti\FrameworkQarti\Evidencias\";

        ITakesScreenshot camera = driver as ITakesScreenshot;
        Screenshot foto = camera.GetScreenshot();
        string caminhoDaFoto = screenshotsPasta + nomeDaImagem + ".jpeg";

        foto.SaveAsFile(caminhoDaFoto, ScreenshotImageFormat.Jpeg);

        return caminhoDaFoto;
    }
}
```

Entendendo o código:

```
string screenshotsPasta =
    @"C:\Users\rafael.camargo\source\repos\FrameworkQarti\FrameworkQarti\Evidencias\"; : Devemos apontar
uma pasta aonde as imagens serão salvas. Neste caso foi criada uma pasta ‘Evidencias’ dentro do
projeto. Para acessar o diretório do projeto clique com o botão direto no seu projeto dentro do
Gerenciador de Soluções e selecione Abrir pasta no Gerenciador de Arquivos.
```

```
ITakesScreenshot camera = driver as ITakesScreenshot; : Cria a instância da interface necessária para
a captura de fotos, necessita da instância do driver para saber quem tira a foto.
```

```

Screenshot foto = camera.GetScreenshot(); : Realiza a captura da imagem na variável foto. A captura é somente da parte visível do navegador.

string caminhoDaFoto = screenshotsPasta + nomeDaImagem + ".jpeg"; : Concatena o caminho da pasta em que o arquivo será salvo, com o nome da imagem e sua extensão. É necessário incluir a extensão, ela não é adicionada automaticamente.

foto.SaveAsFile(caminhoDaFoto, ScreenshotImageFormat.Jpeg); : Salva a imagem capturada em um destino, no caso o caminho especificado acima. Também é necessário informar o tipo de formato em que a imagem será gerada, lembrando que a extensão não é gerada automaticamente no nome do arquivo então caso altere o formato lembre-se de alterar a extensão na variável com o caminho da foto.

return caminhoDaFoto; : Estamos retornando o caminho da imagem no método, iremos utilizar para anexá-lo ao resultado do teste.

```

Para realizar a chamada deste método primeiro devemos incluir em nossos testes a referência para este namespace que contém a classe screenshotHelper:

```
using FrameworkQarti.Helper
```

Agora podemos chamar o método usando:

```
ScreenshotHelper.TiraPrint("<Nome da tela ou passo>", driver);
```

Com isso a imagem da tela no momento desta captura será salva. É recomendado usar esta captura em momentos chave da execução.

Porém apenas com esse comando apenas estamos salvando a imagem localmente, para que seja possível visualizar as imagens ao final de cada teste devemos incluir o comando:

```
TestContext.AddResultFile();
```

O TestContext foi incluído na seção anterior dentro da classe 'BaseTest' onde é inicializado. O método .AddResultFile() permite que um arquivo seja anexado ao resultado final do teste. Então para que a imagem seja adicionada a chamada deve ser feita da seguinte forma:

```
TestContext.AddResultFile( ScreenshotHelper.TiraPrint("<Nome da tela ou passo>", driver) );
```

Exemplo de como configurei em meu teste de validação de email inválido, em três momentos principais

- Quando a home é acessada
- Após os campos em Contact Us são preenchidos
- Quando a mensagem de validação é exibida

```

[TestMethod]
public void FaleConoscoEmailInvalido()
{
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    TestContext.AddResultFile(ScreenshotHelper.TiraPrint("Home", driver));
    driver.FindElement(By.Id("contact-link")).Click();
    SelectElement combo = new SelectElement(driver.FindElement(By.Id("id_contact")));
    combo.SelectByText("Customer service");
    driver.FindElement(By.Id("email")).SendKeys("emailInvalido");
    driver.FindElement(By.Id("id_order")).SendKeys("#123456789");
    driver.FindElement(By.Id("fileUpload")).SendKeys(@"C:\Users\rafael.camargo\Documents\sites.txt");
    driver.FindElement(By.Id("message")).SendKeys("Mensagem de Teste");
    TestContext.AddResultFile(ScreenshotHelper.TiraPrint("ContactUs_Preenchido", driver));
    driver.FindElement(By.Id("submitMessage")).Click();
    string textoRetornado = driver.FindElement(By.CssSelector(".alert.alert-danger")).Text;
    textoRetornado = textoRetornado.Replace("\r\n", " ");
    Assert.AreEqual("There is 1 error Invalid email address.", textoRetornado);
    TestContext.AddResultFile(ScreenshotHelper.TiraPrint("Mensagem de validacao", driver));
}

```

Resultado:

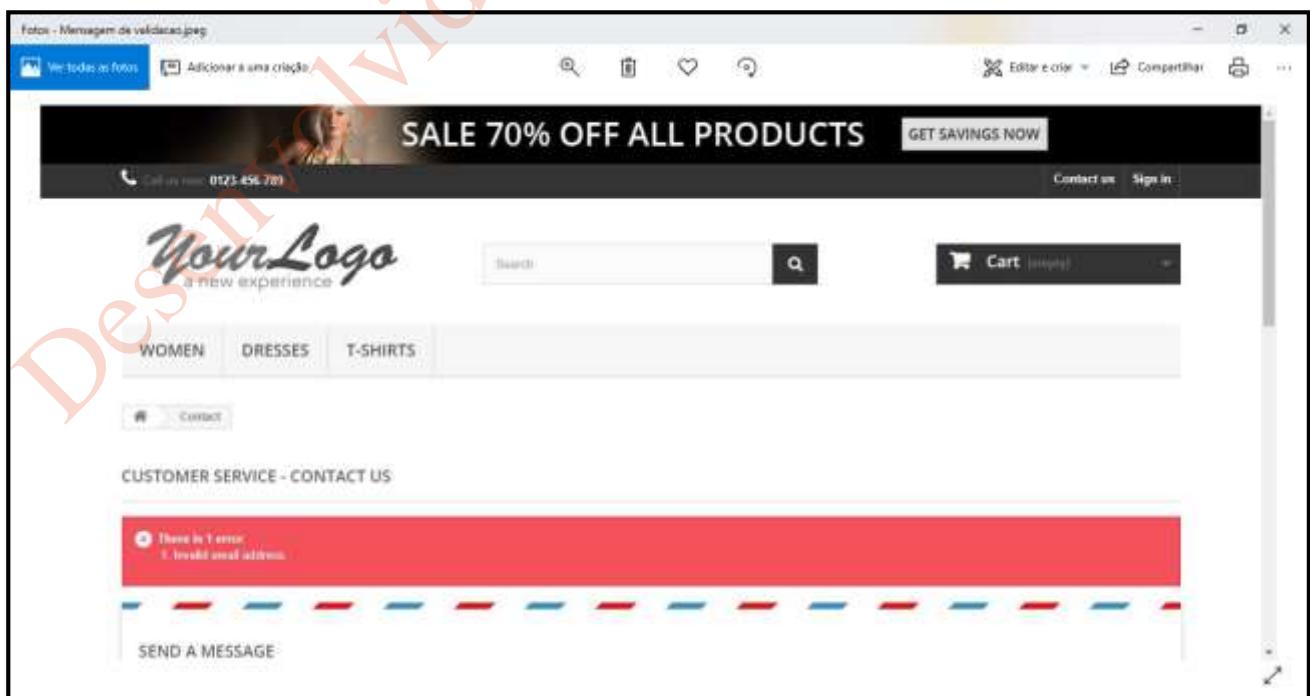
Test Name: FaleConoscoEmailInvalido

Test Outcome: ✓ Aprovado

Attachments

[Home.jpeg](#)
[ContactUs_Preenchido.jpeg](#)
[Mensagem de validacao.jpeg](#)

Ao clicar no nome da imagem, ela é aberta pelo Windows:



Agora vamos simular uma falha no teste, seguindo o exemplo usarei uma mensagem incorreta no assert:

```
Assert.AreEqual("Teste Incorreto", textoRetornado);
```

Após a execução do teste com erro é apresentado o seguinte resultado.

The screenshot shows a test run interface. At the top, it says "Test Name: FaleConoscoEmailInvalido". Below that, "Test Outcome: ✘ Com falha" (Failed). A message follows: "Message: Assert.AreEqual falhou. Esperado:<Teste Incorreto>. Real:<There is 1 error Invalid email address.>." Under "Attachments", there are two files: "Home.jpeg" and "ContactUs_Preenchido.jpeg".

A última imagem não é exibida.

Após um Assert com falha o teste é encerrado automaticamente no ponto de verificação e todo código abaixo é executado.

Para contornar este problema podemos incluir a função para tirar print no método de TestCleanup, já que este trecho é executado após a execução do teste.

Exemplo:

```
[TestCleanup]
public void TestCleanup()
{
    if (TestContext.CurrentTestOutcome == UnitTestOutcome.Failed)
    {
        TestContext.AddResultFile(ScreenshotHelper.TiraPrint("TesteComFalha", driver));
    }

    driver.Quit();
    driver.Dispose();
}
```

TestContext.CurrentTestOutcome : Esta opção contém a informação atual do resultado do teste (Failed ou Passed), como está sendo chamada no método com TestCleanup o resultado já está definido.

UnitTestOutcome : Possui os valores para o resultado do teste (*Aborted, Error, Failed, Inconclusive, InProgress, Timeout, Unknown*)

No exemplo acima escolhi apenas registrar a evidência caso o teste falhe. Caso sempre queira que uma imagem seja adicionada no final, basta remover a condição IF.

Após executar novamente veremos que no final a evidência é registrada.

Test Name:	FaleConoscoEmailInvalido
Test Outcome:	✗ Com falha
Message:	Assert.AreEqual falhou. Esperado:<Teste Incorreto>. Real:<There is 1 error Invalid email address.>.
Attachments	
Home.jpeg	
ContactUs_Preenchido.jpeg	
TesteComFalha.jpeg	

4.12.1 Exercicio:

Dentro da função que captura as imagens, existe a variável que armazena o caminho da pasta em que devem ser salvadas.

```
string screenshotsPasta =
@"C:\Users\rafael.camargo\source\repos\FrameworkQarti\FrameworkQarti\Evidencias\";
```

Por padrão temos que definir uma pasta para salvar as evidências, caso o projeto seja executado em outro computador esta pasta não existirá. Neste caso temos que criar e apontar para a pasta alterando o código com o novo caminho.

Encontre uma forma criar a pasta dinamicamente para que a execução seja feita sem necessidade de intervenção manual.

Sugestão de resolução:

Como ponto de referência, usarei o caminho da DLL do projeto, este caminho pode ser obtido pelo comando:

```
System.Reflection.Assembly.GetCallingAssembly().CodeBase;
```

Para a criação da pasta, o método ficou desta forma:

```
public static void CriaPastaEvidencias()
{
    string caminhoAssembly = System.Reflection.Assembly.GetCallingAssembly().CodeBase;
    caminhoAssembly = new Uri(caminhoAssembly).LocalPath;
    string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../..\")); 
    string caminhoFinal = novoCaminho + "Evidencias\\";
    DirectoryInfo Pasta = new DirectoryInfo(caminhoFinal);
    if (Pasta.Exists != true)
    {
        Pasta.Create();
    }
}
```

Entendendo o código:

`string caminhoAssembly = System.Reflection.Assembly.GetCallingAssembly().CodeBase;` : Obtém o caminho da DLL sendo executada atualmente. No meu caso retorna o seguinte:

`file:///C:/Users/rafael.camargo/source/repos/FrameworkQarti/FrameworkQarti/bin/Debug/FrameworkQarti.DLL`

`caminhoAssembly = new Uri(caminhoAssembly).LocalPath;` : Converte este caminho do arquivo para um formato Uri onde é possível manipula-lo. Agora o caminho fica da seguinte forma:

`C:/Users/rafael.camargo/source/repos/FrameworkQarti/FrameworkQarti/bin/Debug/FrameworkQarti.DLL`

string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../..\\")); : Agora iremos alterar o caminho, voltando três diretórios acima do atual e caindo na pasta do projeto. O caminho fica assim:

```
C:\Users\rafael.camargo\source\repos\FrameworkQarti\FrameworkQarti\
```

string caminhoFinal = novoCaminho + "Evidencias\\"; : Agora incluiremos a pasta “Evidencias” neste caminho. Atualizando para:

```
C:\Users\rafael.camargo\source\repos\FrameworkQarti\FrameworkQarti\Evidencias\
```

DirectoryInfo Pasta = new DirectoryInfo(caminhoFinal); : Cria a variável do tipo DirectoryInfo, onde poderemos saber se a pasta apontada existe.

```
if (Pasta.Exists != true)
{
    Pasta.Create();
}
```

: Se a pasta não existir, será criada.

Para a captura da imagem, usei a mesma lógica, porém o caminho final aqui é o caminho da foto.

```
public static string TiraPrint(string nomeDaImagem, IWebDriver driver)
{
    StringBuilder timeStamp = new StringBuilder(DateTime.Now.ToString());
    timeStamp.Replace("/", "_");
    timeStamp.Replace(":", "_");

    string caminhoAssembly = System.Reflection.Assembly.GetCallingAssembly().CodeBase;
    caminhoAssembly = new Uri(caminhoAssembly).LocalPath;
    string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../..\\"));
    string caminhoDaFoto = novoCaminho + "Evidencias\\" + nomeDaImagem + "_" + timeStamp +
    ".jpeg";

    ITakesScreenshot camera = driver as ITakesScreenshot;
    Screenshot foto = camera.GetScreenshot();

    foto.SaveAsFile(caminhoDaFoto, ScreenshotImageFormat.Jpeg);

    return caminhoDaFoto;
}
```

O método TiraPrint foi apenas refatorado, não sendo necessário nenhuma alteração em sua chamada.

Mas para o método de criação de pasta, a verificação de sua existência e sua criação será feita apenas no ínicio dos testes, e não antes de cada teste. Para isso podemos usar o atributo “**AssemblyInitialize**”. Dentro da classe ‘**BaseTest**’ inclua o seguinte código acima do ‘**TestInitialize**’.

```
[AssemblyInitialize]
public static void AssemblyInitialize(TestContext context)
{
    ScreenshotHelper.CriaPastaEvidencias();
}
```

Agora apague a sua pasta Evidências que foi criada manualmente em seu projeto, e o execute novamente. Verifique que o novo diretório será criado e as evidências registradas da mesma forma.

Caso queira incluir o nome do teste, utilize o `TestContext.TestName` :

```
TestContext.AddResultFile( ScreenshotHelper.TiraPrint(TestContext.TestName + "_Imagen_Final", driver)  
);
```

4.13 Page Objects

Para a organização dos projetos de automação, um padrão muito utilizado é o Page Objects Model (POM). Este padrão permite organizar a estrutura dos testes dividindo suas ações e seus elementos em classes que são abstrações das páginas de um site.

Para cada página do site, uma classe contendo seus elementos e ações serão criados, também iremos criar DSLs específicas para a chamada das ações correspondentes.

Com os elementos centralizados neste repositório temos maior facilidade em sua manutenção e legibilidade.

O padrão sugere que toda ação que leva até uma nova página, deve ter como retorno a classe da nova página. Caso a ação se mantenha na mesma página deve retornar a própria classe. Caso a ação não tenha mais pontos de saída não deve retornar nada, ou retornar apenas o que seja relevante ao seu contexto, como um texto ou valor obtido no elemento por exemplo.

As pages serão responsáveis apenas por manipular e interagir com os elementos da tela, as asserções são de responsabilidade do teste e não devem ser feitas aqui. As pages apenas retornarão os insumos necessários para realizar as asserções dentro da classe de teste.

Este padrão de projeto foi descrito por Martin Fowler, mais detalhes em seu artigo:

<http://www.pedromendes.com.br/2013/10/28/traducao-do-artigo-sobre-pageobjects/>

4.13.1 Notação FindsBy do Page Factory

A nova notação para identificação de elementos através do **PageFactory** possui a seguinte estrutura:

```
[FindsBy(How = How.<TipoDoIdentificador>, Using = <Identificador>)]  
private IWebElement NomeDoElemento { get; set; }
```

Em comparação com a forma normal:

```
private IWebElement NomeDoElemento = driver.FindElement(By.<TipoDoIdentificador>(<Identificador>))
```

Para nosso exemplo iremos usar a nova notação, mas também é possível optar por utilizar a forma de identificação normal atribuindo os elementos a uma variável.

4.13.2 Nomenclatura dos Elementos

Seguindo o princípio do encapsulamento, os elementos devem ser mantidos como private, para acessá-los iremos criar métodos equivalentes aos getters e setters em uma classe comum. Estes poderão retornar apenas o elemento, ou um conjunto de ações do elemento.

Para a Nomenclatura dos Elementos deve ser seguida a seguinte padronização:

- Não utilizar acentuação, caracteres especiais e números nos nomes dos objetos
- Não utilizar atributos dos objetos como ID nos nomes
- Não utilizar nomes duplicados ou ambíguos
- Utilizar CamelCase.

- Para facilitar a identificação do tipo de elemento, utilizar prefixo sugerido abaixo seguido do nome do elemento.

Tipo de Componente	Prefixo	Exemplo
Button	btn	btnEntrar
Calendar	cld	cldDataNascimento
Cell	cell	cellEmpresa
Chart	chr	chrGanhos
CheckBox	chk	chkConfirmar
Column	clm	clmNomes
ComboBox	ccb	ccbEstado
Date picker	dtp	dtpData
Error Message	err	errLogin
Footer	ft	ftCliente
Form	form	formCustomer
Frame	fra	fraPrincipal
Graph	gra	graValores
GroupBox	grp	grpSexo
Header	hd	hdNaoLogado
Image	img	imgCachorro
Label	lbl	lblSobrenome
Link	lnk	lnkRecuperarSenha
ListBox	lst	lstEstados
Loading Icon	ldg	ldgCarregando
Menu	menu	menuCadastro
Message	msg	msgSucesso
Modal	mod	modDesejaContinuar
Panel	pnl	pnlGrupo
PlaceHolder	plh	plhNome
ProgressBar	prg	prgCarregando
RadioButton	rad	radMasculino
RichTextBox	rtf	rtfDescricaoOcorrido
Row	row	rowCustomer
Scrollbar	scb	scbScroll
Table	tbl	tblPrecos
TextBox	txt	txtNome
Tooltip	ttip	ttipAjuda

4.13.3 Convertendo os testes para o padrão POM

Vamos criar uma pasta chamada '**PageObjects**'.

Todas as classes que forem incluídas aqui devem possuir o sufixo '**Pages**' no nome para facilitar a identificação, além de estar no namespace `<nomeDoProjeto>.PageObjects`

Crie uma nova classe chamada de '**BasePage**' e inclua o código abaixo nela

```
public class BasePage
{
    protected IWebDriver driver;

    public BasePage(IWebDriver driver)
    {
        this.driver = driver;
        PageFactory.InitElements(driver, this);
    }
}
```

Esta classe servirá como a classe Pai para as demais classes de page objects que criarmos.

Após importar as dependências necessários, a classe PageFactory terá uma marcação, ao deixar o mouse acima dela será exibida a seguinte mensagem:



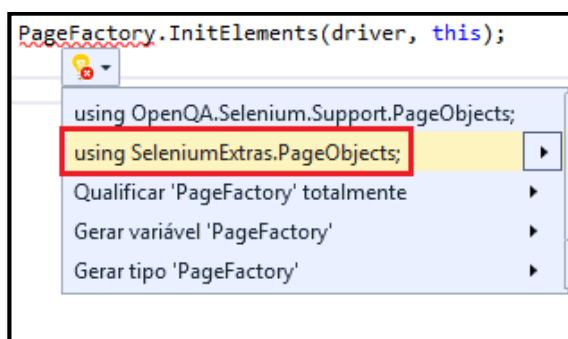
A biblioteca importada pelo PageFactory é a OpenQA.Selenium.Support.PageObjects. A mensagem informa que esta implementação já está obsoleta e a nova implementação está em outra biblioteca. Então devemos remover o 'using OpenQA.Selenium.Support.PageObjects' do início do código. Para corrigir o erro vamos realizar o download do pacote:

- **DotNetSeleniumExtras.PageObjects**

Após o download, teremos como sugestão as opções OpenQA.Selenium.Support.PageObjects e SeleniumExtras.PageObjs.

Selecione a **SeleniumExtras.PageObjs**

Importante: Para as demais classes de pages de forem criadas o import utilizado deve ser sempre o **SeleniumExtras.PageObjs** ! Caso use a primeira opção o código não funcionará corretamente. Este import é requerido apenas para o uso do PageFactory, que inicializa os elementos através da notação FindsBy. Se em algum projeto futuro não for utilizar o PageFactory este import não é necessário.



Voltando ao código que incluímos, a explicação de cada comando:

```
protected IWebDriver driver; : Como sempre, para interagir com os elementos devemos passar a instância do driver em uso. Criaremos esta variável para armazenar o valor recebido nesta classe base e manter sua visibilidade apenas para suas classes filhas.
```

```
public BasePage(IWebDriver driver)
{
    this.driver = driver;
    PageFactory.InitElements(driver, this);
```

} : Construtor que recebe a instância do driver e a armazena na variável criada acima.

```
PageFactory.InitElements(driver, this); : Este comando serve para inicializar os elementos e permite que os elementos sejam identificados sem o uso do comando driver.FindElement(By). Usaremos uma notação específica para a identificação dos elementos. Como parâmetros tem o driver, e a classe a ser inicializada.
```

Caso a forma com a notação não seja utilizada, não é necessário inicializar o PageFactory. Mas usaremos o PageFactory para fim de aprendizado.

Por hora não incluiremos nenhum elemento nesta basePage, como sugestão de uso podemos incluir apenas elementos que são comuns a todas as páginas, como por exemplo mensagens de alerta genéricas a todo o site ou ícones de carregamento.

Feito isso, vamos mudar nosso primeiro teste de login para este padrão.

Antes vamos entender o que precisa ser feito. O Teste de login é o seguinte:

```
[TestMethod]
public void LoginValido()
{
    //Navega ate a url
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    //Já na Home não logada, clica no link que direciona para a página de login

    driver.FindElement(By.XPath("//a[@href='http://automationpractice.com/index.php?controller=my-
account']")).Click();
    //Na Página de login, preenche email e senha e clica no botão
    driver.FindElement(By.Id("email")).SendKeys("teste@testing.com");
    driver.FindElement(By.Id("passwd")).SendKeys("1234qwer");
    driver.FindElement(By.Id("SubmitLogin")).Click();
    //Acessa a Home logada e verifica um elemento válido
    Assert.IsNotNull(driver.FindElement(By.CssSelector(".account").Text));
}
```

Iremos passar por três páginas distintas, então devemos criar três classes.

Primeiro vamos criar página da Home Não Logada, então criaremos a classe HomeNaoLogadaPage

```
public class HomeNaoLogadaPage : BasePage
{
    public HomeNaoLogadaPage(IWebDriver driver) : base(driver)
    {

    }

    [FindsBy(How = How.XPath, Using =
"//a[@href='http://automationpractice.com/index.php?controller=my-account'])]
```

```

private IWebElement lnkSignIn { get; set; }

public LoginPage AcessarPaginaLogin()
{
    lnkSignIn.Click();
    return new LoginPage (driver);
}

}

```

O código representa o seguinte:

```

public class HomeNaoLogadaPage : BasePage : Devemos realizar a herança entre a BasePage e nossa
classe para obter o driver que foi definido lá.

public HomeNaoLogadaPage(IWebDriver driver) : base(driver)
{
} : Um construtor que receber a instância do driver e herda o construtor definido na BasePage, este
por sua vez irá receber a instância do driver e inicializar os elementos

[FindsBy(How=How.XPath,Using="//a[@href='http://automationpractice.com/index.php?controller=my-
account'])"]
private IWebElement lnkSignIn { get; set; } : A notação que usaremos para identificar os elementos
utilizando o PageFactory.

public LoginPage AcessarPaginaLogin()
{
    lnkSignIn.Click();
    return new LoginPage(driver);
}: Não acessaremos diretamente o elemento, acessaremos um método que já realiza uma ação com o
elemento. Neste caso clicaremos no link Sign In, que irá direcionar para uma nova página, a
LoginPage. Como ainda não criamos esta nova página um erro ficará destacado, vamos criar a LoginPage
abaixo.

```

Criando a LoginPage, sempre dentro da pasta PageObjects e com o namespace equivalente.

```

public class LoginPage : BasePage
{
    public LoginPage(IWebDriver driver) : base(driver)
    {

    }

    [FindsBy(How = How.Id, Using = "email")]
    private IWebElement txtEmail { get; set; }

    [FindsBy(How = How.Id, Using = "passwd")]
    private IWebElement txtSenha { get; set; }

    [FindsBy(How = How.Id, Using = "SubmitLogin")]
    private IWebElement btnSignIn { get; set; }

    public LoginPage PreencherEmail()
    {
        txtEmail.SendKeys("teste@testing.com");
        return this;
    }

    public LoginPage PreencherSenha()
    {
        txtSenha.SendKeys("1234qwer");
        return this;
    }
}

```

```

    public HomeLogadaPage ClicarSignIn()
    {
        btnSignIn.Click();
        return new HomeLogadaPage(driver);
    }
}

```

O código representa o seguinte:

```

[FindsBy(How = How.Id, Using = "email")]
private IWebElement txtEmail { get; set; }

[FindsBy(How = How.Id, Using = "passwd")]
private IWebElement txtSenha { get; set; }

[FindsBy(How = How.Id, Using = "SubmitLogin")]
private IWebElement btnSignIn { get; set; } : Primeiro devemos armazenar os objetos utilizados na
página de login seguindo o mesmo padrão.

public LoginPage PreencherEmail()
{
    txtEmail.SendKeys("teste@testing.com");
    return this;
}

public LoginPage PreencherSenha()
{
    txtSenha.SendKeys("1234qwer");
    return this;
}

public HomeLogadaPage ClicarSignIn()
{
    btnSignIn.Click();
    return new HomeLogadaPage(driver);
} : Agora devemos criar os métodos que nos dão acesso a cada elemento, como iremos usar apenas uma
ação em cada objeto teremos os passos que equivalem ao preenchimento do email, preenchimento da
senha. Repare nos passos de preenchimento de email e senha o retorno é a própria classe de LoginPage
através do 'this'. O retorno é feito baseado no fluxo em que cada ação toma, por exemplo o
preenchimento do login e senha são feitos sempre na LoginPage, já o clique no botão que efetua o
login deve nos direcionar para a próxima página, assim retornado a nova instância.

```

Feita a página de login, resta a área logada para a validação. Crie a página HomeLogadaPage como exemplo abaixo.

```

public class HomeLogadaPage : BasePage
{
    public HomeLogadaPage(IWebDriver driver) : base(driver)
    {

    }

    [FindsBy(How = How.XPath, Using = "./h1[@class='page-heading']]")
    private IWebElement lblMyAccount{ get; set; }

    public string VerificaHomeLogada()
    {
        return lblMyAccount.Text;
    }
}

```

Aqui seguimos os mesmo padrão, o único objeto que temos no momento conforme o exemplo é o link com o nome do usuário para validação. Então o método VerificaHomeLogada apenas irá retornar este elemento, mas não fará sua validação efetiva. Iremos retornar o elemento justamente para realizar o assert em nossa classe de teste.

Após a criação das pages o teste agora ficará da seguinte forma.

```
[TestMethod]
public void LoginValido()
{
    //Navega ate a url
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    //Realiza a navegação encadeada
    Assert.IsNotNull(new HomeNaoLogadaPage(driver)
        .AcessarLogin()
        .PreencherEmail()
        .PreencherSenha()
        .ClicarSignIn()
        .VerificaHomeLogada());
}
```

Repare que agora em comparação com o teste original, temos um código mais legível e enxuto. Agora podemos reaproveitar as ações e elementos que já foram criados nas pages.

4.13.4 Abordagens para o POM

4.13.4.1 Estrutural

A abordagem estrutural foi que implementamos neste exemplo do login, temos basicamente uma saída a ação de cada elemento, permitindo uma maior flexibilidade para escolher quais ações iremos utilizar. Este encadeamento de ações no Page objects é conhecido como **Fluent Page Objects**.

```
new HomeNaoLogadaPage(driver)
    .AcessarLogin()
    .PreencherEmail()
    .PreencherSenha()
    .ClicarSignIn()
    .VerificaHomeLogada();
```

Vantagem : Mais flexível

Desvantagem: Exige mais linhas de código devido à criação de mais métodos de retorno para os elementos

4.13.4.2 Funcional

Na abordagem funcional, não representaremos os métodos a cada ação, a abstração será do conjunto de ações que os elementos realizam, neste caso o login poderia ser agrupado em uma função 'Logar', sem a necessidade de criar um método de retorno para cada elemento, como no exemplo abaixo. Em um preenchimento de cadastro ou formulário muito extenso aonde temos um número muito maior de campos, poderíamos quebrar o preenchimento das diversas seções em funções (Dados pessoais, endereço, dados bancários, etc...).

```
new HomeNaoLogadaPage(driver)
    .AcessarLogin()
    .Logar()
    .VerificaHomeLogada();
```

A implementação do método Logar dentro de LoginPage seria a seguinte:

```
public HomeLogadaPage Logar()
{
    txtEmail.SendKeys("teste@testing.com");
    txtSenha.SendKeys("1234qwer");
    btnSignIn.Click();
    return new HomeLogadaPage(driver);
}
```

Veja que aqui as funções PreencherEmail(), PreencherSenha() e ClicarSignIn() não são criadas, usamos apenas a função Logar() que possui a chamada direta aos elementos dentro de si.

Vantagem: Menos código, mais abstrações para realização de várias ações

Desvantagem: No longo prazo pode possuir métodos duplicados ou muito similares

4.13.4.3 Estruturado + Funcional

Também é possível combinar as duas abordagens da seguinte forma, combinando os métodos criados de forma estrutural dentro da chamada da estrutura funcional como no exemplo da LoginPage abaixo:

```
public LoginPage PreencherEmail()
{
    txtEmail.SendKeys("teste@testing.com");
    return this;
}

public LoginPage PreencherSenha()
{
    txtSenha.SendKeys("1234qwer");
    return this;
}

public HomeLogadaPage ClicarSignIn()
{
    btnSignIn.Click();
    return new HomeLogadaPage(driver);
}

public HomeLogadaPage Logar()
{
    PreencherEmail();
    PreencherSenha();
    ClicarSignIn();
    return new HomeLogadaPage(driver);
}
```

Aqui criamos as funções PreencherEmail(), PreencherSenha() e ClicarSignIn() e Logar(), sendo que o Logar(), faz uso das outras específicas já criadas. Isso permite tanto o uso individual de cada uma, como o uso generalizado para realização do Login.

4.13.5 Retorno de novas páginas, questão de contexto

4.13.5.1 Primeira forma: Sempre utilizar o retorno de novas páginas

Os exemplos realizados acima foram feitos seguindo a risca a orientação citada por Martin Fowler em seu artigo sobre Page Objects (*Disponível em: <https://martinfowler.com/bliki/PageObject.html>*):

Similarly if you navigate to another page, the initial page object should return another page object for the new page [2]. In general page object operations should return fundamental types (strings, dates) or other page objects.

Ou seja para ação em que mudamos de página, retornamos a nova página e seu fluxo esperado. Porém este texto possui uma nota de rodapé:

2: Having page objects be responsible for creating other page objects in response to things like navigation is common advice. However some practitioners prefer that page objects return some generic browser context, and the tests control which page objects to build on top of that context based on the flow of the test (particularly conditional flows). Their preference is based on the fact that the test script knows what pages are expected next and this knowledge doesn't need to be duplicated in the page objects themselves. They increase their preference when using statically typed languages which usually reveal page navigations in type signatures.

Em resumo, o retorno de outras pages é um conselho comum, porém nem sempre seguido já que força o fluxo a seguir uma direção específica. Em nosso exemplo o método `ClicarSignIn()` retorna uma nova Page no fluxo que é esperado para um login válido, a `HomeLogadaPage`.

Pensando nos fluxos da aplicação e seus testes, podemos ter o caso em que inserimos um login inválido, este não retornaria a próxima página, e sim a própria `loginPage` para poder realizar a validação da mensagem de erro. E para fazer isso, precisaríamos de outro método em que após o clique, tivesse como retorno a própria Page. Acabaríamos tendo basicamente dois métodos que fazem a mesma ação, mas com retornos diferentes para fluxos diferentes.

Ainda poderia existir o caso em que após o login, o usuário é迫使ido a ir para a página e renovar sua senha, tendo um terceiro método similar. Optando por usar a abordagem funcional também seria preciso que fossem criados três métodos diferentes com ações similares, por exemplo: `LoginVálido()`, `LoginInválido()`, e `LoginAtualizaSenha()`.

Ou seja, preciso de N métodos para um comando que pode me levar a N páginas diferentes. Em um sistema aonde o fluxo possui uma grande variação o trabalho para codificar estes fluxos vai ser maior.

Fizemos o exemplo da forma a risca apenas para o entendimento do conceito geral sobre Page Objects. Nos casos abaixo irei mostrar como ficariam as pages sem os retornos específicos.

4.13.5.2 Segunda forma: Retornar apenas os métodos das próprias página

Exemplo sem retorno de novas páginas, o retorno só é usado para retornar ações da própria página, ainda possibilitando o uso do Fluent Page Objects. Não será utilizado o retorno para novas páginas.

Apesar de não estarmos retornando páginas, o retorno de elementos, strings, ou outros tipos que sejam insumos ainda são utilizados normalmente para a validação do teste. Estamos limitando apenas a chamada da navegação nestes casos.

Na `HomeNaoLogadaPage` o `AcessarLogin()` não retorna a próxima Page:

```
public HomeNaoLogadaPage AcessarLogin()
{
    lnkSignIn.Click();
    return this;
}
```

Na `Login Page`, o método `Logar()` também não retorna a próxima Page. Porém ainda pode retornar a mesma página, que ainda possui um fluxo em que ela permaneça aqui. As demais funções `PreencherEmail()`, `PreencherSenha()` e `ClicarSignIn()` já retornavam a própria Page, então permanecem da mesma forma.

```

public LoginPage Logar()
{
    PreencherEmail();
    PreencherSenha();
    ClicarSignIn();
    return this;
}

```

Não temos alterações na HomeLogadaPage já que o fluxo do nosso exemplo acaba aqui.

E nos testes a chamada ficaria da seguinte forma:

```

HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
homenaoLogadaPage.AcessarLogin();

```

//Forma Estrutural

```

LoginPage loginPage = new LoginPage(driver);

```

```

loginPage.PreencherEmail()
    .PreencherSenha()
    .ClicarSignIn();

```

//Ou da forma funcional

```

//loginPage.Logar();

```

```

HomeLogadaPage homeLogada = new HomeLogadaPage(driver);
Assert.IsNotNull(homeLogada.VerificaHomeLogada());

```

Comparado ao primeiro exemplo, este modelo é bem mais verboso. As chamadas em Fluent Style se mantêm, porém apenas para as ações da mesma página (independente de usar forma estrutural ou funcional).

Outro ponto que muda é que quando mudamos de página, temos que criar a nova instância desta página já que não temos mais os retornos específicos. O ganho neste ponto é que agora quando realizo o login, não sou forçado a usar alguma função específica, posso chamar a instância da nova página e caso permaneça na mesma, como também posso dar continuidade ao método da mesma página.

4.13.5.3 Terceira forma: Não retornar nenhuma página

Neste caso, não retornaremos nenhuma page, os métodos são restritos a executarem o comando.

Na HomeNaoLogadaPage o AcessarLogin() não retorna nenhuma page:

```

public void AcessarLogin()
{
    lnkSignIn.Click();
}

```

Na Login Page, todos os métodos também não irão ter retorno algum.

```

public void PreencherEmail()
{
    txtEmail.SendKeys("teste@testing.com");
}

public void PreencherSenha()
{
    txtSenha.SendKeys("1234qwer");
}

public void ClicarSignIn()
{
}

```

```

        btnSignIn.Click();
    }

    public void Logar()
    {
        PreencherEmail();
        PreencherSenha();
        ClicarSignIn();
    }
}

```

O método da `VerificaHomeLogada()` da `homeLogadaPage()` permanece da mesma forma pois seu retorno é de um elemento para a verificação do teste.

A chamada do teste fica da seguinte forma:

```
HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
homenaoLogadaPage.AcessarLogin();
```

```
//Forma Estrutural
LoginPage loginPage = new LoginPage(driver);
loginPage.PreencherEmail();
loginPage.PreencherSenha();
loginPage.ClicarSignIn();
```

```
//Ou da forma funcional
//loginPage.Logar();
```

```
HomeLogadaPage homeLogada = new HomeLogadaPage(driver);
Assert.IsNotNull(homeLogada.VerificaHomeLogada());
```

Como não existe mais nenhuma relação entre as pages, temos que chamar cada método a partir da instância de classe criada, não sendo mais possível usar o fluent style. O ponto em não usar fluent style aqui, é que agora temos a possibilidade de incluir algum código entre as chamadas já que nenhuma delas está encadeada.

4.13.5.4 Qual forma utilizar?

Depende.

Vimos que o cada caso possui seus prós e contras:

Primeira forma: Sempre utilizar o retorno de novas páginas

- Prós: Permite um código de teste menor e mais legível
- Contras: Caso o fluxo possua muitas variações de tela, a codificação de cada método e suas saídas pode ser muito trabalhosa e ter uma manutenção proporcional.

Segunda forma: Retornar apenas os métodos das próprias páginas

- Prós: Possui um grau de flexibilidade maior com relação a navegação sem forçar métodos com fluxo direcionado e menos métodos com funções semelhantes
- Contras: Exige mais código nos testes em relação à primeira forma, criando mais instâncias de páginas

Terceira forma: Não retornar nenhuma página

- Prós: Permite incluir código entre as instruções de página
- Contras: Não faz uso o de fluent style, necessitando sempre declarar a instância no código e tem a leitura mais difícil

A avaliação das necessidades do projeto é que vai definir qual método se encaixa melhor. Em linhas gerais, o primeiro caso pode ser aplicado a projetos menores e com pouca variação de fluxos. O segundo já é uma opção mais balanceada e pode ser aplicada a projetos maiores. O terceiro pode ser usado quando é necessário ter algum tipo de codificação entre os comandos (como manter a captura das evidências a nível de teste), ou também pode ser usado dentro de um framework com BDD, já que os métodos ficarão separados em frases de qualquer maneira, não necessitando de manter o seu encadeamento.

Outros frameworks possuem outras formas de realizar a estrutura de Page objects, para conhecimento inicial estas três já dão uma base para o entendimento e uso da idéia geral sobre o uso do POM.

Como sugestão para nosso projeto, irei utilizar o segundo caso que permite o uso de fluent styles e não nos deixa preso a codificar as repetições. Caso queira utilizar as outras opções faça os ajustes necessários para se adequar, o importante é manter um padrão em seu projeto.

4.13.6 Exercício

Converta as demais classes de teste de seu projeto para o formato Page objects

Sugestão de resolução para os testes FaleConoscoValido e FaleConoscoEmailInvalido:

Dentro da HomeNaoLogadaPage, incluir o elemento e o método com a ação para clicar no link que direciona para a página ContactUs :

```
[FindsBy(How = How.XPath, Using =
"//a[@href='http://automationpractice.com/index.php?controller=contact'])"]
private IWebElement lnkContactUs { get; set; }

public HomeNaoLogadaPage AcessarContactUs()
{
    lnkContactUs.Click();
    return this;
}
```

Criar a classe ContactUsPage, esta ficará com os elementos e métodos existentes na página Contact Us do site.

Obs: A mensagem de sucesso é exibida em uma nova página. Mas para manter o mesmo contexto e não criar uma nova página que terá apenas um elemento, podemos assumir neste caso que a mensagem de sucesso também pertence à mesma página.

```
namespace FrameworkQarti.PageObjects
{
    public class ContactUsPage : BasePage
    {
        public ContactUsPage(IWebDriver driver) : base(driver)
        {}

        [FindsBy(How = How.Id, Using = "id_contact")]
        private IWebElement cbbAssunto { get; set; }

        [FindsBy(How = How.Id, Using = "email")]
        private IWebElement txtEmail { get; set; }

        [FindsBy(How = How.Id, Using = "id_order")]
        private IWebElement txtNumeroPedido { get; set; }
    }
}
```

```

[FindsBy(How = How.Id, Using = "fileUpload")]
private IWebElement btnEnviarArquivo { get; set; }

[FindsBy(How = How.Id, Using = "message")]
private IWebElement txtMensagem { get; set; }

[FindsBy(How = How.Id, Using = "submitMessage")]
private IWebElement btnEnviarMensagem { get; set; }

[FindsBy(How = How.CssSelector, Using = ".alert.alert-success")]
private IWebElement msgEnviadoComSucesso { get; set; }

[FindsBy(How = How.CssSelector, Using = ".alert.alert-danger")]
private IWebElement msgAlertaErro { get; set; }

public ContactUsPage PreencherComboAssunto(string texto)
{
    SelectElement Selec_Event = new SelectElement(cbbAssunto);
    Selec_Event.SelectByText("Customer service");
    return this;
}

public ContactUsPage PreencherEmail(string texto)
{
    txtEmail.SendKeys(texto);
    return this;
}

public ContactUsPage PreencherNumeroPedido(string texto)
{
    txtNumeroPedido.SendKeys(texto);
    return this;
}

public ContactUsPage EnviarArquivoDeUpload(string caminho)
{
    btnEnviarArquivo.SendKeys(caminho);
    return this;
}

public ContactUsPage PreencherMensagem(string texto)
{
    txtMensagem.SendKeys(texto);
    return this;
}

public ContactUsPage ClicarEnviar()
{
    btnEnviarMensagem.Click();
    return this;
}

public ContactUsPage PreencherSolicitacao(string assunto, string email, string numPedido,
string caminhoArquivo, string mensagem)
{
    PreencherComboAssunto(assunto);
    PreencherEmail(email);
    PreencherNumeroPedido(numPedido);
    EnviarArquivoDeUpload(caminhoArquivo);
    PreencherMensagem(mensagem);
    ClicarEnviar();
    return this;
}

```

```

    }

    public string VerificarMensagemSucesso()
    {
        return msgEnviadoComSucesso.Text;
    }

    public string VerificarMensagemAlertaErro()
    {
        return msgAlertaErro.Text.Replace("\r\n", " ");
    }
}
}

```

Repare também, que não estou mais preenchendo os valores de entrada diretamente pela página de Page objects, apenas recebo os parâmetros através dos métodos. A classe de teste é a que irá informar os valores a serem preenchidos.

A classe de teste ContactUsTest ficou com os testes da seguinte forma:

```

[TestMethod]
public void FaleConoscoValido()
{
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");

    HomeNaoLogadaPage homePage = new HomeNaoLogadaPage(driver);
    homePage.AcessarContactUs();

    ContactUsPage contactPage = new ContactUsPage(driver);

    //Forma Estruturada
    string textoRetornado =
        contactPage.PreencherComboAssunto("Customer service")
            .PreencherEmail("teste@teste.com")
            .PreencherNumeroPedido("12346579")
            .EnviarArquivoDeUpload(@"C:\Dev\teste.txt")
            .PreencherMensagem("Mensagem de Teste")
            .ClicarEnviar()
            .VerificarMensagemSucesso();

    //Forma Funcional
    //string textoRetornado =
    //    contactPage.PreencherSolicitacao("Customer service", "teste@teste.com", "12346579",
    @"C:\Dev\teste.txt", "Mensagem de Teste")
    //        .VerificarMensagemSucesso();

    Assert.AreEqual("Your message has been successfully sent to our team.", textoRetornado);
}

[TestMethod]
public void FaleConoscoEmailInvalido()
{
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    HomeNaoLogadaPage homePage = new HomeNaoLogadaPage(driver);
    homePage.AcessarContactUs();

    ContactUsPage contactPage = new ContactUsPage(driver);

    //Forma Estruturada
    string textoRetornado =
        contactPage.PreencherComboAssunto("Customer service")

```

```

        .PreencherEmail("emailinvalido")
        .PreencherNumeroPedido("12346579")
        .EnviarArquivoDeUpload(@"C:\Dev\teste.txt")
        .PreencherMensagem("Mensagem de Teste")
        .ClicarEnviar()
        .VerificarMensagemAlertaErro();

    //Forma Funcional
    //string textoRetornado =
    //contactPage.PreencherSolicitacao("Customer service", "emailinvalido", "12346579",
    @"C:\Dev\teste.txt", "Mensagem de Teste")
    //           .VerificarMensagemAlertaErro();

    Assert.AreEqual("There is 1 error Invalid email address.", textoRetornado);
}

```

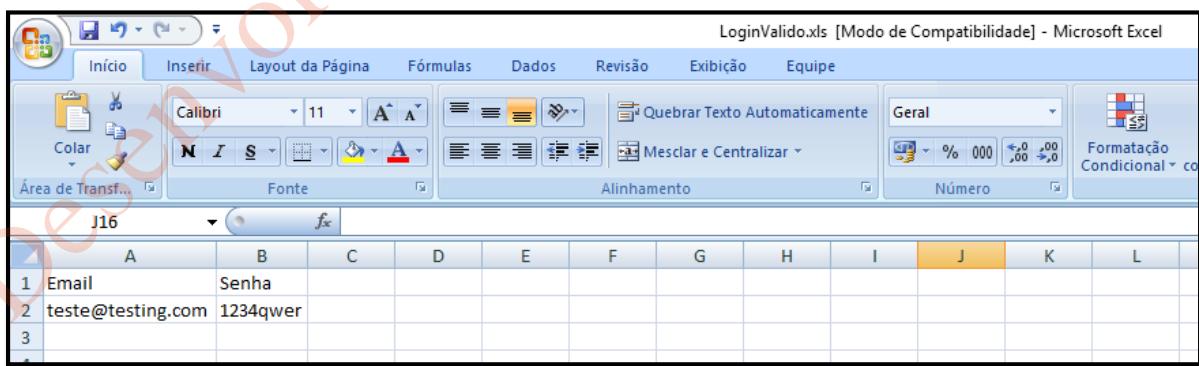
4.14 Lendo a massa de dados pelo Excel

Até o momento estamos preenchendo as massas de dados necessárias diretamente pelo código, mas isso na realidade é uma má prática. Imagine que a massa de login que estamos utilizando foi cancelada, para que não seja necessário acessar o código e realizar sua alteração, podemos manter a massa de dados em um arquivo externo. Para nosso exemplo usaremos um arquivo xls do Excel, mas este arquivo poderia ser um csv, json ou mesmo um banco de dados por exemplo. Assim caso alguém que não conheça o código da automação necessite atualizar um dado, pode realizar a alteração em um arquivo independente do código.

A forma que irei exibir serve apenas para ser utilizar neste tipo de teste usando o MsTest V1. Caso futuramente queira ler em outro framework como MsTestV2, ou NUnit, este exemplo não servirá. Na internet é possível encontrar classes prontas que fazem a leitura de arquivos excel, dependendo da forma que foi feita será necessário ajustar para o framework de testes que você está utilizando.

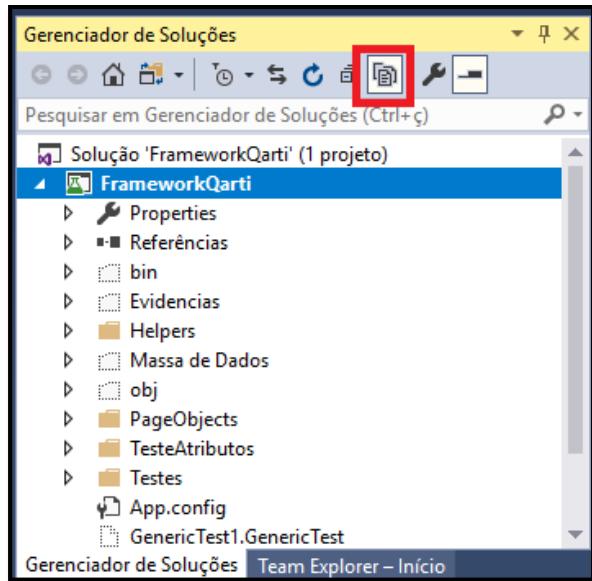
Vamos usar o teste de login válido como exemplo.

Primeiro crie dentro do projeto (e não da solução) uma pasta com o nome '**Massa de Dados**', e dentro dela um arquivo **xls** com o nome do teste correspondente. Dentro do arquivo devemos incluir as colunas que contém os respectivos dados utilizados no teste, neste caso uma coluna **Email** e outra **Senha**. Na linha abaixo preencha com os dados.

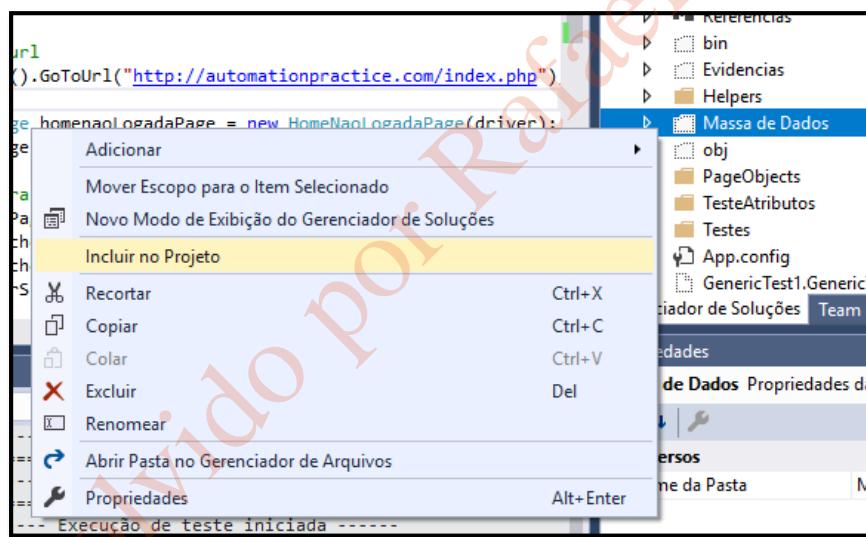


	Email	Senha
1	teste@testing.com	1234qwer
2		
3		

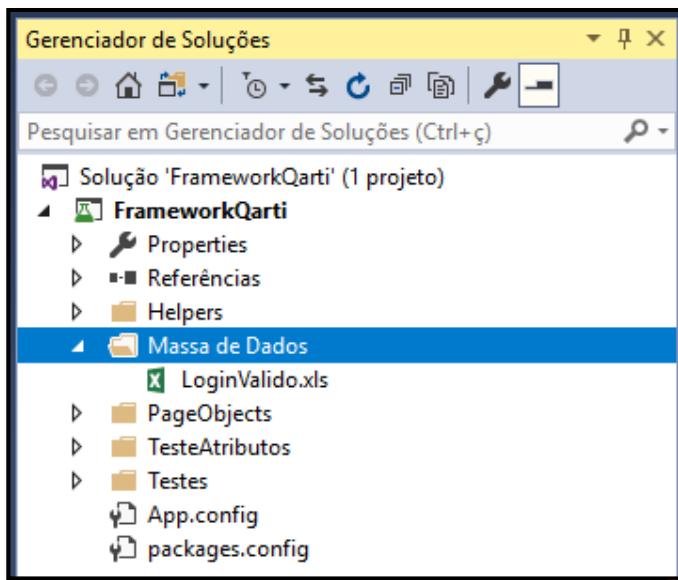
Para importar esta pasta dentro de nosso projeto, acesse o Gerenciador de Soluções e no Visual Studio, clique em projeto e procure pelo ícone '**Mostrar Todos os Arquivos**':



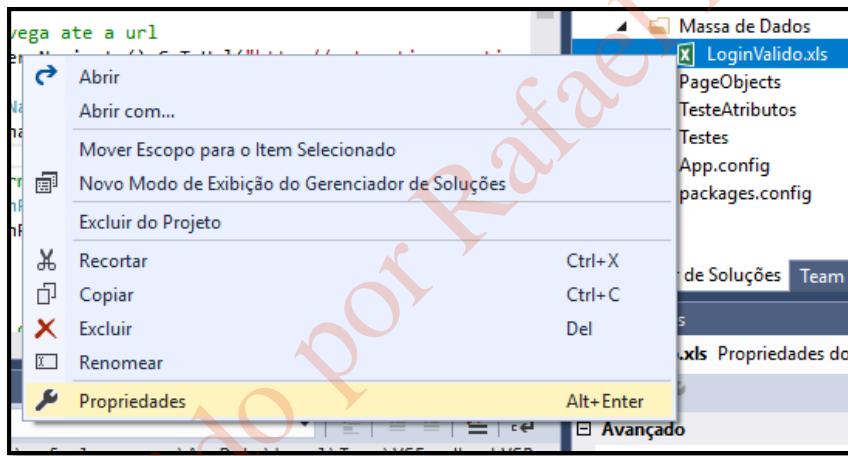
Ao clicar nele, todas as pastas existentes em nosso projeto serão exibidas. As que não estão incluídas estarão com seus desenhos pontilhados. Para incluir alguma pasta, clique nela com o botão direito, e selecione ‘Incluir no Projeto’.



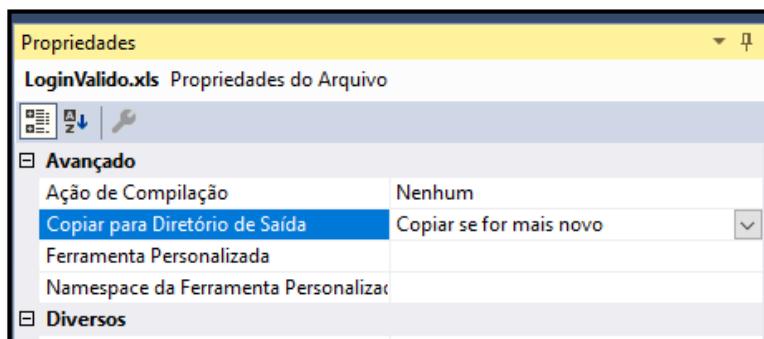
Após isso, a pasta deve ser exibida junto com o arquivo dentro dela. Sempre que um novo arquivo for adicionado através do Windows, repita os passos acima para incluí-lo no projeto. Apenas por questão de visibilidade, clique novamente no ícone ‘Mostrar Todos os Arquivos’ para deixar de exibir as pastas ocultas pelo projeto.



Depois de incluir o arquivo xls no projeto, clique com o botão direito acima do mesmo, e selecione 'Propriedades'



Na guia propriedades, em '**Copiar para Diretório de Saída**', selecione a opção '**Copiar se for mais novo**'. Isto fará com que o arquivo seja copiado para a saída junto com as demais dlls utilizadas. A opção '**Copiar Sempre**' também é válida, a diferença que esta sempre irá substituir o arquivo, enquanto a outra só irá mover caso o arquivo tenha sofrido alguma alteração.



Agora usaremos o seguinte atributo para realizar sua leitura:

```
[DataSource("System.Data.Odbc", "Dsn=Excel Files;Driver={Microsoft Excel Driver (*.xls)};dbq=|DataDirectory|\\Massa de Dados\\LoginValido.xls;defaultdir=.;driverid=790;maxbuffersize=2048;pagetimeout=5;readonly=true", "Plan1$", DataAccessMethod.Sequential)]
```

Os campos em vermelho representam o caminho do arquivo e o nome da aba onde será lido o dado.

O '|DataDirectory|' representa o diretório de saída dos arquivos compilados, em nosso projeto é a pasta no caminho: <Pasta do Projeto>/Bin/debug/.

Este atributo deve ser utilizado acima do método de teste em que deseja carregar a planilha apontada nele.

Para ler as colunas devemos usar o comando:

```
TestContext.DataRow["NomeDaColuna"].ToString()
```

O teste de login então ficaria da seguinte forma:

```
[TestMethod]
[DataSource("System.Data.Odbc", "Dsn=Excel Files;Driver={Microsoft Excel Driver (*.xls)};dbq=|DataDirectory|\\Massa de Dados\\LoginValido.xls;defaultdir=.;driverid=790;maxbuffersize=2048;pagetimeout=5;readonly=true", "Plan1$", DataAccessMethod.Sequential)]
public void LoginValido()
{
    string email = TestContext.DataRow["Email"].ToString();
    string senha = TestContext.DataRow["Senha"].ToString();

    //Navega ate a url
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");

    HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
    homenaoLogadaPage.AcessarLogin();

    //Forma Estrutural
    LoginPage loginPage = new LoginPage(driver);
    loginPage.PreencherEmail()
        .PreencherSenha()
        .ClicarSignIn();

    //Ou da forma funcional
    //loginPage.Logar();

    HomeLogadaPage homeLogada = new HomeLogadaPage(driver);
    Assert.IsNotNull(homeLogada.VerificaHomeLogada());
}
```

Tente inserir um breakpoint e execute o teste em modo de depuração na linha com a senha e verifique se os valores estão sendo lidos a partir da planilha.

Importante:

- Verifique se os caminhos apontados no atributo estão iguais aos que foram criados
- Verifique o arquivo está com a propriedade configurada para ser copiado no diretório de saída

Se caso for exibido uma falha ao tentar executar mesmo após ter checado os itens acima, tente baixar e instalar o seguinte arquivo:

<https://www.microsoft.com/en-us/download/details.aspx?id=23734>

Este método utiliza um driver do Excel para a leitura da planilha que pode estar ausente em seu computador. No link acima é feito o download dele. Depois de instalar reinicie o Visual Studio e teste novamente.

Caso a execução tenha dado certo, agora temos que passar os valores para os testes. Basta incluir os parâmetros de entrada em cada método, ficando da seguinte forma:

Dentro da login Page:

```
public LoginPage PreencherEmail(string email)
{
    txtEmail.SendKeys(email);
    return this;
}

public LoginPage PreencherSenha(string senha)
{
    txtSenha.SendKeys(senha);
    return this;
}

public LoginPage Logar(string email, string senha)
{
    PreencherEmail(email);
    PreencherSenha(senha);
    ClicarSignIn();
    return this;
}
```

E no teste de login:

```
[TestMethod]
[DataSource("System.Data.Odbc", "Dsn=Excel Files;Driver={Microsoft Excel Driver (*.xls)};dbq=|DataDirectory|\Massa de Dados\LoginValido.xls;defaultdir =.;driverid=790;maxbuffersize=2048;pagetimeout=5;readonly=true", "Plan1$",
DataAccessMethod.Sequential)]
public void LoginValido()
{
    string email = TestContext.DataRow["Email"].ToString();
    string senha = TestContext.DataRow["Senha"].ToString();

    //Navega ate a url
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");

    HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
    homenaoLogadaPage.AcessarLogin();

    //Forma Estrutural
    LoginPage loginPage = new LoginPage(driver);
    loginPage.PreencherEmail(email)
        .PreencherSenha(senha)
        .ClicarSignIn();

    //Ou da forma funcional
    //loginPage.Logar(email,senha);

    HomeLogadaPage homeLogada = new HomeLogadaPage(driver);
    Assert.IsNotNull(homeLogada.VerificaHomeLogada());
}
```

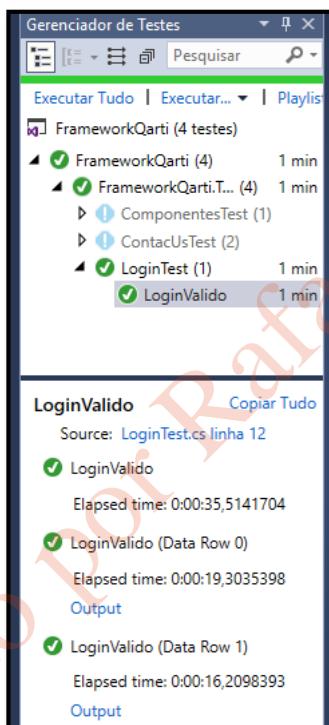
Agora ao executar, os valores inseridos na planilha serão utilizados pelo teste.

Um detalhe, este atributo oferece a possibilidade executar mais de um teste caso outra linha seja incluída no arquivo xls.

Caso inclua mais uma linha, serão executados dois testes. No exemplo abaixo repliquei os valores. Caso queira, teste com valores diferentes.

A	B
Email	Senha
teste@testing.com	1234qwer
teste@testing.com	1234qwer

A saída será a seguinte:



Este método é chamado de **Data Driven Test**, permitindo que novos testes sejam incluídos apenas com a adição de mais linhas com dados na massa de dados de entrada.

Mesmo sendo um recurso disponível aqui não faremos uso dele. Mantenha apenas uma linha por arquivo, onde cada teste deve ler de um arquivo diferente.

Repare que na saída é exibido o seguinte:

LoginVálido (Data Row 0)

LoginVálido (Data Row 1)

Apesar do suporte ao data-driven, o MsTest v1 não detalha de maneira clara quais testes foram executados, dificultando a análise dos resultados, em outros frameworks pode ser exibido de forma mais clara.

Sugestão : Busque mais informações sobre **Data Driven Tests (DDT)**.

Existem outros formatos suportados por esse atributo do MsTest:

- CSV

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
    "|DataDirectory|\data.csv", "data#csv", DataAccessMethod.Sequential),
DeploymentItem("data.csv"), TestMethod]
```

- Excel

```
DataSource("System.Data.Odbc", "Dsn=ExcelFiles;Driver={Microsoft Excel Driver (*.xls)};dbq=|DataDirectory|\Data.xls;defaultdir=.;driverid=790;maxbuffersize=2048;pagetimeout=5;readonly=true", "Sheet1$", DataAccessMethod.Sequential),
DeploymentItem("Sheet1.xls"), TestMethod]
```

- Test case in Team Foundation Server

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.TestCase",
    "http://vml13261329:8080/tfs/DefaultCollection;Agile", "30",
    DataAccessMethod.Sequential), TestMethod]
```

- XML

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML",
    "|DataDirectory|\data.xml", "Iterations", DataAccessMethod.Sequential),
DeploymentItem("data.xml"), TestMethod]
```

- SQL Express

```
[DataSource("System.Data.SqlClient", "Data Source=.\\sqlexpress;Initial Catalog=tempdb;Integrated Security=True", "Data", DataAccessMethod.Sequential),
TestMethod]
```

Detalhes em : https://docs.microsoft.com/pt-pt/visualstudio/test/creating-a-data-driven-coded-ui-test?view=vs-2015#CreateDataDrivenCUIT_QA_DataSourceAttributes

4.14.1 Exercício

Tente realizar a leitura de dados via Excel em outro teste.

Realize os mesmos procedimentos citados acima, mas crie um novo arquivo de entrada para cada teste.

4.15 Faker

Os testes que criamos até o momento são simples e com poucos campos, imaginando um teste onde teríamos muito mais campos seria necessário um trabalho maior para leitura de todos os campos.

Dependendo do projeto, podemos gerar aleatoriamente alguns campos que não possuem tanta relevância ou que não precisam de uma massa específica.

Em diversas linguagens é possível encontrar alguma biblioteca que gere estes dados aleatórios falsos, normalmente conhecidas como '**Fakers**'.

Vamos inclui-la no projeto realizando o download através do gerenciador de pacotes do NuGet, busque por:

- **Bogus**

Após a inclusão, crie uma nova classe de teste chamada ‘**FakerTest**’. Não é necessário fazer com que esta classe herde do ‘**BaseTest**’, primeiro iremos testar as possibilidades e depois criaremos um teste utilizando o faker.

Esta biblioteca foi escolhida por sua grande gama de opções, mas existem muitas outras disponíveis.

Em sua classe de teste ‘**FakerTest**’ inclua o seguinte código:

```
[TestMethod]
public void TesteFaker()
{
    Faker gerador = new Faker();
    string nome = gerador.Name.FirstName();
    Console.WriteLine(nome);
}
```

```
Faker gerador = new Faker(); : Cria a instância para o gerador de dados
gerador.Name.FirstName(); : Gera um nome aleatório
```

Execute o teste, verá que na saída foi gerado um nome qualquer.

Agora altere o código para o seguinte:

```
Faker gerador = new Faker();
string nome = gerador.Name.FirstName();
string sobrenome = gerador.Name.LastName();
Console.WriteLine(nome + " " + sobrenome);
```

Após executar veremos um outro nome, seguido do seu sobrenome.

Ainda temos a opção de escolher o gênero com o seguinte código:

```
Name.Gender genero = Name.Gender.Female; : Aqui iremos gerar uma variável do tipo feminino, mas
podemos alterar o Female para Male e gerar um masculino.
```

Usando como parâmetro nos nomes a variável ‘genero’ o código fica da seguinte forma:

```
Faker gerador = new Faker();
Name.Gender genero = Name.Gender.Female;
Console.WriteLine(genero);
string nome = gerador.Name.FirstName(genero);
string sobrenome = gerador.Name.LastName(genero);
Console.WriteLine(nome + " " + sobrenome);
```

Note que agora os nomes são gerados de acordo com o gênero escolhido.

Os demais exemplos que irei mostrar são apenas alguns itens disponíveis na biblioteca Bogus. Basta incluir abaixo deste último trecho que executamos com os nomes.

Para gerar um email:

```
string email = gerador.Internet.Email();
Console.WriteLine(email);
```

E para gerar um email com base nos nomes já gerados anteriormente:

```
string emailCustomizado = gerador.Internet.Email(nome, sobrenome);
```

```
Console.WriteLine(emailCustomizado);
```

Gerar uma data passada para simular uma data de aniversário (note que a data vem com padrão americano):

```
DateTime dataNascimento = gerador.Date.Past();  
Console.WriteLine(dataNascimento);
```

Simular que a data de aniversário seja de alguém entre 18 e 70 anos (agora também formatando a data):

```
string birthDate = gerador.Date.Past(70, DateTime.Now.AddYears(-  
18)).ToString("dd/MM/yyyy");  
Console.WriteLine(birthDate);
```

Gerar um número de telefone:

```
string telefone = gerador.Phone.PhoneNumber().ToString();  
Console.WriteLine(telefone);
```

Gerar um número de telefone com máscara de DDD, e que seja iniciado por 9 para simular um celular.

```
string celular = gerador.Phone.PhoneNumber("(##)9#####").ToString();  
Console.WriteLine(celular);
```

Gerar dados de endereço:

```
string rua = gerador.Address.StreetName();  
Console.WriteLine(rua);  
string complemento = gerador.Address.SecondaryAddress();  
Console.WriteLine(complemento);  
string numero = gerador.Address.BuildingNumber();  
Console.WriteLine(numero);  
string cidade = gerador.Address.City();  
Console.WriteLine(cidade);  
string estado = gerador.Address.State();  
Console.WriteLine(estado);
```

Gerar um lorem ipsum com 5 parágrafos:

```
string lorem = gerador.Lorem.Paragraph(5);  
Console.WriteLine(lorem);
```

Porém, repare que até agora todos os dados vieram com nomes em inglês ou outro idioma estrangeiro. Para localizar os dados em Português, basta incluir o parâmetro “pt_BR” no construtor do Faker.

```
Faker gerador = new Faker("pt_BR");
```

Execute o teste novamente, note que agora os dados gerados estão em português.

Além disso, também é possível gerar CPF e CNPJ (será solicitada a importação de uma classe de extensão do bogus):

```
string cpf = gerador.Person.Cpf();  
Console.WriteLine(cpf);  
string cnpj = gerador.Company.Cnpj();  
Console.WriteLine(cnpj);
```

Estas foram apenas algumas possibilidades, a documentação com todas as opções está em:
<https://github.com/bchavez/Bogus>

4.15.1 Exercício com faker

Agora que vimos como gerar os dados aleatoriamente, podemos tentar realizar o teste de cadastro utilizando esta nova biblioteca. Vamos assumir que para nosso teste o cadastro sempre será realizado utilizando um novo email válido qualquer. E para os demais dados do formulário de cadastro também vamos escolher dados aleatórios.

Caso tivéssemos acesso à base de dados, poderíamos, por exemplo, conferir após a geração do email se o ele já existe para evitar uma possível falha de tentativa de cadastro com email já existente.

Tente criar o novo teste de cadastro antes de ver como ficará no exemplo. Os passos são os seguintes:

- Acessar o site: <http://automationpractice.com/index.php>
- Clicar em Sign In
- Em ‘Create an Account’, insira o email gerado e clique em ‘Create an Account’
- Na página de cadastro preencha todos os dados de gerando ou selecionado dados aleatórios
- Após o preenchimento clique em ‘Register’
- Valide a confirmação do cadastro

Sugestão de resolução:

A navegação até a página de login já está feita, então inclua o campo para preenchimento do novo email e o clique no botão ‘Create na Account’ na LoginPage:

```
[FindsBy(How = How.Id, Using = "email_create")]
private IWebElement txtNovoEmail { get; set; }

[FindsBy(How = How.Id, Using = "SubmitCreate")]
private IWebElement btnCreateAccount { get; set; }

public LoginPage PreencherNovoEmail(string email)
{
    txtNovoEmail.SendKeys(email);
    return this;
}

public LoginPage ClicarCriarConta()
{
    btnCreateAccount.Click();
    return this;
}
```

Após isso, será criada a CadastroPage com seus elementos e ações:

```
public class CadastroPage : BasePage
{
    public CadastroPage(IWebDriver driver) : base(driver)
    {
    }

    #region Objetos

    #region YourPersonalInformation

    [FindsBy(How = How.Id, Using = "id_gender1")]
    private IWebElement radTituloMasculino { get; set; }
```

```

[FindsBy(How = How.Id, Using = "id_gender2")]
private IWebElement radTituloFeminino { get; set; }

[FindsBy(How = How.Id, Using = "customer_firstname")]
private IWebElement txtNomePersonalInfo { get; set; }

[FindsBy(How = How.Id, Using = "customer_lastname")]
private IWebElement txtSobrenomePersonalInfo { get; set; }

[FindsBy(How = How.Id, Using = "email")]
private IWebElement txtEmail { get; set; }

[FindsBy(How = How.Id, Using = "passwd")]
private IWebElement txtSenha { get; set; }

[FindsBy(How = How.Id, Using = "days")]
private IWebElement cbbDiaNascimento { get; set; }

[FindsBy(How = How.Id, Using = "months")]
private IWebElement cbbMesNascimento { get; set; }

[FindsBy(How = How.Id, Using = "years")]
private IWebElement cbbAnoNascimento { get; set; }

[FindsBy(How = How.Id, Using = "newsletter")]
private IWebElement chkSignUpNewsletter { get; set; }

[FindsBy(How = How.Id, Using = "optin")]
private IWebElement chkReceiveSpecialOffers { get; set; }

#endregion YourPersonalInformation

#region YourAddress

[FindsBy(How = How.Id, Using = "firstname")]
private IWebElement txtNomeAddress { get; set; }

[FindsBy(How = How.Id, Using = "lastname")]
private IWebElement txtSobrenomeAddress { get; set; }

[FindsBy(How = How.Id, Using = "company")]
private IWebElement txtEmpresa { get; set; }

[FindsBy(How = How.Id, Using = "address1")]
private IWebElement txtEndereço { get; set; }

[FindsBy(How = How.Id, Using = "address2")]
private IWebElement txtComplemento { get; set; }

[FindsBy(How = How.Id, Using = "city")]
private IWebElement txtCidade { get; set; }

[FindsBy(How = How.Id, Using = "id_state")]
private IWebElement cbbEstado { get; set; }

[FindsBy(How = How.Id, Using = "postcode")]
private IWebElement txtCep { get; set; }

[FindsBy(How = How.Id, Using = "id_country")]
private IWebElement cbbPais { get; set; }

[FindsBy(How = How.Id, Using = "other")]

```

```

private IWebElement txtInformacoesAdicionais { get; set; }

[FindsBy(How = How.Id, Using = "phone")]
private IWebElement txtTelefone { get; set; }

[FindsBy(How = How.Id, Using = "phone_mobile")]
private IWebElement txtCelular { get; set; }

[FindsBy(How = How.Id, Using = "alias")]
private IWebElement txtIdentificacaoEndereco { get; set; }

[FindsBy(How = How.Id, Using = "submitAccount")]
private IWebElement btnRegistrar { get; set; }

#endregion YourAddress

#region Objetos

#region Metodos

public CadastroPage SelecionarTitulo(string texto)
{
    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(radTituloMasculino));

    if (texto == "Mr.")
    {
        radTituloMasculino.Click();
    }
    else if (texto == "Mrs.")
    {
        radTituloFeminino.Click();
    }
    return this;
}

public CadastroPage PreencherNomeInformacoesPessoais(string texto)
{
    txtNomePersonalInfo.SendKeys(texto);
    return this;
}

public CadastroPage PreencherSobreNomeInformacoesPessoais(string texto)
{
    txtSobrenomePersonalInfo.SendKeys(texto);
    return this;
}

public CadastroPage ClicarCampoEmail()
{
    txtEmail.Click();
    return this;
}

public CadastroPage PreencherCampoSenha(string texto)
{
    txtSenha.SendKeys(texto);
    return this;
}

public CadastroPage SelecionarDiaNascimento(string texto)

```

```
{  
    SelectElement combo = new SelectElement(cbbDiaNascimento);  
    combo.SelectByValue(texto);  
    return this;  
}  
  
public CadastroPage SelecionarMesNascimento(string texto)  
{  
    SelectElement combo = new SelectElement(cbbMesNascimento);  
    combo.SelectByText(texto + " ");  
    return this;  
}  
  
public CadastroPage SelecionarAnoNascimento(string texto)  
{  
    SelectElement combo = new SelectElement(cbbAnoNascimento);  
    combo.SelectByValue(texto);  
    return this;  
}  
  
public CadastroPage SelecionarRecebimentoNewsletter()  
{  
    chkSignUpNewsletter.Click();  
    return this;  
}  
  
public CadastroPage SelecionarRecebimentoOfertasEspeciais()  
{  
    chkReceiveSpecialOffers.Click();  
    return this;  
}  
  
public CadastroPage PreencherNomeEndereço(string texto)  
{  
    txtNomeAddress.SendKeys(texto);  
    return this;  
}  
  
public CadastroPage PreencherSobreNomeEndereço(string texto)  
{  
    txtSobrenomeAddress.SendKeys(texto);  
    return this;  
}  
  
public CadastroPage PreencherEmpresa(string texto)  
{  
    txtEmpresa.SendKeys(texto);  
    return this;  
}  
  
public CadastroPage PreencherEndereço(string texto)  
{  
    txtEndereço.SendKeys(texto);  
    return this;  
}  
  
public CadastroPage PreencherComplemento(string texto)  
{  
    txtComplemento.SendKeys(texto);  
    return this;  
}
```

```
public CadastroPage PreencherCidade(string texto)
{
    txtCidade.SendKeys(texto);
    return this;
}

public CadastroPage SelecionarEstado()
{
    SelectElement combo = new SelectElement(cbbEstado);
    int randomIndex = new Random().Next(1, (combo.Options.Count - 1));
    combo.SelectByIndex(randomIndex);
    return this;
}

public CadastroPage PreencherCep(string texto)
{
    txtCep.SendKeys(texto);
    return this;
}

public CadastroPage SelecionarPais(string texto)
{
    SelectElement combo = new SelectElement(cbbPais);
    combo.SelectByText(texto);
    return this;
}

public CadastroPage PreencherInformacoesAdicionais(string texto)
{
    txtInformacoesAdicionais.SendKeys(texto);
    return this;
}

public CadastroPage PreencherTelefone(string texto)
{
    txtTelefone.SendKeys(texto);
    return this;
}

public CadastroPage PreencherCelular(string texto)
{
    txtCelular.SendKeys(texto);
    return this;
}

public CadastroPage PreencherIdentificacaoEndereco(string texto)
{
    txtIdentificacaoEndereco.Clear();
    txtIdentificacaoEndereco.SendKeys(texto);
    return this;
}

public CadastroPage ClicarRegistrar()
{
    btnRegistrar.Click();
    return this;
}

//Criando um metodo para forma funcional
public CadastroPage PreencherCadastroValido(string titulo, string nome, string sobreNome,
    string senha, string diaNascimento, string mesNascimento,
    string anoNascimento, string empresa, string endereco,
```

```

        string complemento, string cidade, string cep,
        string pais, string informacoesAdicionais, string telefone,
        string celular, string identificacaoEndereco)
    {
        SelecionarTitulo(titulo)
        .PreencherNomeInformacoesPessoais(nome)
        .PreencherSobreNomeInformacoesPessoais(sobreNome)
        .ClicarCampoEmail()
        .PreencherCampoSenha(senha)
        .SelecionarDiaNascimento(diaNascimento)
        .SelecionarMesNascimento(mesNascimento)
        .SelecionarAnoNascimento(anoNascimento)
        .SelecionarRecebimentoNewsletter()
        .SelecionarRecebimentoOfertasEspeciais()
        .PreencherEmpresa(empresa)
        .PreencherEndereco(endereco)
        .PreencherComplemento(complemento)
        .PreencherCidade(cidade)
        .SelecionarEstado()
        .PreencherCep(cep)
        .SelecionarPais(pais)
        .PreencherInformacoesAdicionais(informacoesAdicionais)
        .PreencherTelefone(telefone)
        .PreencherCelular(celular)
        .PreencherIdentificacaoEndereco(identificacaoEndereco)
        .ClicarRegistrar();
        return this;
    }

    #endregion Metodos
}

```

Notas:

Repare que usei as tags `#region` e `#endregion` para organização do código. Útil especialmente nestes caso em que a classe fica muito extensa, subdividindo o código de acordo com seus tipos e utilizações.

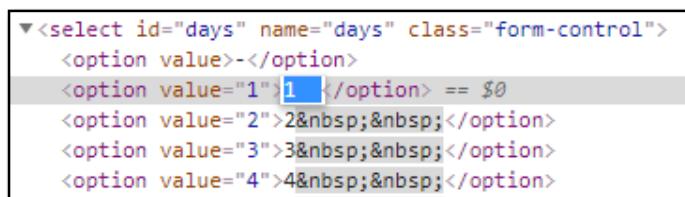
Para aguardar o carregamento da página, aguardei que o primeiro elemento com o qual interagimos fosse clicável através de uma espera explícita.

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(radTituloMasculino));
```

Dependendo do título selecionado,(Mr. Ou Mrs.), faço a seleção do checkbox.

No campo email, foi apenas dado um clique para que fosse validado pelo site.

As combos Dia, Mês e Ano de nascimento possuem o caractere ‘ ’ após o texto, que significa um espaço em branco sem quebra no HTML. Caso a seleção dos valores seja feito por texto, é necessário adicionar dois espaços em branco como feito no campo Mês.



```
SelectElement combo = new SelectElement(cbbMesNascimento);
combo.SelectByText(texto + " ");
```

Na combo Dia e Ano, optei por selecionar pelo value, que correspondem ao texto. Assim não necessitando dos espaços. Repare também que neste caso a busca por texto é mais lenta se comparada a busca por valor.

Na combo Estado, foi selecionado um valor aleatório, sendo que o valor inicial do index é (1), já que o conteúdo do primeiro index (0),é apenas um caractere “-“. O valor máximo é o total de elementos menos um.

```
SelectElement combo = new SelectElement(cbbEstado);
int randomIndex = new Random().Next(1, (combo.Options.Count - 1));
combo.SelectedIndex(randomIndex);
```

Com isso agora podemos criar a classe CadastroTest com o seguinte teste:

```
[TestMethod]
public void CadastroValido()
{
    //Criando os dados necessários para o teste
    Faker gerador = new Faker("en_US");
    Name.Gender genero;
    genero = gerador.Person.Gender;
    string titulo;
    if (genero.ToString() == "Male")
    {
        titulo = "Mr.";
    }
    else
    {
        titulo = "Mrs.";
    }
    string nome = gerador.Name.FirstName(genero);
    string sobreNome = gerador.Name.LastName(genero);
    string email = gerador.Internet.Email(nome, sobreNome);
    string senha = "1234qwer";
    DateTime dataNascimento = gerador.Date.Past(70, DateTime.Now.AddYears(-18));
    string diaNascimento = dataNascimento.Day.ToString();
    string mesNascimento = new CultureInfo("en-
US").DateTimeFormat.GetMonthName(dataNascimento.Month);
    string anoNascimento = dataNascimento.Year.ToString();
    string empresa = gerador.Company.CompanyName();
    string endereco = gerador.Address.StreetAddress();
    string complemento = gerador.Address.SecondaryAddress();
    string cidade = gerador.Address.City();
    string cep = gerador.Address.ZipCode("#####");
    string pais = "United States";
    string informacoesAdicionais = gerador.Lorem.Paragraph(1);
    string telefone = gerador.Phone.PhoneNumber("(###) ###-####");
    string celular = gerador.Phone.PhoneNumber("(###) #####-####");
    string identificacaoEndereco = gerador.Address.StreetSuffix();

    //Navega ate a url
    driver.Navigate().GoToUrl("http://automationpractice.com/index.php");

    HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
    homenaoLogadaPage.AcessarLogin();

    LoginPage loginPage = new LoginPage(driver);
    loginPage.PreencherNovoEmail(email)
        .ClicarCriarConta();
```

```

//Forma Estrutural
CadastroPage cadastroPage = new CadastroPage(driver);
cadastroPage.SelecionarTitulo(titulo)
    .PreencherNomeInformacoesPessoais(nome)
    .PreencherSobreNomeInformacoesPessoais(sobreNome)
    .ClicarCampoEmail()
    .PreencherCampoSenha(senha)
    .SelecionarDiaNascimento(diaNascimento)
    .SelecionarMesNascimento(mesNascimento)
    .SelecionarAnoNascimento(anoNascimento)
    .SelecionarRecebimentoNewsletter()
    .SelecionarRecebimentoOfertasEspeciais()
    .PreencherEmpresa(empresa)
    .PreencherEndereco(endereco)
    .PreencherComplemento(complemento)
    .PreencherCidade(cidade)
    .SelecionarEstado()
    .PreencherCep(cep)
    .SelecionarPais(pais)
    .PreencherInformacoesAdicionais(informacoesAdicionais)
    .PreencherTelefone(telefone)
    .PreencherCelular(celular)
    .PreencherIdentificacaoEndereco(identificacaoEndereco)
    .ClicarRegistrar();

//Forma Funcional
//cadastroPage.PreencherCadastroValido(titulo, nome, sobreNome, senha, diaNascimento,
//    mesNascimento, anoNascimento, empresa, endereco,
//    complemento, cidade, cep, pais,
//    informacoesAdicionais, telefone, celular, identificacaoEndereco);

HomeLogadaPage homeLogada = new HomeLogadaPage(driver);
Assert.IsNotNull(homeLogada.VerificaHomeLogada());
}

```

Como já temos pronto a navegação até a página de login, e a verificação de um elemento da home logada, apenas complementamos o teste com as duas chamadas.

4.16 Arquivo de Configurações de Teste - .runsettings

Para incrementar a configuração do nosso framework, vamos incluir um arquivo do tipo `.runsettings`. Este arquivo é opcional e é incluído por projeto. Pode conter informações como parâmetros de teste, a versão do .netframework a ser utilizada, versão de compilação (x86 ou x64), diretório de saída dos testes, entre outros (veja link de referência abaixo).

Referência: <https://docs.microsoft.com/pt-br/visualstudio/test/configure-unit-tests-by-using-a-dot-runsettings-file?view=vs-2017>

No exemplo que iremos fazer irei mostrar apenas a inclusão de parâmetros de configuração de nossos testes.

Este arquivo serve apenas para Configurações do teste, e não deve ser utilizado para Massa de dados ! As massas de dados devem ser lidas a partir de outros arquivos externos, como banco de dados, excel, json, etc...

Alguns exemplos de parâmetros que podemos ter, o navegador a ser utilizado, o tipo de ambiente (testes, homologação, desenvolvimento, produção), usuário e senha de alguma aplicação (recomendado não manter neste arquivo, mas estes valores podem ser lidos de forma oculta pelo TFS), valores de configuração do tipo ‘ligado’ ou ‘desligado’, como abrir um bug automaticamente, ou tirar evidências de um determinado tipo entre outros...

Este arquivo pode parecer redundante quando utilizado aqui pelo Visual Studio, mas seu uso real é visto quando configurando uma Build ou uma Release que executa os testes pelo TFS. Através destes parâmetros não é necessário que o usuário tenha contato com o código para alterar estes valores.

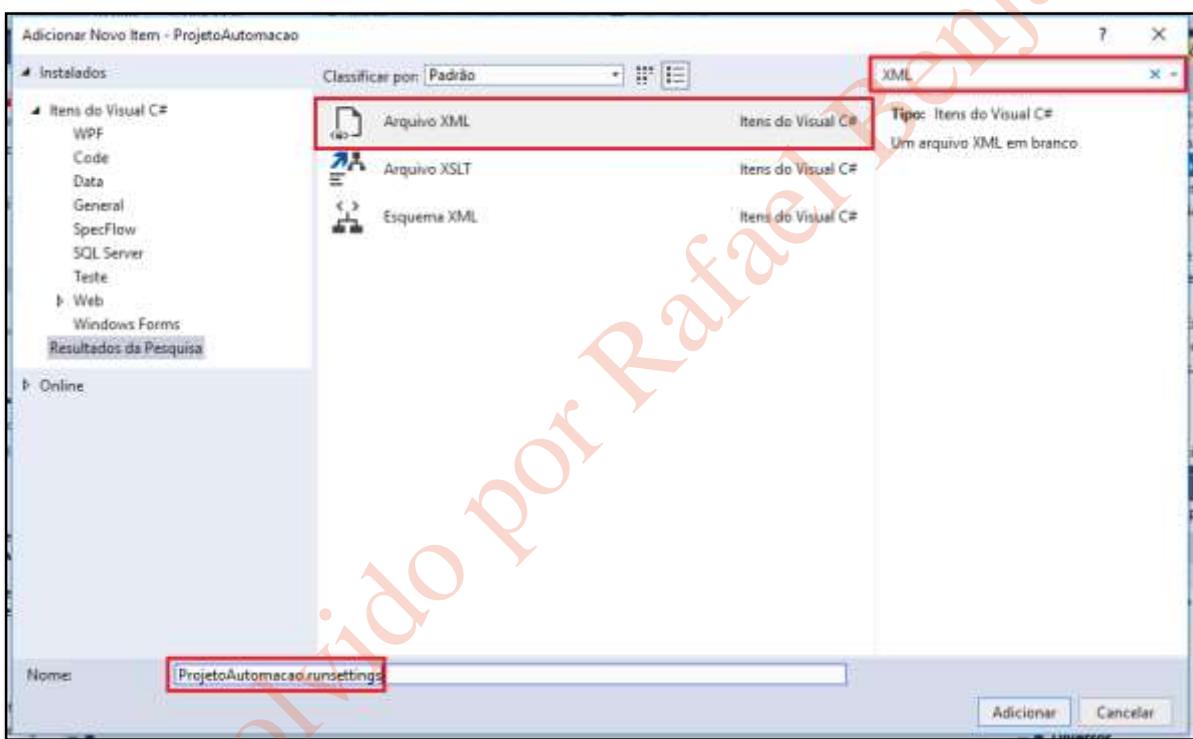
Estes valores podem ser sobreescritos pelos que forem definidos no TFS, mas caso não sejam sobreescritos por lá, os valores padrão que já estão definidos no arquivo .runsettings serão utilizados.

Mais em : <https://devblogs.microsoft.com/devops/supplying-run-time-parameters-to-tests/>

Para criar o arquivo, inclua um arquivo XML em seu projeto.

Botão direito no **Projeto** > Adicionar Novo Item > Arquivo XML

Use a barra de pesquisa no canto superior direito e busque por XML para facilitar.



Renomeie seu arquivo para **ProjetoAutomacao.runsettings** (remova a extensão final .xml)

O importante é que o arquivo possua a extensão **.runsettings**, neste caso dei o nome de ProjetoAutomacao que é o nome do meu projeto, mas poderia ser qualquer nome. Caso você tenha mais de um projeto com esse tipo de arquivo, o ideal é manter um nome que remeta ao projeto no qual ele está.

Após incluir o arquivo, copie o seguinte código dentro dele:

```
<?xml version="1.0" encoding="utf-8" ?>
<RunSettings>

<TestRunParameters>

<Parameter name="browser" value="Chrome" />
```

```
</TestRunParameters>  
</RunSettings>
```

O que foi feito agora, é que estamos especificando uma tag do tipo **RunSettings**, e outra tag do tipo **TestRunParameters**. Dentro desta última é que iremos incluir os parâmetros referentes aos nossos testes, no código de exemplo já criei o primeiro parâmetro de configuração que será o navegador.

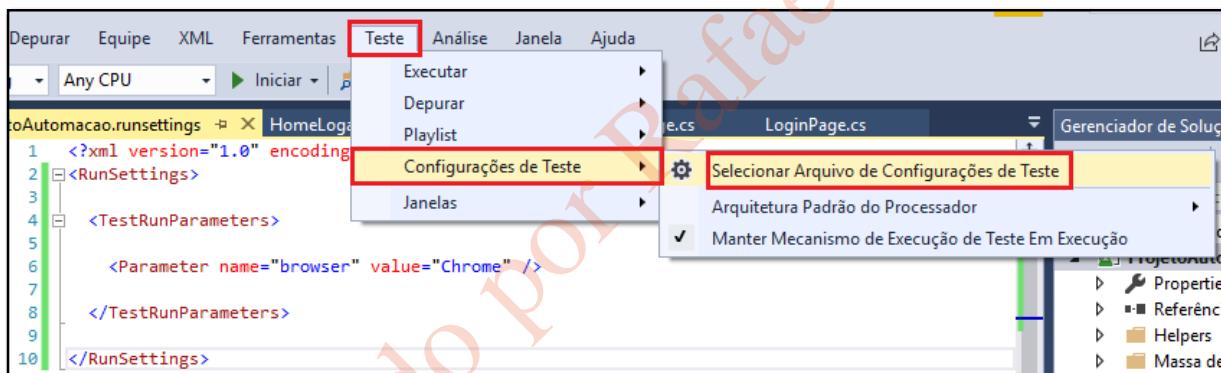
Cada parâmetro deve estar dentro da tag TestRunParameters, e devem seguir este formato:

```
<Parameter name="NomePropriedade" value="Valor" />
```

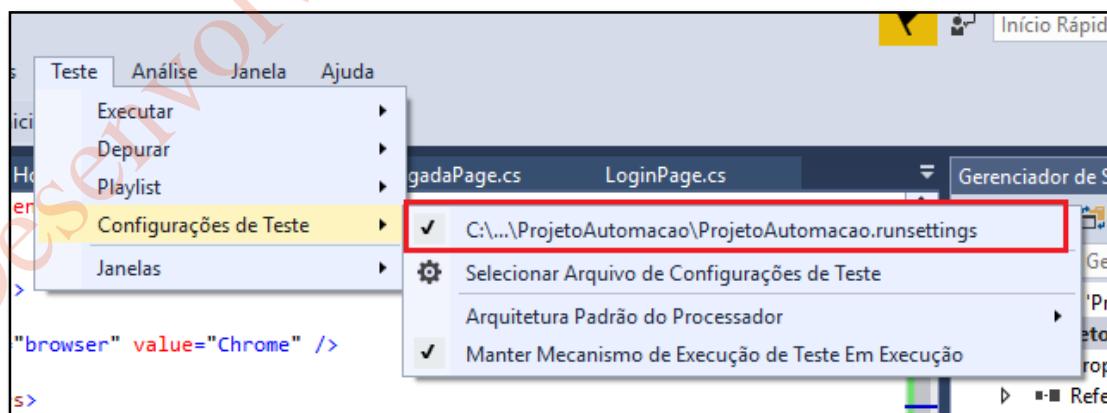
O nome e valor serão definidos de acordo com sua necessidade.

Para podermos utilizar este arquivo, agora temos que apontar para ele pelo Visual Studio.

No menu **Teste**, escolha **Configurações de Teste > Selecionar Arquivo de Configurações de Teste**. Procure o arquivo **.runsettings** que você criou e, em seguida, selecione **OK**.



Após selecionar o arquivo ele será exibido acima desta opção:



Sempre que carregar um projeto que necessita deste tipo de arquivo, será necessário associá-lo da primeira vez que carregar o projeto em seu computador.

Pronto, agora nosso projeto pode ler os valores que estão definidos no .runsettings especificado.

Neste exemplo temos o browser, então vamos atualizar nosso arquivo **BaseTest** para que o browser seja lido através dele.

Para que seja possível ler o dado deste arquivo, temos que possuir um **TestContext** inicializado.

Podemos ler os parâmetros da seguinte forma:

```
TestContext.Properties["NomePropriedade"]
```

Este código retorna o valor especificado, dependendo do tipo de valor talvez seja necessário realizar uma conversão para string, int, bool, etc...

Então o nosso método **TestInitialize**, aonde setamos o navegador ficará da seguinte forma:

```
[TestInitialize]
public void TestInitialize()
{
    string browser = TestContext.Properties["browser"].ToString();
    driver = CriaDriver(browser);
    driver.Manage().Window.Maximize();
    //Espera Implicita
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
}
```

Entendendo a alteração:

```
string browser = TestContext.Properties["browser"].ToString(); : Lê o parâmetro que definimos como browser no arquivo runsettings e salva o valor na variável browser. Como a variável é do tipo string foi necessário converte-la usando o .ToString() ao final.
driver = CriaDriver(browser); : No lugar da string "CHROME" que utilizávamos anteriormente, agora recebemos o valor salvo na variável browser, que por sua vez recebe o valor do arquivo .runsettings.
```

Note que antes recebíamos o valor "CHROME", ou o navegador desejado dentro da função **CriaDriver** em letras maiúsculas. Para evitar problemas de case sensitive, vamos converter o valor recebido para letras maiúsculas dentro da nossa função **CriaDriver**.

Use a função **.ToUpper** na linha aonde é realizado o switch, assim independente de como foi digitado o navegador ele será sempre convertido para maiúsculo e será compatível com nosso switch.

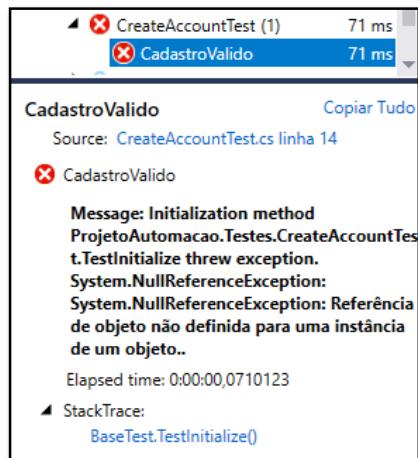
```
public IWebDriver CriaDriver(string browser)
{
    switch (browser.ToUpper())
    {
```

Tente executar algum teste agora e verifique se o chrome continua sendo aberto.

Após realizar o teste com sucesso, altere o navegador que deseja utilizar, pode ser escrito alternando letras maiúsculas ou minúsculas, já que fizemos o ajuste. Mas o nome do navegador desejado deve corresponder ao configurado no switch. Por exemplo, se no meu switch configurei o Internet Explorer como "ie", então devo utilizar este nome, caso escreva "Internet Explorer" não irá funcionar.

O ideal é que ao final do projeto, seja criado um manual de execução para que o usuário contivesse também quais valores ele pode utilizar em seu arquivo de configuração.

Caso seu teste apresente o seguinte erro:



Verifique as duas possibilidades:

- O arquivo de .runsettings não foi selecionado no menu **Teste**
- O nome do parâmetro que você está tentando chamar pelo `TestContext.Properties` não existe no runsettings, ou está escrito de formas diferentes no seu código e no seu arquivo .runsettings.

4.16.1 Exercício

No arquivo que acabamos de criar, inclua um novo parâmetro de teste para a definição do timeout implícito (não é recomendado usar, mas é só pra treinar aqui :) que também está configurado no `TestInitialize` do `BaseTest`.

Sugestão de Resolução:

No arquivo .runsettings inclui mais um parâmetro com o nome `timeoutImplicito`, e o valor é tempo em segundos.

```
<?xml version="1.0" encoding="utf-8" ?>
<RunSettings>
  <TestRunParameters>
    <Parameter name="browser" value="Chrome" />
    <Parameter name="timeoutImplicito" value="10" />
  </TestRunParameters>
</RunSettings>
```

E na função de `TestInitialize`, li o valor configurado e o atribui para uma variável do tipo int.

```
[TestInitialize]
public void TestInitialize()
{
    string browser = TestContext.Properties["browser"].ToString();
    driver = CriaDriver(browser);
    driver.Manage().Window.Maximize();
    //Espera Implicita
    int implicitTimeout = Convert.ToInt32(TestContext.Properties["timeoutImplicito"]);
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(implicitTimeout);
}
```

Os nomes dos parâmetros precisam ser de fácil entendimento para quem for utilizar.

5 Módulo 2 - Behaviour Driven Development - BDD

5.1 Conceitualização

O BDD é uma metodologia de desenvolvimento que inspira a comunicação entre os envolvidos no projeto. (Analistas de Negócios, Desenvolvedores e Analistas de testes)

O BDD surgiu em após a necessidade de criar algo que fosse de um entendimento mais fácil do que o [TDD](#) (Test Driven Development), onde é feito primeiro o desenvolvimento do teste e em seguida criado a funcionalidade. Os testes criados no TDD deveriam servir como guias, dando exemplos e de certa forma documento o que era esperado. Os testes eram uma consequência deste processo, mas era difícil para os desenvolvedores entenderem esta abordagem.

Então o criador do BDD criou um framework onde são descritos os comportamentos esperados nestas especificações para que o desenvolvimento seja feito, e como consequencia, estas especificações por exemplos podem ser automatizados.

Seu valor real é visto quando todos os envolvidos possuem colaboram na escrita destas especificações, para que todos definam o comportamento esperado do software antes do seu desenvolvimento. Se após sua implementação aconteça alguma atualização e os exemplos já estão automatizados, eles irão quebrar. Assim sendo necessária sua correção tanto da parte automatizada, quanto da sua documentação, mantendo sempre uma documentação viva.

TDD/BDD is not about testing

A common misunderstanding of TDD and BDD is that they are testing techniques. They're not. As the name suggests, TDD and BDD are about software development.

It is the process of approaching your design and forcing you to think about the desired outcome and API before you code.

Automated tests are a by-product of TDD and BDD.

Extraído de : <https://cucumber.io/docs/bdd/overview/>

Podemos dizer que é uma metodologia que utiliza a linguagem simples para descrever os *comportamentos* de uma aplicação, é muito semelhante aos casos de uso quando utilizamos o:

- Título da funcionalidade
- Descrição da funcionalidade
- Cenários compostos por uma sequência de ações

Faz uso do **Gherkin**, uma linguagem ubíqua que se apoia no uso de um vocabulário pequeno e bem específico, com foco na linguagem e no comportamento do usuário.

Escrevendo especificações que ilustram as regras de negócio, ao invés de funcionalidades da interface, permitindo envolver toda a equipe (analistas de negócios, desenvolvedores e analistas de negócios).

Referência: <https://docs.cucumber.io/gherkin/reference/>

5.2 Estrutura Principal

A estrutura que se usa para as especificações é o formato Given/When/Then em conformidade com gramática da linguagem Gherkin.

- Given (Dado) descreve uma pré-condição existente do estado de software antes de começar o comportamento que você está especificando.
- When (Quando) é o próprio comportamento.
- Then (Então) descreve o resultado esperado do comportamento.

Por exemplo:

Dado que me inscrevi para Loja de Livros Online, quando eu acesso a loja com minhas credenciais, então vejo uma mensagem "Bem-vinda à Loja de Livros Online!".

Como vimos acima, existem algumas palavras-chave, que detém um significado específico:

Funcionalidade, Contexto, Cenário, Esquema do Cenário, Exemplo, Dado, Quando, Então, E, Mas.

Abaixo o detalhe das palavras chave.

5.3 Palavras-chave

Funcionalidade: é a descrição do comportamento da funcionalidade, escritas em linguagem natural, pode-se incrementar a definição respondendo às perguntas: 'Quem? , O que?, Para que? '

Contexto: É a pré-condição do cenário (que agrupa os passos em comum). Pode-se usar mais de um por funcionalidade geralmente tem a instrução sobre o que preparação antes de cada cenário é executada.

Cenário: Os cenários são as “unidades executáveis” dos testes e normalmente representam uma fração da funcionalidade e contém os passos necessários para que um usuário que utiliza a aplicação possa validar essa fração.

Esquema do Cenário: Permite-nos formular exemplos de testes com maior precisão através da utilização de um modelo com espaços reservados, como se fossem um grupo de cenários com algumas variações.

Exemplo: Espaço reservado para a definição das variações de um Esquema do Cenário.

Dado: Ele descreve o pré-requisito para o teste a ser executado. Utilizado para a pré-Condição do cenário.

Quando: Ele define o ponto de disparo para qualquer execução cenário de teste, o comportamento da ação executada.

Então: Descreve o resultado esperado para o teste

E, Mas, *: Podem ser utilizados em conjunto com o DADO, QUANDO e ENTÃO. Servem para complementar ou facilitar a leitura dos passos sem criar uma repetição de palavras chave.

5.4 Regras de Escrita

5.4.1 Definir a linguagem no inicio do documento

Para que as palavras chaves sejam identificadas na língua utilizada, a notação com a língua deve ser comentada no início do arquivo. Comentários são feitos pelo uso de "#".

```
#language: pt
```

5.4.2 Funcionalidade

Descrever de uma maneira mais abstrata qual o propósito da funcionalidade, sem entrar em detalhes técnicos.

Um template a seguir como exemplo:

Como um <ator explícito do sistema>

Eu quero <O que quero alcançar>

Para que <Por que quero alcançar>

O cucumber não valida o conteúdo deste texto, então é possível adicionar mais detalhes como critérios de aceitação, ou seguir um template diferente dependendo da necessidade.

Ex:

```
Funcionalidade: Recuperar senha
```

```
Como um usuário padrão
```

```
Eu quero obter uma nova senha de acesso
```

```
Para que eu possa realizar o acesso quando esquecer minha senha
```

5.4.3 Cenário / Esquema do Cenário

Descrever de maneira objetiva qual teste será realizado, especificando o que será validado neste cenário.

Ex:

```
Cenário: Realizar o login utilizando credenciais válidas
```

```
Esquema do Cenário: Validar a exibição dos contratos
```

5.4.4 Contexto

O contexto permite descrever um ou uma série de passos que são comuns para os cenários. Ao invés de repetir os passos em todo cenário ou esquema do cenário, o Contexto é utilizado para especificar as pré-condições do teste.

Ex:

```
Contexto:
```

```
Dado Dado que acessei o sistema como um supervisor
```

```
E acessei o cadastro de novos funcionários
```

5.4.5 Dado / Quando / Então

Dado: Situação que deve ser atendida para seguir o cenário. Pré-condições/Contexto

Quando: Ação a ser realizada ou passos para atender os cenários

Então: Resultado esperado após a Ação.

5.4.6 Uso de Tags

Realizar o uso de tags para uma identificação rápida do tipo de cenário. Estas podem ser encadeadas de acordo com sua necessidade.

Cenários pendentes: Utiliza-se quando ainda não existe alguma definição final. Por exemplo quando o comportamento foi apresentado no protótipo, porém ainda não foi implementado pelo desenvolvimento, ou seja, é um planejamento futuro.

Ex:

```
@Pendente
```

Ignore: É uma tag reservada do cucumber, esta pula o cenário e não realiza sua execução. Utilizado por exemplo em um cenário que já foi implementado pelo desenvolvimento, mas não será executado pela automação, como o pendente por exemplo.

Ex:

```
@Ignore
```

Demais tag: Outras tags específicas e que possuam uma relevância para os cenários podem ser acordadas e incluídas caso necessário.

Ex:

```
@Regressivo @Smoke @Financeiro
```

5.4.7 Uso de Aspas

Usar aspas duplas ("") para se referir a textos ou mensagens ou para variáveis dentro das frases.

```
Quando receber o valor "10"
```

```
Quando visualizar a mensagem "Confirmado"
```

5.4.8 Doc String

Usado para validar textos que não estão em uma linha única.

O uso consiste em três aspas duplas, o texto a ser validado e fechado com mais três aspas duplas.

```
Então visualizarei a mensagem de sucesso
```

```
"""
```

```
Sua mensagem foi enviada com sucesso!
```

```
Nosso prazo de resposta é de 48 horas.
```

```
Solicitamos que fique atento à sua caixa de entrada, pois a resposta  
retornará com o email noreply@contato.com
```

```
"""
```

5.4.9 Dados variáveis em Esquema do Cenário

- Ao se referir a um campo ou resultado variável sempre utilizar o placeholder "<>"

- Sempre manter o nome do campo de exemplo em minúsculo, tanto dentro do placeholder quanto na tabela de exemplos.

Esquema do Cenário: Login inválido

Quando realizar o login com o usuário "<Email>" e Senha "<Senha>"

Então visualizarei a mensagem de erro "<Mensagem>"

Exemplos:

tipoTeste	Email	Senha	Mensagem
emailInvalido	teste	1234qwer	Invalid email address.
senhaInvalida	teste@te.com	123	Invalid password.

5.4.10 Foco no comportamento

Como a própria metodologia sugere “BDD” (Behaviour Driven Development), é voltada para o comportamento da aplicação, então é sugerido escrever os cenários com o foco no comportamento da mesma.

Os cenários devem ser escritos de maneira que qualquer um dos envolvidos, incluindo os que não estão envolvidos com a parte técnica e até os usuários possam ler e ter um entendimento do que o cenário está validando.

Os cenário podem ser mais **declarativos**, resumindo os comportamentos, ou mais **imperativos**, onde os passos ficam mais detalhados.

Exemplo de cenário **imperativo** com descrição focada nos elementos da tela (mais próximo de um caso de teste):

Cenário: Cadastro de novo usuário

Dado que acessei a home page

Quando clico no botão "Criar Novo Usuário"

E preencho o campo "Nome" com "João"

E preencho o campo "Email" com "joao@email.com"

E preencho o campo "Senha" com "1234qwer"

E clico em "Confirmar"

Então visualizarei a mensagem "Usuário cadastrado"

Exemplo de cenário **declarativo** com descrição focada no comportamento da tela:

Cenário: Cadastro de novo usuário

Dado que acessei a aplicação

Quando acesso o cadastro de novos usuários

E preencho as informações de um novo usuário

Então confirmo a efetuação do cadastro

5.4.11 Refatoração

Sempre que analisar o cenário ou esquema de cenário está muito grande ou existem muitas regras avalie se o cenário não pode ser melhorado e quebrado em novos cenários específicos.

Após escrita da feature, reler novamente e verificar se não existem ações idênticas sendo tratadas de formas diferentes. O uso de pronomes e artigos pode ser tratado, mas é recomendado que o padrão seja mantido.

Dado que acessei a aplicacao

Dado que realizei o login na aplicacao

Dado que eu me loguei na aplicacao

Evitar passos com mais de uma ação. Estes podem acabar se tornando desnecessariamente complexos e sem oportunidade para reaproveitamento de suas ações.

Quando preencho um email e realizo seu envio

Quando preencho em email

E envio o email

5.5 Specflow

Specflow é a implementação em .NET para utilização do cucumber. Possui integração com Visual Studio e com os principais frameworks de teste, como MsTest, NUnit e xUnit.

Atualmente nossos projetos usam a versão **2.4.1**, então neste tutorial irei tomar como base esta configuração. A versão **3** não é compatível o MsTestV1.

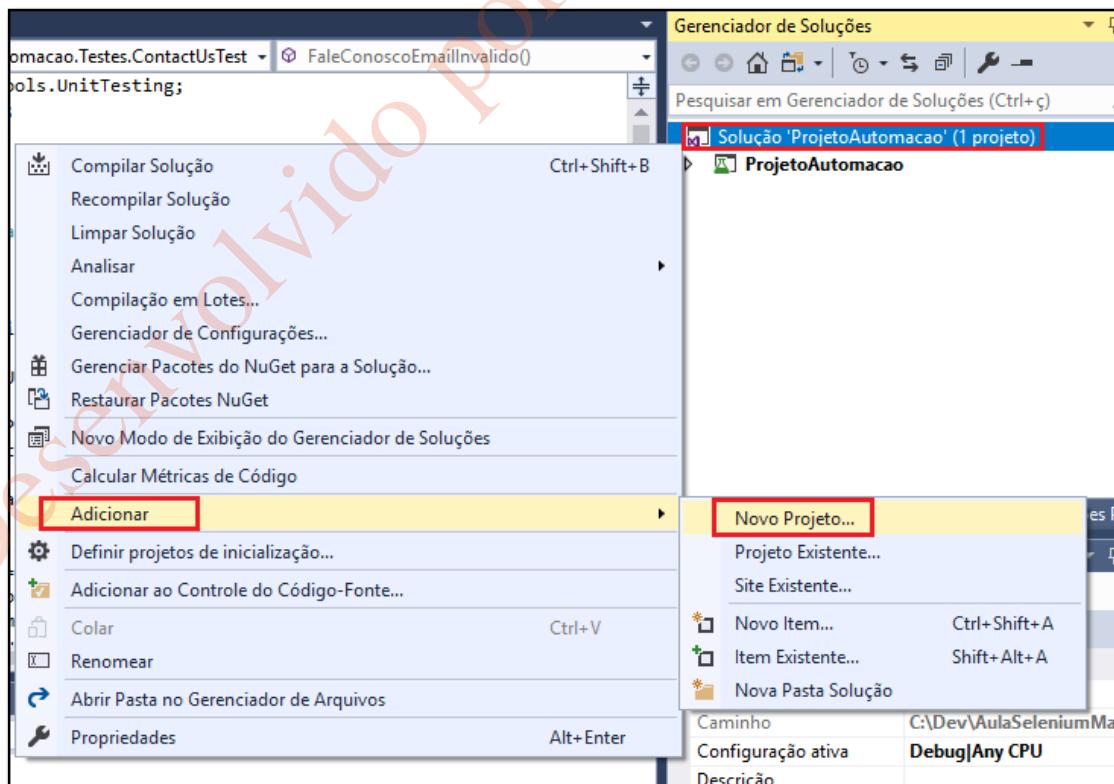
Esta seção do curso teve como base as seguintes vídeo aulas que estão apresentadas para o NUnit:

<https://www.youtube.com/playlist?list=PL6tu16kXT9Pp3wrsaYyNRnK1QkvVv6qdl>

Primeiro serão mostrados os principais recursos sem utilizar a automação de testes, e após isto iremos criar outro projeto para incluir os testes que criamos no módulo anterior neste formato.

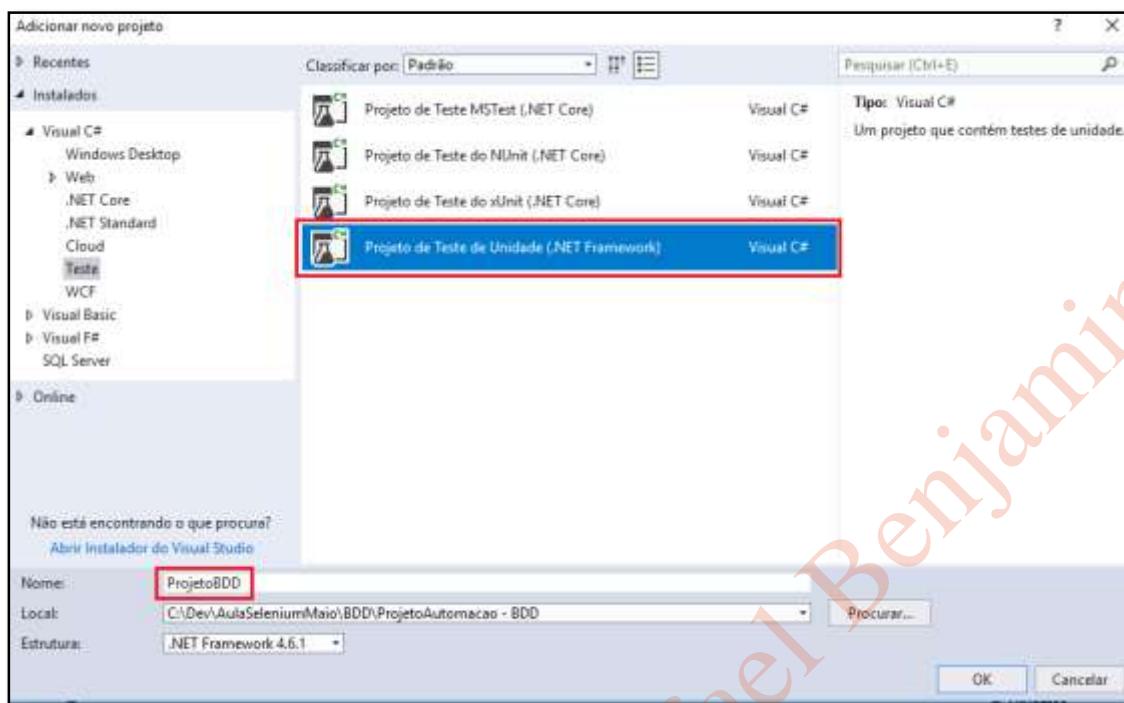
Vamos criar um novo projeto dentro da solução para separar os tipos de testes. O projeto que estamos usando será mantido utilizando apenas o selenium. O novo utilizará os testes em BDD.

Clique com o botão direito na **Solução > Adicionar > Novo Projeto**



Escolha o mesmo tipo que usamos para o primeiro projeto, **Projeto de Teste de Unidade (.Net Framework)**

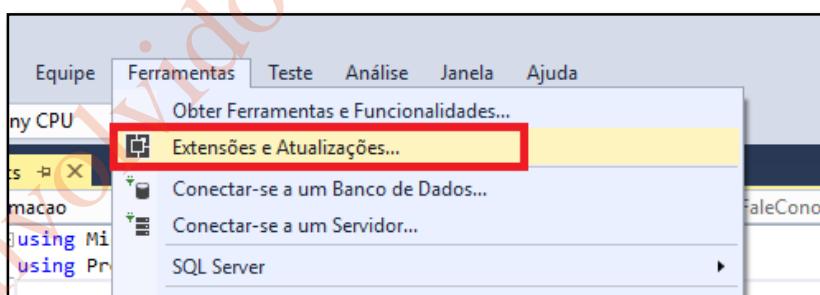
Vou chamar o meu de **ProjetoBDD**, e clique em OK



5.6 Plugin Specflow Visual Studio

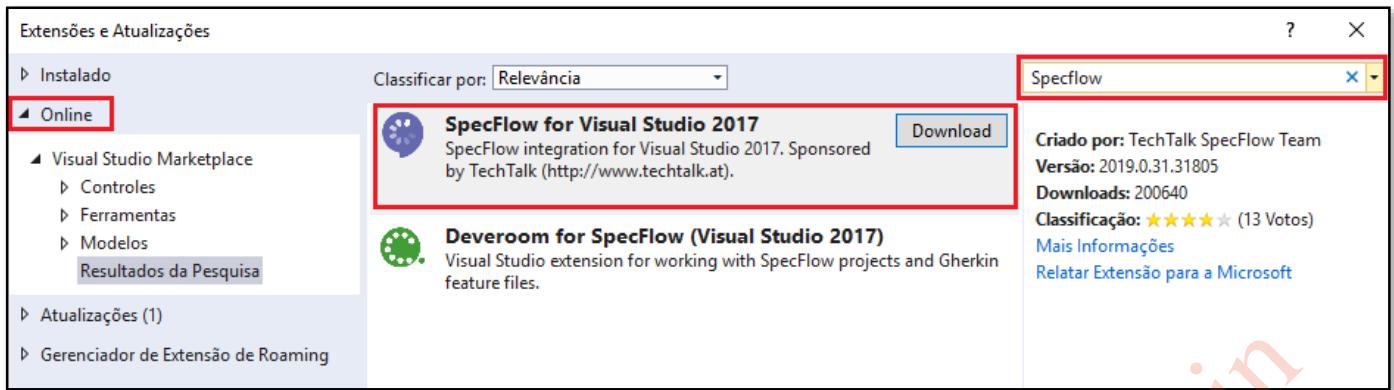
Antes de baixar os pacotes necessários, vamos instalar a extensão do Specflow para o Visual Studio. A instalação deste plugin é feita apenas uma vez.

Clique em **Ferramentas > Extensões e Atualizações...**



Clique em **Online** à esquerda e busque por: **Specflow for Visual Studio**

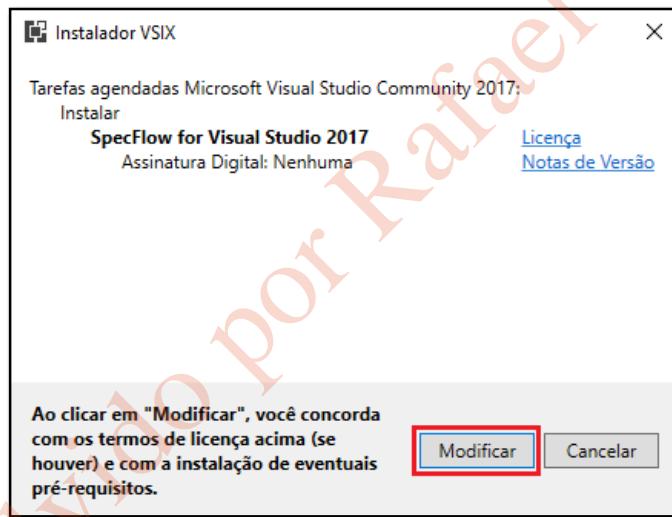
Clicar em Download



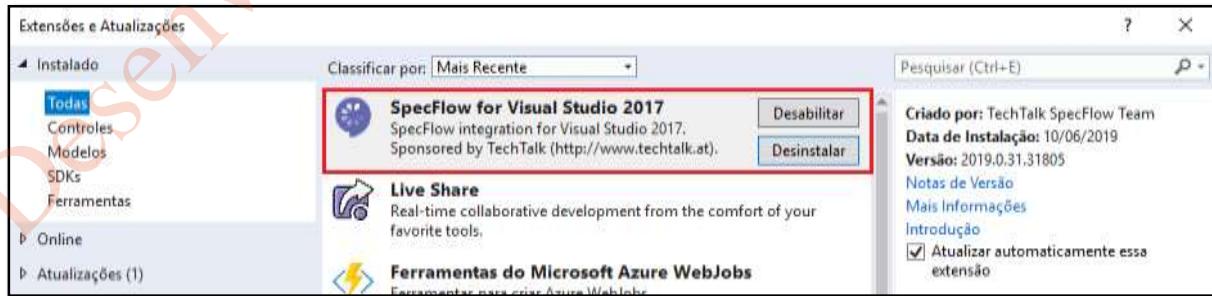
Será necessário fechar o visual Studio para concluir a instalação.



Após fechar aguarde a tela de instalação ser exibida, e selecione Modificar



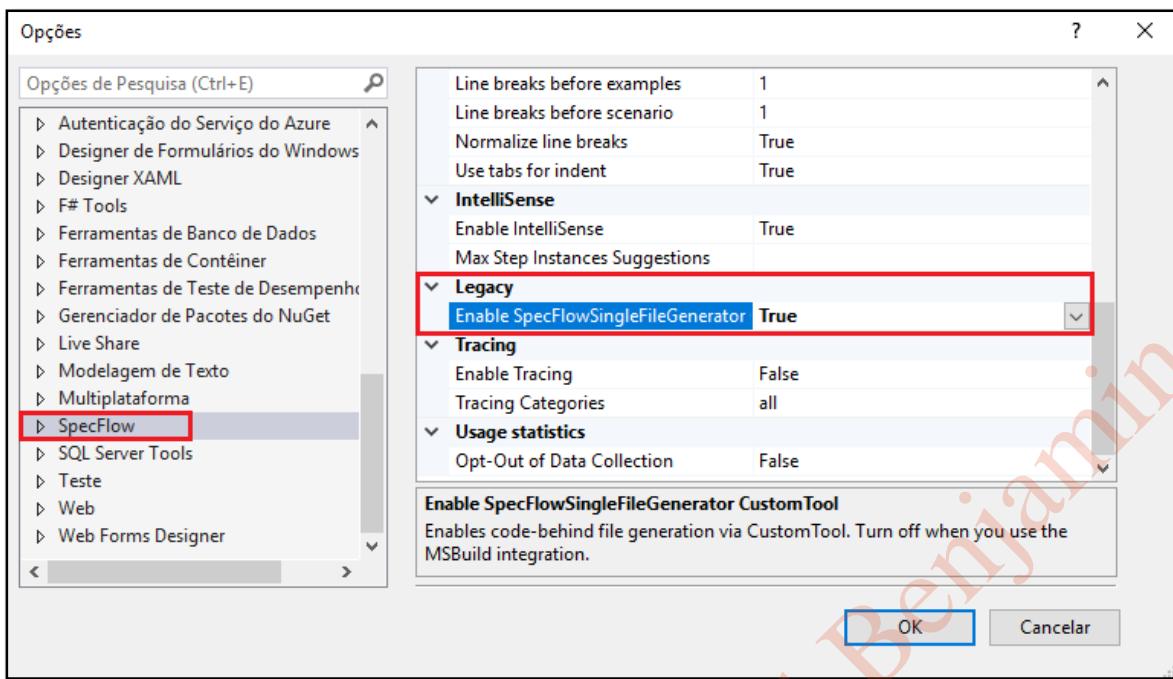
Após a conclusão abra o visual Studio novamente e verifique se a extensão foi instalada com sucesso.



Agora para configurarmos o plugin para que reconheça os arquivo do specflow v2.4.0.

Acesse o menu **Ferramentas > Opções**

Procure por **Specflow**, e em seguida na opção **Enable SpecFlowSingleFileGenerator**, altere o valor para **True**.

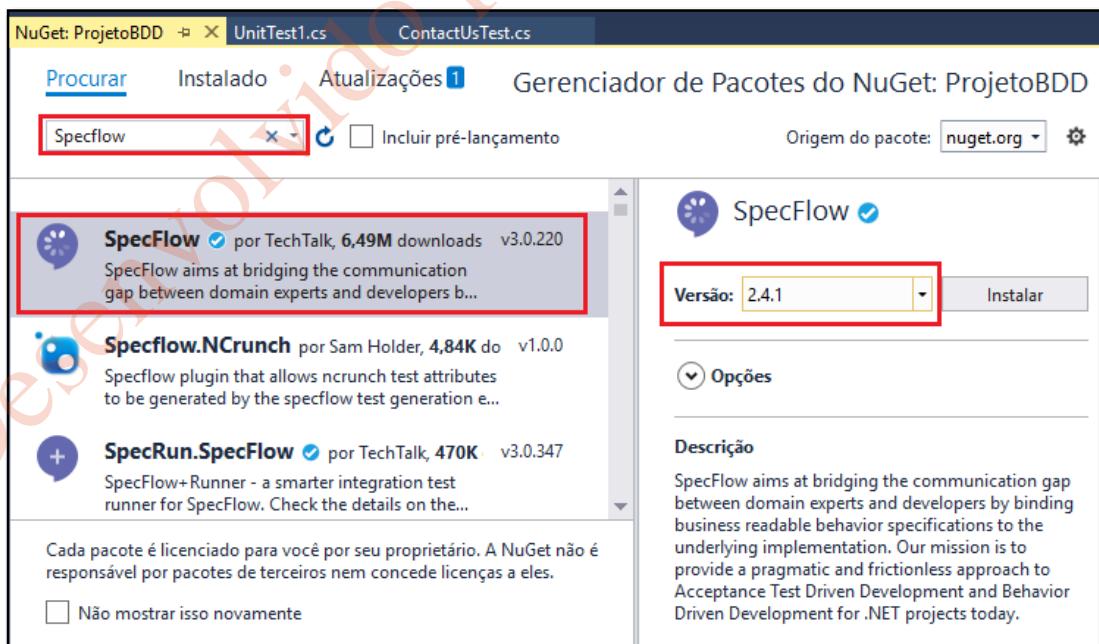


Como também usaremos o MsTest V1, siga novamente os passos descritos na seção [Alterando o framework de testes para MsTest Versão 1](#) para configurar o framework de testes removendo o MsTestV2 deste novo projeto.

Depois de configurar, apague a classe **UnitTest1.cs** que vem por padrão no projeto, não iremos criar testes neste formato aqui.

Após configurado, abra o gerenciador de pacotes de NuGet desta versão e baixe o seguinte pacote:

- **SpecFlow v2.4.1**



5.7 Specflow - Configuração

Para a configuração de projetos specflow versão 2.x, devemos apontar qual o framework de teste utilizado dentro do arquivo app.config do projeto.

Após o download do pacote nuget do specflow v2.4.1, um arquivo *app.config* será criado automaticamente no projeto. Abra-o, e entre as tags <specflow> insira a tag:

```
<unitTestProvider name="MsTest" />
```

Esta tag serve para que o specflow identifique qual framework está sendo utilizado, e realize a conversão interna dos arquivos features para o formato correspondente.

O arquivo final deve ficar da seguinte forma:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="specFlow" type="TechTalk.SpecFlow.Configuration.ConfigurationSectionHandler,
TechTalk.SpecFlow" />
  </configSections>
  <specFlow>
    <unitTestProvider name="MsTest" />
    <!-- For additional details on SpecFlow configuration options see http://go.specflow.org/doc-
config -->
  </specFlow>
</configuration>
```

Fonte: <https://specflow.org/documentation/Configuration/>

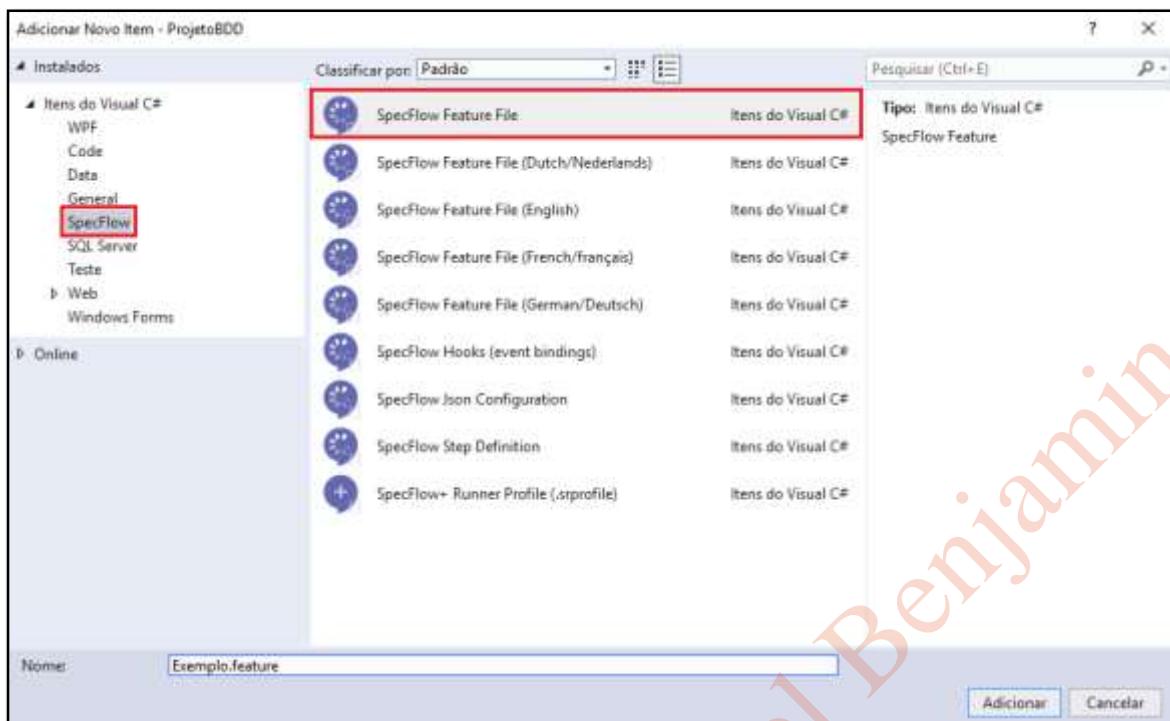
5.8 Specflow – Features e Step Definitions

Os arquivos features são os que armazenam os cenários, e o detalhamento do que cada passo deve fazer fica em arquivos step definitions que são classes onde a implementação do código será realizada.

Para criar uma feature clique com o botão direito no **Projeto > Adicionar > Novo Item**

Selecione **Specflow > Specflow Feature File**

Irei chamar essa primeira feature de *Exemplo.feature*



Este é o modelo que será criado:

```

Exemplo.feature
1 Feature: Exemplo
2   In order to avoid silly mistakes
3   As a math idiot
4   I want to be told the sum of two numbers
5
6   @mytag
7   Scenario: Add two numbers
8     Given I have entered 50 into the calculator
9     And I have entered 70 into the calculator
10    When I press add
11    Then the result should be 120 on the screen
12

```

Repare que é criado um arquivo **Exemplo.feature** e ‘dentro’ dele um arquivo **‘Exemplo.feature.cs’**

O arquivo Feature é que irá fazer o uso do Gherkin, e internamente o specflow gera um arquivo de classe ‘.cs’ correspondente a esta feature. O código interno é convertido de acordo com framework selecionado no **app.config**, e através dele os steps poderão ser lidos na classe de stepDefinition.

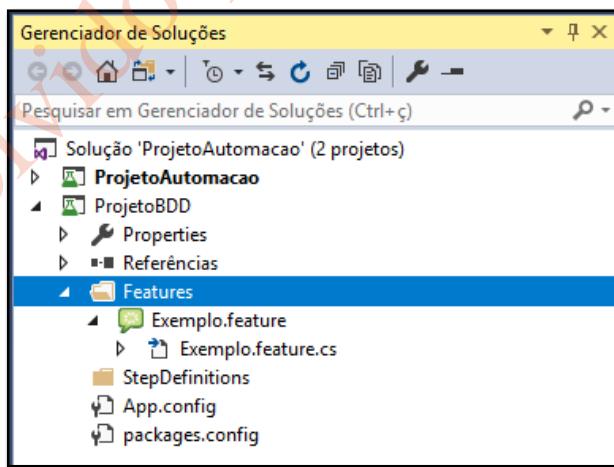
Este arquivo sempre será gerado automaticamente a cada alteração no arquivo .feature, então não é necessário fazer nenhum tipo de alteração nele.

```
1 //<!--
2 //<!-- auto-generated
3 // This code was generated by SpecFlow (http://www.specflow.org/).
4 // SpecFlow Version:2.4.0.0
5 // SpecFlow Generator Version:2.4.0.0
6 //
7 // Changes to this file may cause incorrect behavior and will be lost if
8 // the code is regenerated.
9 //-->
10 //
11 #region Designer generated code
12 #pragma warning disable
13 namespace ProjetoBDD.Features
14 {
15     using TechTalk.SpecFlow;
16
17     [System.CodeDom.Compiler.GeneratedCodeAttribute("TechTalk.SpecFlow", "2.4.0.0")]
18     [System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
19     [Microsoft.VisualStudio.TestTools.UnitTesting.TestClassAttribute()]
20     public partial class ExemploFeature
21     {
22
23         private static TechTalk.SpecFlow.ITestRunner testRunner;
24
25         private Microsoft.VisualStudio.TestTools.UnitTesting.TestContext _testContext;
26
27     }
28 }
```

Para organizar nosso projeto, crie duas novas pastas:

- Features
- StepDefinitions

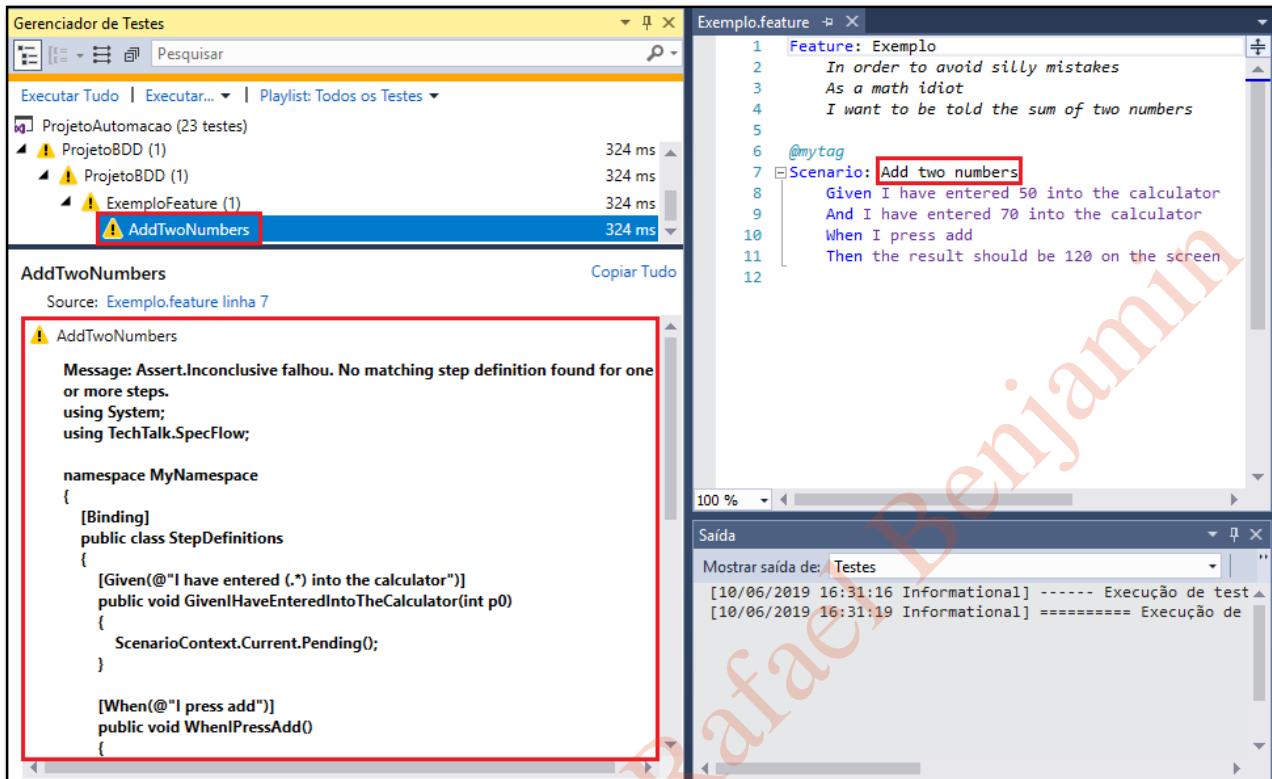
Em seguida mova este arquivo **Exemplo.feature** para dentro da pasta **Features**.



Abra seu gerenciador de testes, se tudo foi configurado corretamente o teste '**AddTwoNumbers**' deve ser exibido nele. Tente executá-lo.

Veja que após a execução o teste fica com um alerta e com a seguinte exceção: **No matching step definition found for one or more steps.**

Significa que o nenhum dos passos definidos nesta feature de exemplo foi implementado, logo abaixo deste alerta gera uma estrutura que é a classe contendo os steps a serem implementados.



Antes de iniciar a implementação destes steps, vamos mudar as frases para melhorar o exemplo, altere o cenário para o seguinte:

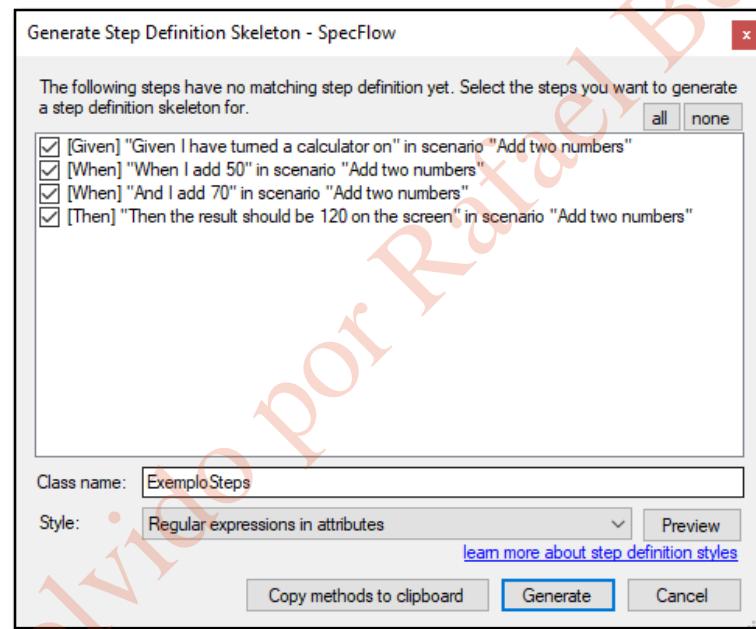
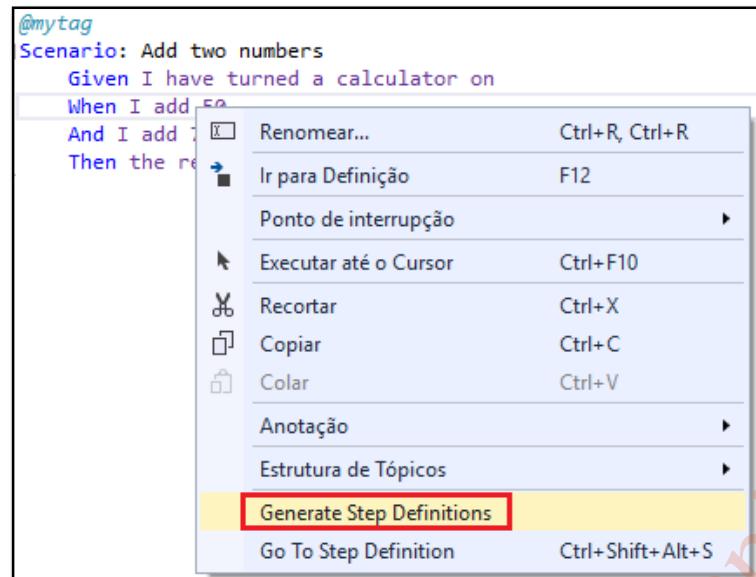
```
@mytag
Scenario: Add two numbers
  Given I have turned a calculator on
  When I add 50
  And I add 70
  Then the result should be 120 on the screen
```

Para começar a implementação desses passos temos algumas opções:

Podemos copiar esse código gerado e inserir em uma nova classe, é uma opção... Porém um pouco trabalhosa.

Vamos tentar outra opção mais fácil... Repare que os passos da feature estão em roxo, isso significa que ainda não foram implementados.

Clique com o botão direito em cima de qualquer um destes roxos e selecione **Generate Step Definition**



Neste menu podemos selecionar para quais passos podemos gerar as implementações. Para este exemplo vamos manter todos selecionados.

Também podemos selecionar qual o estilo dos métodos, por padrão use o que já está setado **Regular expressions in attributes**.

Mais em: <https://specflow.org/documentation/Step-Definition-Styles/>

Agora podemos escolher entre as duas opções:

Copy methods to clipboard : gera os métodos dos passos e copia para área de transferência

Generate : Gera uma nova classe de stepDefinition com os passos que foram selecionados

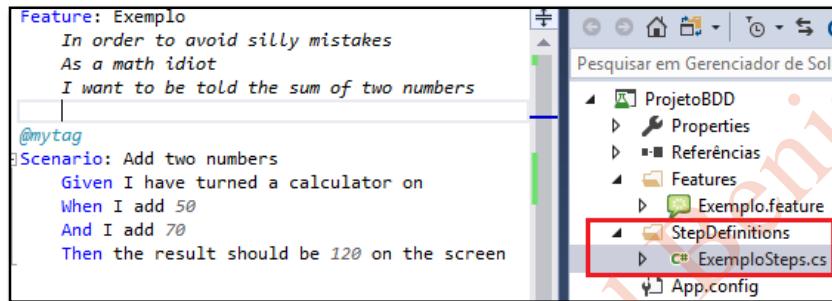
Vamos usar a Generate.

Clique em Generate, o prompt do Windows para salvar arquivo será exibido, navegue até a pasta deste novo projeto, procure a pasta StepDefinitions que criamos e salve este arquivo dentro dela.

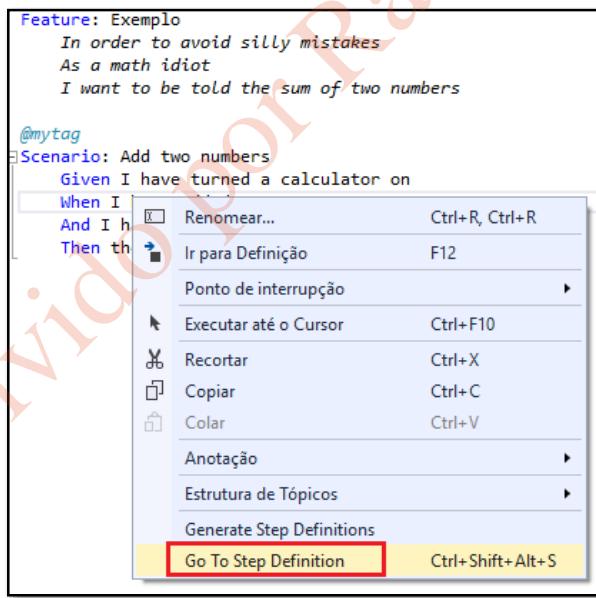
Por padrão, os nomes do arquivos terão o sufixo **Step** para identificação, e geralmente são o nome correspondente da feature como sufixo. Neste caso o arquivo terá o nome **ExemploSteps**.

Após salvar o arquivo, este será exibido automaticamente no projeto dentro da pasta StepDefinitions.

Repare também que agora que estes passos foram inseridos e encontrados pelo código, os passos na feature agora não estão mais roxos. Quando a feature encontra os steps estes ficam na cor preta.



Quando os passos estão definidos, podemos navegar até a implementação dos mesmos pela feature. Clique com o botão direito em algum passo da feature e selecione **Go To Step Definition**.



O código mudará o foco para a classe de stepDefinitions aonde este passo foi implementado:

```

using System;
using TechTalk.SpecFlow;

namespace ProjetoBDD.StepDefinitions
{
    [Binding]
    public class ExemploSteps
    {
        [Given(@"I have turned a calculator on")]
        public void GivenIHaveTurnedACalculatorOn()
        {
            ScenarioContext.Current.Pending();
        }

        [When(@"I add (.*)")]
        public void WhenIAdd(int p0)
        {
            ScenarioContext.Current.Pending();
        }

        [Then(@"the result should be (.*) on the screen")]
        public void ThenTheResultShouldBeOnTheScreen(int p0)
        {
            ScenarioContext.Current.Pending();
        }
    }
}

```

Repare que é uma estrutura de classe normal, porém acima da definição da classe temos um atributo `[Binding]` este atributo é o que permite que os passos definidos aqui sejam encontrados pelo arquivo de classe convertido pela feature.

Como geramos a classe automaticamente, a referência para o specflow que é a `using TechTalk.SpecFlow;` também já vem inclusa, sem ela não conseguimos usar o atributo `Binding`, ou os atributos que definem os passos.

Vamos ver um exemplo o seguinte passo na feature:

```
When I add 50
```

É transformado para a seguinte função:

```

[When(@"I add (.*)")]
public void WhenIAdd(int p0)
{
    ScenarioContext.Current.Pending();
}

```

Entendendo o código:

`[When(@"I add (.*)")]:` Na hora de gerar os steps optamos por manter o estilo dos métodos no padrão *Regular expressions in attributes*. O atributo se inicia com o a palavra chave usada neste caso o `When`, e entre parênteses a frase do passo em forma de expressão regular*. Repare que o número foi substituído pelo `(.*)`, que nas expressões regulares significa que quaisquer caracteres podem ser inseridos neste trecho e frase ainda será compreendida.

Caso você altere alguma parte desta frase, será entendido que é uma nova frase. Tente alterar esta, salve e veja o arquivo feature novamente, a frase original ficará roxa novamente.

`public void WhenIAdd(int p0):` Abaixo temos a definição do método, que também recebe o texto da frase em PascalCase. Este nome da função pode ser alterado e é independente da frase definida no atributo, porém é recomendável manter semelhante ao atributo pra organização. Repare também que esta função vem com um parâmetro `p0` do tipo `int`, significa que o trecho definido por expressão regular `(.*)` aceita qualquer valor, e como foi um número que estava definido na feature, veio com o tipo `int`. Ou seja neste caso o valor `50` entra com um parâmetro para a o método. O nome do parâmetro pode ser alterado,

e é recomendável que seja alterado para algo mais significativo em relação ao contexto da ação, neste caso poderia ser ‘value’.

```
{  
    ScenarioContext.Current.Pending();
```

} : O corpo do método que vem com este método Pending por padrão. Este significa que o método ainda não possui uma definição de ação, ele pode ser executado mas ao encontrar este método será exibido um alerta **One or more step definitions are not implemented yet**. Sempre remova esta chamada quando iniciar seu desenvolvimento.

*Pesquise mais sobre expressões regulares

<https://tableless.com.br/o-basico-sobre-expressoes-regulares/>

Repare que na feature existem 4 passos, mas no arquivo com o steps definitions foram gerados apenas 3.

```
When I add 50  
And I add 70
```

Os dois primeiros passos são exatamente a mesma ação, só variando o número de entrada que já está configurado para ser recebido como um parâmetro em nosso step. E sempre que usamos a palavra chave **And** em nossas features, internamente sempre será assumido que este and equivale a ultima palavra chave utilizada, neste caso o Given. Vamos implementar um exemplo simples desta calculadora, sem o uso de Selenium por enquanto para entender um pouco do fluxo no specflow.

```
[Binding]  
public class ExemploSteps  
{  
    int result = 0;  
  
    [Given(@"I have turned a calculator on")]  
    public void GivenIHaveTurnedACalculatorOn()  
    {  
        Console.WriteLine("Calculadora Ligada");  
    }  
  
    [When(@"I add (.*)")]  
    public void WhenIAAdd(int value)  
    {  
        result += value;  
    }  
  
    [Then(@"the result should be (.*) on the screen")]  
    public void ThenTheResultShouldBeOnTheScreen(int expectedResult)  
    {  
        Assert.AreEqual(expectedResult, result);  
    }  
}
```

Na frase adicionar temos que armazenar os valores recebidos, como esta frase pode ser executada mais de uma vez criei uma variável no escopo de classe para somar e armazenar estes valores. Não temos interface então apenas imprimei uma mensagem para ligar a calculadora. E na validação usei o `Assert.AreEqual` para comparar o valor calculado com o valor recebido no passo da feature.

Lembrando que o `Assert` vem da biblioteca do MsTest, então realize a sua importação:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

O resultado exibido imprime os passos executados, o tempo e os valores dos parâmetros.

```
Test Name: AddTwoNumbers
Test Outcome: ✅ Aprovado
Standard Output
-> Using app.config
Given I have turned a calculator on
Calculadora Ligada
-> done: ExemploSteps.GivenIHaveTurnedACalculatorOn() (0,0s)
When I add 50
-> done: ExemploSteps.WhenIAdd(50) (0,0s)
And I add 70
-> done: ExemploSteps.WhenIAdd(70) (0,0s)
Then the result should be 120 on the screen
-> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(120) (0,0s)
```

Tente alterar o valor esperado na feature, e veja que a mensagem de falha é exibida juntamente ao step na qual ela foi realizada.

```
Test Name: AddTwoNumbers
Test Outcome: ❌ Com falha
Message: Assert.AreEqual falhou. Esperado:<121>, Real:<120>.
Standard Output
-> Using app.config
Given I have turned a calculator on
Calculadora Ligada
-> done: ExemploSteps.GivenIHaveTurnedACalculatorOn() (0,0s)
When I add 50
-> done: ExemploSteps.WhenIAdd(50) (0,0s)
And I add 70
-> done: ExemploSteps.WhenIAdd(70) (0,0s)
Then the result should be 121 on the screen
-> error: Assert.AreEqual falhou. Esperado:<121>, Real:<120>.
```

5.9 Specflow – Alterando a língua e renomeando os passos

Antes de seguir com o próximo exemplo, vamos alterar a língua da feature para português. Ao alterar a língua o que vai ser alterado é a identificação das palavras chave na feature, dentro da step definition permanecerão em inglês.

<https://cucumber.io/docs/gherkin/reference/#spoken-languages>

Para isso inclua na primeira linha da feature:

```
#language: pt
```

Assim que incluir as palavras chave que estão em inglês deixarão de ser reconhecidas, agora altere para português:

```
#language: pt

Funcionalidade: Exemplo
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
```

```
@mytag
```

Cenario: Add two numbers

Dado I have turned a calculator on

Quando I add 50

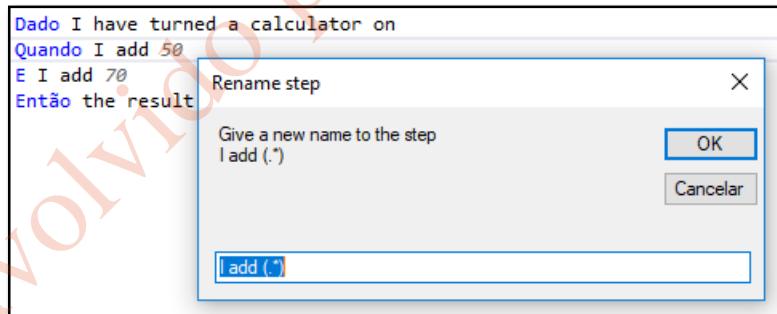
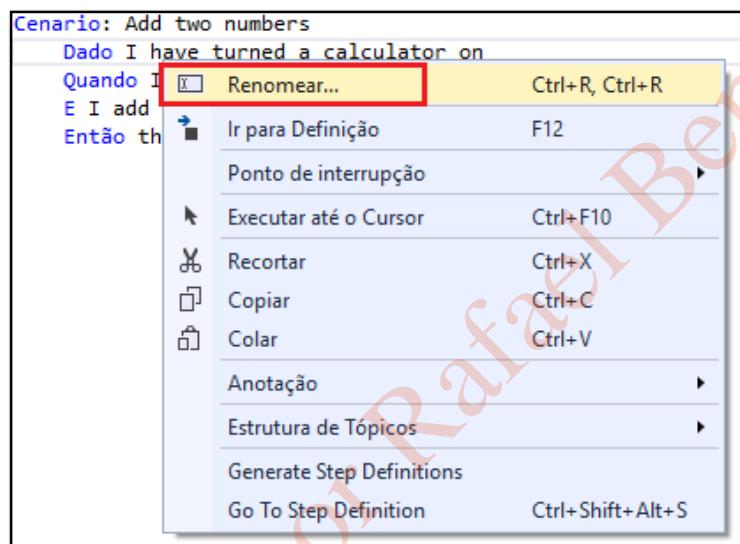
E I add 70

Então the result should be 120 on the screen

As frases são apenas um texto que é interpretado pelas expressões regulares então continua sendo reconhecidos. Mas apenas para entender a dinâmica, vamos traduzir as frases.

Para renomear um passo existente e atualizar seu método, clique com botão direito acima da frase e selecione

Renomear.



É exibido o texto mapeado no atributo do passo. Vamos alterar a frase que soma de :

I add (*)

Para:

somo (*)

Manterei a expressão regular para indicar a entrada de um dado, e clique em OK.

```
[When(@"somo (.*)")]
public void WhenIAdd(int value)
{
    result += value;
}
```

Note que apenas a frase mapeada no atributo foi alterada. Ao salvar este arquivo de step, veja que na feature ambos os passos com a mesma frase foram atualizados e já serão reconhecidos.

```
@mytag
Cenario: Add two numbers
Dado I have turned a calculator on
Quando somo 50
E somo 70
Então the result should be 120 on the screen
```

Tente atualizar os demais passos e veja a solução proposta abaixo.

Minha feature ficou da seguinte forma:

```
#language: pt

Funcionalidade: Exemplo
  Como um analista de teste
  Quero escrever cenários de exemplo
  Para aprender a usar o specflow

@mytag
Cenario: Somar dois números
  Dado que liguei a calculadora
  Quando somo 50
  E somo 70
  Então o resultado na tela deve ser 120
```

Traduzi os cenários e o nome do cenário também. Na parte descritiva abaixo de Funcionalidade atualizei a descrição para o que iremos fazer na feature, lembrando que neste trecho qualquer coisa pode ser escrita, mas é recomendável apenas informações ou anotações que sejam relevantes a sua feature.

Talvez a sua frase tenha ficado diferente, mas não tem problema desde que ela transmita o mesmo significado.

Obs: Não costumo usar o "Eu" antes das frases pois parece ficar muito repetitivo na leitura. Tento manter um padrão de manter as frases de "Dado" no passado, "Quando" no presente e "Então" no futuro. Não é uma regra, mas achei que ajuda na leitura. Esses pontos e outros como tempos verbais por exemplo podem ser acordados com o seu time durante a escrita dos cenários.

E meu stepDefinition:

```
[Binding]
public class ExemploSteps
{
    int result = 0;

    [Given(@"que liguei a calculadora")]
    public void GivenIHaveTurnedACalculatorOn()
    {
        Console.WriteLine("Calculadora Ligada");
    }
}
```

```

    }

    [When(@"somar (.*)")]
    public void WhenIAdd(int value)
    {
        result += value;
    }

    [Then(@"o resultado na tela deve ser (.*)")]
    public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
    {
        Assert.AreEqual(expectedResult, result);
    }
}

```

5.10 Specflow – Tabelas

Vamos ver como usar tabelas dentro dos cenários, as tabelas são compostas por colunas e dados e podemos extrair estes dados e usa-los dentro dos cenários.

Abaixo do cenário ‘Somar dois números’ que acabamos de fazer cole este novo cenário para exemplificar o uso de uma tabela.

```

Cenario: Realizar um cadastro de usuário apenas com os dados obrigatórios
#Dado que accesei minha aplicação
#E accesei a área de cadastro de usuários
Quando preencher apenas os dados obrigatórios
| Nome | Idade | Telefone | Email |
| Jose | 44 | 11963822718 | jose@teste.com |
#E realizar o cadastro
#Então o usuário deve ser cadastrado

```

Mantive os demais passos com um comentário inserindo uma # antes do passo. Isso faz com que estes passos não sejam executados, e como ainda não iremos interagir com a tela vamos manter o foco apenas neste passo que contém o exemplo da tabela.

Neste passo ‘Quando preencher apenas os dados obrigatórios’ inclui uma tabela abaixo para representar quais dados gostaria que fossem obtidos e utilizados nele.

Para gerar o método correspondente, clique o botão direito no passo e selecione **Generate Step Definitions**. Agora ao invés de gerar uma nova classe de stepDefinition, vamos apenas copiar o método gerado e colar dentro da classe existente. O ideal é sempre tentar manter os passos relativos a mesma feature dentro do mesmo arquivo de step definition.

Clique em **Copy methods to clipboard**. Agora o método gerado está disponível para ser colado em algum lugar. Abra o arquivo de step definition e cole o método antes ou depois do método When já existente. Você pode copiar o método em qualquer lugar da classe de stepDefinition, mas apenas para fins de organização vamos tentar agrupar os passos por tipo, mantendo os Given, When e Then juntos.

A classe deve ficar assim:

```

[Binding]
public class ExemploSteps
{
    int result;

    [Given(@"que liguei a calculadora")]

```

```

public void GivenIHaveTurnedACalculatorOn()
{
    Console.WriteLine("Calculadora Ligada");
}

[When(@"preencher apenas os dados obrigatórios")]
public void QuandoPreencherApenasOsDadosObrigatorios(Table table)
{
    ScenarioContext.Current.Pending();
}

[When(@"somo (.*)")]
public void WhenIAdd(int value)
{
    result += value;
}

[Then(@"Então o resultado na tela deve ser (.*)")]
public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
{
    Assert.AreEqual(expectedResult, result);
}

```

Veja que agora o parâmetro do método gerado é do tipo *Table*.

Para que possamos ler as colunas, precisamos criar um objeto que contenha como atributos as colunas criadas na tabela.

Clique com o botão direito no **Projeto > Adicionar > Classe**

Vou chamar esta classe de DadosUsuario.cs

Altere a classe para public e crie os atributos para as colunas correspondentes da seguinte forma:

```

public class DadosUsuario
{
    public string nome { get; set; }
    public int idade { get; set; }
    public string telefone { get; set; }
    public string email { get; set; }
}

```

O nome das propriedades deve ser igual ao escrito na tabela, e os seus tipos devem corresponder ao que será lido, por exemplo, não posso deixar o nome como tipo int.

Agora de volta a classe de stepDefinition, inclua o seguinte import que será necessário para poder criar a instância do tipo que criamos agora:

```
using TechTalk.SpecFlow.Assist;
```

Para poder ler um determinado tipo da tabela, vamos usar o comando:

```
table.CreateInstance<Tipo>();
```

Então para ler a nossa classe vamos usar o seguinte:

```
table.CreateInstance<DadosUsuario>();
```

Com isso a tabela lê os dados correspondentes a cada coluna e obtém seus valores, mas para podermos acessar de fato os valores vamos salvar estes valores em uma instância do tipo *DadosUsuario*.

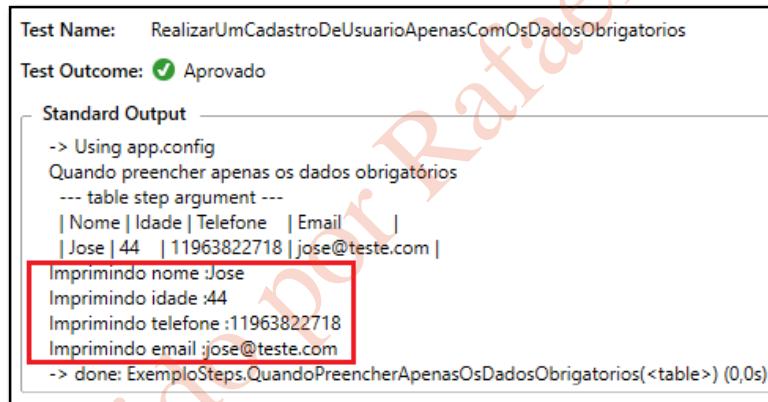
```
DadosUsuario dados = table.CreateInstance<DadosUsuario>();
```

Agora posso acessar os valores pelo objeto *dados* que criamos acima. Como exemplo vou fazer com que a função imprima estes valores na saída do teste:

```
[When(@"preencher apenas os dados obrigatórios")]
public void QuandoPreencherApenasOsDadosObrigatorios(Table table)
{
    DadosUsuario dados = table.CreateInstance<DadosUsuario>();

    Console.WriteLine("Imprimindo nome :" + dados.nome);
    Console.WriteLine("Imprimindo idade :" + dados.idade);
    Console.WriteLine("Imprimindo telefone :" + dados.telefone);
    Console.WriteLine("Imprimindo email :" + dados.email);
}
```

Execute este teste e veja a saída:



Agora vamos fazer outro exemplo porém lendo dados de várias linhas da tabela.

Altere a feature para o seguinte, comentando este passo e tabela que acabamos de executar e incluindo outro abaixo com várias linhas na tabela:

```
Cenario: Realizar um cadastro de usuário apenas com os dados obrigatórios
#Dado que accesei minha aplicação
#E accesei a área de cadastro de usuários
#Quando preencher apenas os dados obrigatórios
# | Nome | Idade | Telefone | Email |
# | Jose | 44 | 11963822718 | jose@teste.com |
Quando preencher apenas os dados obrigatório de vários usuários
| Nome | Idade | Telefone | Email |
| Jose | 44 | 11963822718 | jose@teste.com |
| Carla | 23 | 11956218312 | carla@teste.com |
| Roger | 31 | 1136956486 | roger@teste.com |
#E realizar o cadastro
#Então o usuário deve ser cadastrado
```

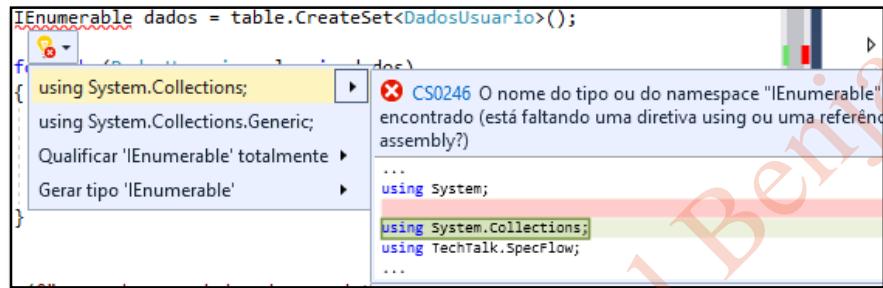
Como a frase é outra, gere um novo método da mesma forma que fizemos da última vez, copiando o método dentro do step definition.

A forma de fazer será semelhante, como mantive a tabela com os mesmos dados podemos usar a mesma classe para ler os valores, mas o que muda é que no lugar de `table.CreateInstance<Tipo>()` usarei o `table.CreateSet<Tipo>();`

O retorno deste método é do tipo `IEnumerable<Tipo>`, ou seja é uma lista com esses tipos.



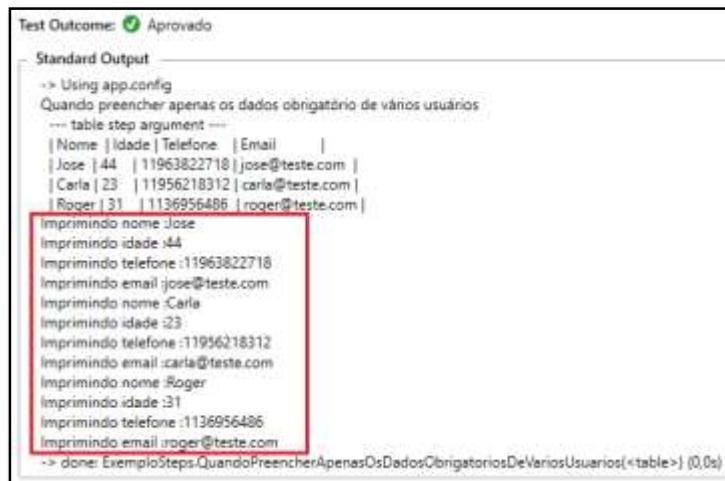
Então vamos criar uma variável do tipo `IEnumerable`, que por sua vez precisa da referência para o `System.Collections`



Depois que recebermos os valores de cada linha, vamos imprimir usando um foreach para percorrer todos os valores a cada linha. A função então fica da seguinte forma:

```
[When(@"preencher apenas os dados obrigatório de vários usuários")]  
public void QuandoPreencherApenasOsDadosObrigatoriosDeVariosUsuarios (Table table)  
{  
    IEnumerable dados = table.CreateSet<DadosUsuario>();  
  
    foreach (dadosUsuario valor in dados)  
    {  
        Console.WriteLine("Imprimindo nome :" + valor.nome);  
        Console.WriteLine("Imprimindo idade :" + valor.idade);  
        Console.WriteLine("Imprimindo telefone :" + valor.telefone);  
        Console.WriteLine("Imprimindo email :" + valor.email);  
    }  
}
```

Execute o teste novamente e veja os dados sendo exibidos:



5.11 Specflow – Lendo Tabelas com Assist.Dynamic

Vimos como obter os valores a partir de uma tabela, mas sempre que tivermos uma nova tabela, teremos que criar uma nova classe com seus respectivos atributos representando as colunas. Para contornar isso temos a biblioteca Assist.Dynamic que nos permite ler as colunas dinamicamente.

Primeiro instale o seguinte pacote do NuGet:

- **SpecFlow.Assist.Dynamic v1.3.1**

Temos que instalar a versão 1.3.1 pois as mais recentes só funcionam no specflow a partir da versão 3.

Em seguida na sua classe de stepDefinition inclua a seguinte referência

```
using SpecFlow.Assist.Dynamic;
```

Em nosso arquivo Exemplo.feature inclua mais um cenário:

```
Cenario: Realizar o cadastro de vários produtos
  #Dado que acessei minha aplicação
  #E acessei a área de cadastro de produtos
  Quando preencher os dados dos produtos
    | Id | Nome   | Valor |
    | 10 | Vassoura | 5,00 |
    | 11 | Cadeira  | 20,00 |
    | 12 | Mesa     | 35,00 |
  #E realizar o cadastro
  #Então os produtos devem ser cadastrados
```

Crie o passo correspondente dentro da step Definition.

Agora podemos ter acesso as funções que são semelhantes as que usamos anteriormente, porém através da biblioteca dynamic.

```
table.CreateDynamicInstance(); - Para ler uma linha
table.CreateDynamicSet(); - Para ler várias linhas
```

A função ficará desta forma:

```
[When(@"preencher os dados dos produtos")]
public void QuandoPreencherOsDadosDosProdutos(Table table)
{
    var produtos = table.CreateDynamicSet();

    foreach(var produto in produtos)
    {
        Console.WriteLine("Imprimindo id :" + produto.Id);
        Console.WriteLine("Imprimindo nome :" + produto.Nome);
        Console.WriteLine("Imprimindo valor :" + produto.Valor);
    }
}
```

O detalhe é que agora não precisamos criar uma classe que contém os atributos correspondentes as colunas. Na hora de acessar o atributo use sempre o mesmo nome da coluna e da forma como está escrito na feature.

Exemplo, se alterar o `produto.Id` para `produto.id` (tudo minúsculo) a coluna não será reconhecida e o valor não poderá ser lido.

```

Test Name: RealizarUmCadastroDeProdutos
Test Outcome: ✅ Aprovado

Standard Output
-> Using app.config
Quando preencher os dados dos produtos
--- table step argument ---
| Id | Nome | Valor |
| 10 | Vassoura | 5,00 |
| 11 | Cadeira | 20,00 |
| 12 | Mesa | 35,00 |
Imprimindo id :10
Imprimindo nome :Vassoura
Imprimindo valor :5
Imprimindo id :11
Imprimindo nome :Cadeira
Imprimindo valor :20
Imprimindo id :12
Imprimindo nome :Mesa
Imprimindo valor :35
-> done: ExemploSteps.QuandoPreencherOsDadosDosProdutos(<table>) (0,3s)

```

5.12 Specflow – Esquema do Cenário

Tomando como base o cenário do exemplo anterior:

```

Cenario: Realizar o cadastro de vários produtos
#Dado que accesei minha aplicação
#E accesei a área de cadastro de produtos
Quando preencher os dados dos produtos
    | Id | Nome | Valor |
    | 10 | Vassoura | 5,00 |
    | 11 | Cadeira | 20,00 |
    | 12 | Mesa | 35,00 |
#E realizar o cadastro
#Então os produtos devem ser cadastrados

```

Ainda não temos uma interface para testar, mas apenas para fins de exemplo, imagine que neste cenário o passo '*Quando preencher os dados dos produtos*' irá entrar com os valores da linha de uma vez e realizar um único cadastro (assumindo que isto seja possível, como por exemplo expandir os campo para um novo produto nesta tela de cadastro).

Agora vamos imaginar que a tela de cadastro está limitada a um produto por vez. Para isso podemos usar o '**Esquema do Cenário**', onde o teste pode reutilizar os mesmos tipos de dados e passos dentro um cenário:

Reescrevi o cenário da seguinte forma:

```

Esquema do Cenário: Realizar o cadastro de um produto
#Dado que accesei minha aplicação
#E accesei a área de cadastro de produtos
Quando preencher o <Id> , <Nome> e <Valor> do produto
E informar o usuário <Usuário>
#E realizar o cadastro
#Então o produto deve ser cadastrado
Exemplos:
    | Nome | Id | Valor | Usuário |
    | Vassoura | 10 | 5,00 | Amanda |
    | Cadeira | 11 | 20,00 | Tulio |
    | Mesa | 12 | 35,00 | Maria |

```

Inclui o passo de informar o usuário apenas para demonstrar que os dados podem ser lidos em qualquer passo.

Para poder usar os exemplos é obrigatório que use a palavra chave **Esquema do Cenário** acompanhada dos **Exemplos**.

Quando quiser referência aonde quero usar os valores das colunas, devo apontar usando a forma <coluna> em qualquer frase deste cenário. Neste ponto o valor da coluna será utilizado.

Se tentarmos gerar o método para o passo aonde estou cadastrando o produto, o seguinte código é gerado:

```
[When(@"preencher o (.*) , Vassoura e (.*) do produto")]
public void QuandoPreencherOVassouraEDoProduto(int p0, Decimal p1)
{
    ScenarioContext.Current.Pending();
}
```

O specflow não conseguiu interpretar um dos parâmetros e trouxe o texto Vassoura, isto pode acontecer as vezes.

Para evitar esse tipo de problema **SEMPRE use aspas duplas nos parâmetros**.

Seja incluindo as aspas duplas nos passos desta forma : “<coluna>”

Esquema do Cenário: Realizar o cadastro de um produto

```
#Dado que acessei minha aplicação
#E acessei a área de cadastro de produtos
Quando preencher o "<Id>" , "<Nome>" e "<Valor>" do produto
E informar o usuário "<Usuário>"
#E realizar o cadastro
#Então os produtos devem ser cadastrados
Exemplos:
```

Nome	Id	Valor	Usuário
Vassoura	10	5,00	Amanda
Cadeira	11	20,00	Tulio
Mesa	12	35,00	Maria

Ou deixando as aspas duplas direto em cada valor dos exemplos:

Esquema do Cenário: Realizar o cadastro de um produto

```
#Dado que acessei minha aplicação
#E acessei a área de cadastro de produtos
Quando preencher o <Id> , <Nome> e <Valor> do produto
E informar o usuário <Usuário>
#E realizar o cadastro
#Então o produto deve ser cadastrado
Exemplos:
```

Nome	Id	Valor	Usuário
"Vassoura"	"10"	"5,00"	"Amanda"
"Cadeira"	"11"	"20,00"	"Tulio"
"Mesa"	"12"	"35,00"	"Maria"

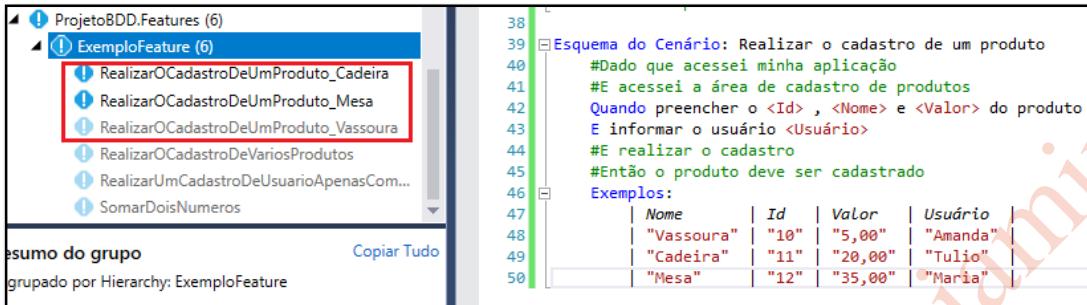
Isto também pode ser acordado com seu time para definir qual fica melhor em seu projeto.

Agora ao gerar o método do passo novamente, os três parâmetros são identificados corretamente:

```
[When(@"preencher o """(.*)" , """(.*)" e """(.*)" do produto")]
public void QuandoPreencherOVassouraEDoProduto(int p0, string p1, Decimal p2)
{
    ScenarioContext.Current.Pending();
}
```

Os parâmetros são sempre incluídos na ordem que aparecem na frase, então o specflow identificou primeiro o Id como um int, em seguida o Nome como uma string e por fim o Valor como um decimal. Imaginando que é o preenchimento de um cadastro, seria possível, por exemplo, alterar os três para que sejam lidos como string.

Note que agora foram gerados três cenários, um com final Cadeira, outro Mesa, e outro Vassoura. O specflow gera os cenários com base no número de linhas que são incluídas nos exemplos. E usa os valores da primeira coluna para diferenciar os testes.



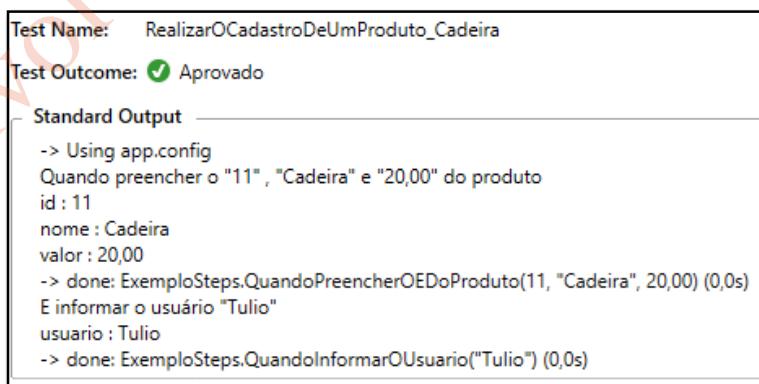
Apenas para visualização dos resultados, mantive os passos imprimindo os dados recebidos:

```
[When(@"preencher o ""(.*)"" , ""(.*)"" e ""(.*)"" do produto")]
public void QuandoPreencherOEDoProduto(int id, string nome, Decimal valor)
{
    Console.WriteLine("id : " + id);
    Console.WriteLine("nome : " + nome);
    Console.WriteLine("valor : " + valor);
}

[When(@"informar o usuário ""(.*)""")]
public void QuandoInformarOUSuario(string usuario)
{
    Console.WriteLine("usuario : " + usuario);
}
```

Agora tente executar um ou todos os testes dentro deste esquema do cenário. Lembrando que mesmo executando os três, eles são considerados como testes individuais.

Resultado da execução de um dos testes:



Como agora temos um cenário que pode gerar mais cenários dependendo dos dados que são inputados, temos a possibilidade de executar cada teste individualmente, e em cada execução serão lidos apenas os dados correspondentes à sua linha.

Também podemos criar novos cenários apenas incluindo novas linhas de dados.

5.13 Specflow – Scenario Context

Referência: <https://specflow.org/documentation/ScenarioContext/>

O Scenario Context provém algumas informações e ações para nossos cenários.

- ScenarioContext.Current.Pending()

Já vimos o ScenarioContext.Current.Pending(); que serve para indicar que o cenário ainda não possui alguma ação implementada.

E informações referentes ao cenário atual como por exemplo:

- ScenarioContext.Current.ScenarioInfo.Title

O ScenarioContext.Current.ScenarioInfo.Title exibe o nome do cenário em execução.

Em algum teste tente exibir o nome incluindo o comando:

```
Console.WriteLine("Nome do cenário: " + ScenarioContext.Current.ScenarioInfo.Title);
```

- ScenarioContext.Current.CurrentScenarioBlock

O ScenarioContext.Current.CurrentScenarioBlock exibe o tipo de passo em execução. Se é um Given, um When ou um Then.

Em algum teste tente exibir o nome incluindo o comando:

```
Console.WriteLine("Tipo de passo: " + ScenarioContext.Current.CurrentScenarioBlock);
```

- ScenarioContext.Current

Referência: <https://specflow.org/documentation/ScenarioContext.Current/>

O ScenarioContext também pode ser utilizado como um dicionário de dados e compartilhar estas informações entre step definitions.

Até o momento criamos um cenário que possui todos os dados necessários dentro do mesmo arquivo de stepDefinition. Se caso precisarmos acessar um valor que está em outro arquivo step podemos utilizar o ScenarioContext.Current["valor"] para armazenar o valor em um dicionário do specflow que compartilha estes valores, também pode armazenar objetos.

Lembrando que a feature pode encontrar o passo em qualquer arquivo stepdefinition desde que possua o atributo [Binding] em sua classe.

Para exemplificar vamos incluir um novo cenário dentro da mesma feature para a calculadora com um passo extra que irá realizar uma subtração e criar outro arquivo de step definition e implementar um step nele.

```
Cenario: Somar um número e subtrair um
        Dado que liguei a calculadora
        Quando somo 50
        E subtraio 30
        Então o resultado na tela deve ser 20
```

Agora gere este novo passo em um novo arquivo step definition.

The screenshot shows a Visual Studio interface. On the left is a code editor containing a feature file named 'Exemplo.feature'. The file contains two scenarios: one for addition and one for subtraction. On the right is a Solution Explorer window showing a project named 'ProjetoAutomacao' with files like 'ProjetoBDD.cs', 'Properties', 'Referências', 'Features', 'StepDefinitions', and configuration files.

```

@mytag
Cenario: Somar dois números
    Dado que liguei a calculadora
    Quando somo 50
    E soma 70
    Então Então o resultado na tela deve ser 120

Cenario: Somar um número e subtrair um
    Dado que liguei a calculadora
    Quando somo 50
    E subtraio 30
    Então Então o resultado na tela deve ser 20

```

Mesmo o passo estando em outro arquivo ele é encontrado pela feature.

Dentro deste novo arquivo de step definiton que no meu exemplo chamei de **ExemploDoisSteps**, fiz a implementação da mesma forma que o passo de adição, porém agora subtraindo do resultado:

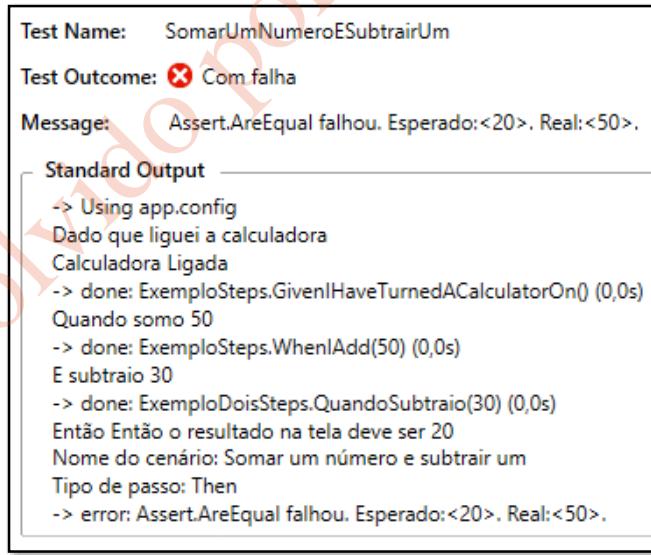
```

[Binding]
public class ExemploDoisSteps
{
    int result;

    [When(@"subtraio (.*)")]
    public void QuandoSubtraio(int value)
    {
        result -= value;
    }
}

```

Tente executar este teste, o resultado deve ser o seguinte:



O passo com a subtração não foi realizado, já que o variável resultado criada na outra classe **ExemploSteps** não é visível para esta classe. Podemos contornar esse problema usando o dicionário de dados do `ScenarioContext.Current`.

Vamos atualizar os steps de somar e subtrair para o seguinte:

ExemploSteps:

Aqui no passo de somar após atualizar a variável de resultado, iremos salvar seu valor dentro um registro do dicionário chamado também de ‘result’

```
[When(@"somo (.*)")]
public void WhenIAAdd(int value)
{
    result += value;
    ScenarioContext.Current["result"] = result;
}
```

E no passo de validação final, teremos quer obter o valor armazenado no resultado após todas a operações

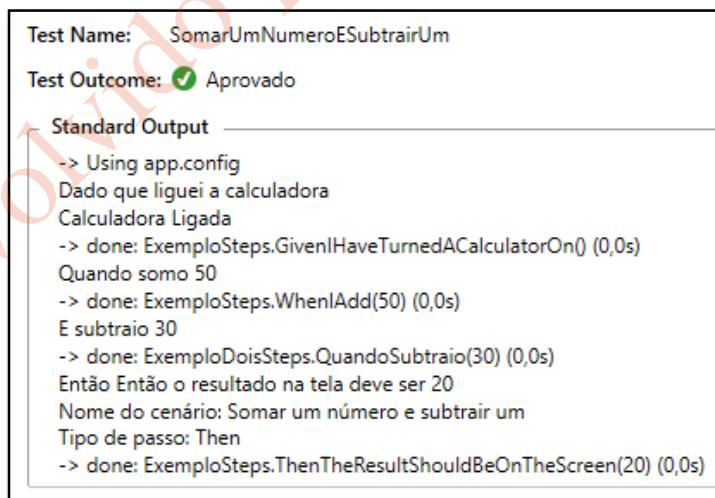
```
[Then(@"o resultado na tela deve ser (.*)")]
public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
{
    result = Convert.ToInt32(ScenarioContext.Current["result"]);
    Assert.AreEqual(expectedResult, result);
}
```

ExemploDoisSteps:

E aqui no passo da subtração temos que primeiro obter o valor do resultado que foi salvo após a operação de soma, realizar a subtração e salvar o valor atualizado no dicionário.

```
[When(@"subtraio (.*)")]
public void QuandoSubtraio(int value)
{
    result = Convert.ToInt32(ScenarioContext.Current["result"]);
    result -= value;
    ScenarioContext.Current["result"] = result;
}
```

Agora tente executar os passos novamente. Caso queira debugue os passos para ver os valores sendo atualizados.



Vamos incluir outro cenário alterando a ordem dos passos e verificar se irá funcionar

```
Cenario: Subtrair um número e somar um
Dado que liguei a calculadora
Quando subtraio 25
E somo 10
Então o resultado na tela deve ser -15
```

Test Name: SubtrairUmNumeroESumarUm

Test Outcome: ✘ Com falha

Message: Test method ProjetoBDD.Features.ExemploFeature.SubtrairUmNumeroESumarUm threw exception:
System.Collections.Generic.KeyNotFoundException: A chave fornecida não estava presente no dicionário.

Standard Output

```

-> Using app.config
Dado que liguei a calculadora
Calculadora Ligada
-> done: ExemploSteps.GivenIHaveTurnedACalculatorOn() (0,0s)
Quando subtraio 25
-> error: A chave fornecida não estava presente no dicionário.
E somo 10
-> skipped because of previous errors
Então Então o resultado na tela deve ser -15
-> skipped because of previous errors

```

Após a execução é exibido o erro: **System.Collections.Generic.KeyNotFoundException: A chave fornecida não estava presente no dicionário.** O erro ocorreu no passo da subtração:

```

[When(@"subtraio (.*)")]
public void QuandoSubtraio(int value)
{
    result = Convert.ToInt32(ScenarioContext.Current["result"]);
    result -= value;
    ScenarioContext.Current["result"] = result;
}

```

Agora o passo não funcionou pois primeiro ele tenta acessar o valor do dicionário e salvar na variável *result*, mas como não executamos o passo da soma, ele ainda não foi criado.

Podemos tentar solucionar da seguinte forma, na classe **ExemploSteps** atualize os métodos:

Vamos fazer com que o passo inicial que liga calculadora inicialize o dicionário o variável *result*, que é 0.

```

[Given(@"que liguei a calculadora")]
public void GivenIHaveTurnedACalculatorOn()
{
    Console.WriteLine("Calculadora Ligada");
    ScenarioContext.Current["result"] = result;
}

```

E no passo da soma, antes de realizar a operação vamos deixar igual ao passo da subtração e fazer que o resultado também seja atualizado pelo valor que estiver no dicionário.

Assim tanto na soma quanto na subtração, os valores serão primeiramente atualizados localmente recebendo o valor salvo no dicionário e depois que realizar as ações, atualizo este valor novamente, garantindo assim que independente da ordem de chamada dos passos posso reutilizá-los.

```

[When(@"somo (.*)")]
public void WhenIAdd(int value)
{
    result = Convert.ToInt32(ScenarioContext.Current["result"]);
    result += value;
}

```

```

        ScenarioContext.Current["result"] = result;
    }
}

```

Agora na nova execução deste cenário temos o status de aprovado:

Test Name:	SubtrairUmNumeroESomarUm
Test Outcome:	Aprovado
Standard Output	<pre> -> Using app.config Dado que liguei a calculadora Calculadora Ligada -> done: ExemploSteps.GivenIHaveTurnedACalculatorOn() (0,0s) Quando subtraio 25 -> done: ExemploDoisSteps.QuandoSubtrai(25) (0,0s) E somo 10 -> done: ExemploSteps.WhenIAdd(10) (0,0s) Então Então o resultado na tela deve ser -15 Nome do cenário: Subtrair um número e somar um Tipo de passo: Then -> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(-15) (0,0s) </pre>

Lá no início depois de gerar a feature, eu alterei a forma deste cenário da calculadora já pensando na possível reutilização de seus passos. Por isso além de apenas escrever os passos do cenário, é necessário escrevê-los de uma maneira com que eles possam ser modulares e reaproveitados para a automação.

- Sugestão – Assista ao vídeo : Meetup - Como descrever cenários de teste utilizando Gherkin de forma correta - https://www.youtube.com/watch?v=SAmwMD1_xJg

5.14 Specflow – Context Injection

Referência: <https://specflow.org/documentation/Context-Injection/>

O specflow possui uma implementação de um pequeno framework de injeção de dependência para as instâncias das classes nos cenários. Serve para compartilhar dados entre as classes de binding.

Para usar essa injeção de contexto é necessário:

1. Criar uma classe POCO(Plain old CLR objects), é uma classe composta apenas por métodos e atributos nativos do .NET Framework
2. Definir-las como parâmetros do construtor em cada classe com Binding que for necessário
3. Salve o argumento do construtor em campos de instância, para que possa usá-los nas step definitions

Antes de iniciarmos, vamos criar outro cenário para este exemplo dentro de nossa feature e implementar um passo que fornece os dados em uma step definition, e o outro passo na outra step definition para ver os dados sendo compartilhados:

```

Cenario: Realizar um cadastro de uma fruta
    #Dado que accesei minha aplicação
    Quando cadastrar uma fruta
        | Fruta | Quantidade | Fornecedor |
        | Abacaxi | 8 | Atacadão |
    Então devo visualizar os dados preenchidos

```

Para o passo ‘Quando cadastrar uma fruta’, gere seu step definition dentro da classe **ExemploSteps**. E para para o ‘Então devo visualizar os dados preenchidos’, gere seu step definition dentro da classe **ExemploDoisSteps**.

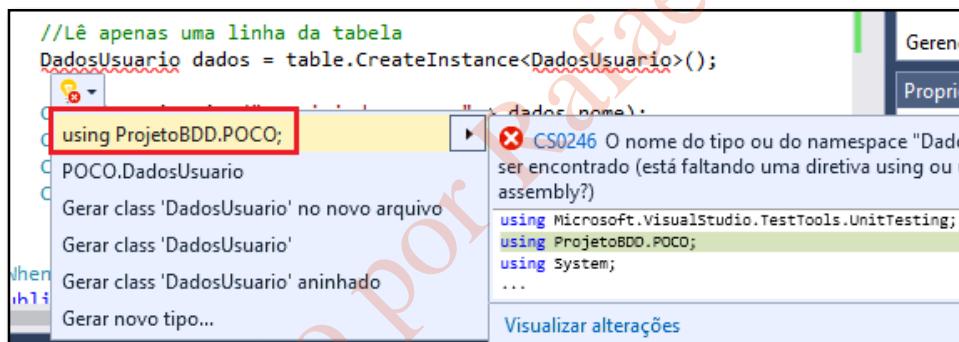
Primeiro passo – Criar a classe simples contendo os dados que iremos compartilhar.

Crie uma pasta chamada ‘POCO’ e dentro dela crie uma classe chamada ‘**DadosFruta**’, e altere a visibilidade desta classe para pública:

```
public class DadosFruta
{
    public string fruta { get; set; }
    public int quantidade { get; set; }
    public string fornecedor { get; set; }
}
```

Mova a classe que criados anteriormente ‘**DadosUsuario**’ para esta pasta também, apenas para organização. Assim que mover atualize o namespace desta classe para **NomeDoSeuProjeto.POCO** , no meu caso ficará como **ProjetoBDD.POCO**

Agora que esta classe foi alterada, dentro do ExemploSteps, atualize sua referência incluindo o using deste novo namespace.



Segundo Passo – Criar um construtor e atribuir a classe POCO a ser compartilhada como seu parâmetro

```
public readonly DadosFruta dadosFruta;

public ExemploSteps(DadosFruta dadosFruta)
{
    this.dadosFruta = dadosFruta;
}
```

Coplando o construtor acima, e campo de instância que será de apenas leitura o ExemploStep deve ficar da seguinte forma:

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using ProjetoBDD.POCO;
using System;
using System.Collections;
using TechTalk.SpecFlow;
using TechTalk.SpecFlow.Assist;

namespace ProjetoBDD.StepDefinitions
{
    [Binding]
    public class ExemploSteps
    {
        int result;

        public readonly DadosFruta dadosFruta;

        public ExemploSteps(DadosFruta dadosFruta)
        {
            this.dadosFruta = dadosFruta;
        }

        [Given(@"que liguei a calculadora")]
        public void GivenIHaveTurnedACalculatorOn()
        {
            Console.WriteLine("Calculadora Ligada");
            ScenarioContext.Current["result"] = result;
        }

        [When(@"somo (.*)")]
        public void WhenToAdd(int value)
    }
}

```

Terceiro Passo – Salvar os dados do objeto a ser compartilhado na variável de instância

Agora precisamos obter os dados necessários, neste caso a linha da coluna que criamos em nossa feature.

Vamos ler essa linha usando o que foi apresentado no Assist.Dynamic. No exemplo dado lemos várias linhas, aqui para ler apenas uma linha vamos usar desta forma:

```

[When(@"cadastrar uma fruta")]
public void QuandoCadastrarUmaFruta(Table table)
{
    dynamic produtos = table.CreateDynamicInstance();

    dadosFruta.fruta = produtos.Fruta;
    dadosFruta.quantidade = produtos.Quantidade;
    dadosFruta.fornecedor = produtos.Fornecedor;
}

```

Assim os dados que lermos da tabela, serão usado para preencher nosso objeto dadosFruta que por sua vez alimentará o construtor.

Com estes passos podemos ler os dados e compartilhá-los para outra classe de step definition, a qual vamos fazer agora.

Na classe **ExemploDoisSteps** inclua o mesmo construtor e variável de instância. Lembrando para alterar o nome do construtor de acordo com a classe.

```

using ProjetoBDD.POCO;
using System;
using TechTalk.SpecFlow;

namespace ProjetoBDD.StepDefinitions
{
    [Binding]
    public class ExemploDoisSteps
    {
        int result;

        public readonly DadosFruta dadosFruta;

        public ExemploDoisSteps(DadosFruta dadosFruta)
        {
            this.dadosFruta = dadosFruta;
        }

        [When(@"subtraio (.*)")]
        public void QuandoSubtraio(int value)
        {
            result = Convert.ToInt32(ScenarioContext.Current["result"]);
            result -= value;
            ScenarioContext.Current["result"] = result;
        }

        [Then(@"devo visualizar os dados preenchidos")]
        public void EntaoDevoVisualizarOsDadosPreenchidos()
        {
            ScenarioContext.Current.Pending();
        }
    }
}

```

E finalmente para ler os dados que vieram da outra classe podemos chamar as propriedades do objeto que veio pelo construtor:

```

[Then(@"devo visualizar os dados preenchidos")]
public void EntaoDevoVisualizarOsDadosPreenchidos()
{
    Console.WriteLine("Fruta: " + dadosFruta.fruta);
    Console.WriteLine("Qtde " + dadosFruta.quantidade);
    Console.WriteLine("Fornecedor " + dadosFruta.fornecedor);
}

```

Após a execução:

```

Test Name: RealizarUmCadastroDeUmaFruta
Test Outcome: Aprovado

Standard Output
-> Using app.config
Quando cadastrar uma fruta
--- table step argument ---
| Fruta | Quantidade | Fornecedor |
| Abacaxi | 8 | Atacadao |
-> done: ExemploSteps.QuandoCadastrarUmaFruta(<table>) (0,2s)
Então devo visualizar os dados preenchidos
Fruta: Abacaxi
Qtde 8
Fornecedor Atacadao
-> done: ExemploDoisSteps.EntaoDevoVisualizarOsDadosPreenchidos() (0,0s)

```

5.15 Specflow – Doc String

Doc strings são utilizados quando é necessário informar ou validar algum texto muito grande, e geralmente possuindo quebra de linha. São representados por três aspas duplas para abrir e fechar seu conteúdo. Dentro de da Feature de Exemplo inclua o seguinte cenário:

```

Cenario: Visualizar a mensagem de alerta
#Dado que accesei minha aplicação
    #E accesei a área de cadastro de animais
    #Quando tentar incluir um animal
        Então a mensagem deve ser visualizada
        """
        Animal incluído com sucesso !
        Não se esqueça de manter os registros atualizados.
        """

```

E gere seu passo dentro de **ExemploSteps**:

```

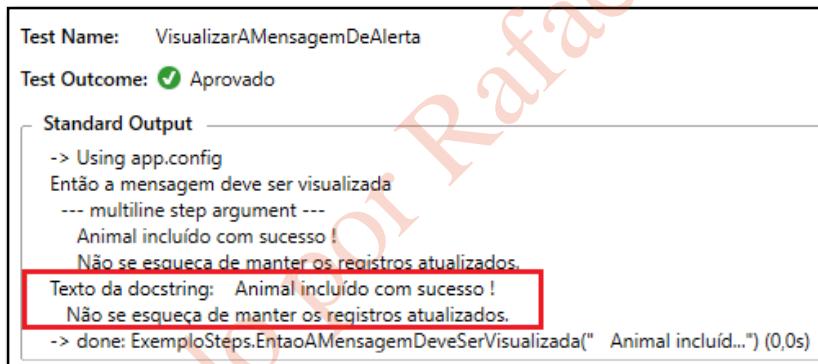
[Then(@"a mensagem deve ser visualizada")]
public void EntaoAMensagemDeveSerVisualizada(string multilineText)
{
    ScenarioContext.Current.Pending();
}

```

Basicamente recebe uma variável do tipo string como parâmetro e pode ser utilizada como vimos nos passos anteriores. Na implementação imprima este texto:

```
Console.WriteLine("Texto da docstring: " + multilineText);
```

Execute o teste:



5.16 Specflow - Hooks

Referência: <https://specflow.org/documentation/Hooks/>

O specflow possui a classe de Hooks que pode definir ações a serem executadas antes e depois dos testes. Possuem atributos equivalentes, por exemplo, ao TestInitalize/TestCleanup que vimos no outro projeto neste capítulo:

[Atributos de Inicialização e Finalização dos testes.](#)

- **[BeforeTestRun]**

Executado uma vez antes na inicialização antes da execução de todas as features.

O método deve ser estático.

Equivalente ao **[AssemblyInitialize]** do MsTest V1

- **[BeforeFeature]**

Executado uma vez antes da inicialização de cada Feature

O método deve ser estático.

Equivalente ao *[ClassInitialize]* do MsTest V1

- **[BeforeScenario]**

Executado antes de cada Cenário.

Equivalente ao *[TestInitialize]* do MsTest V1

- **[BeforeScenarioBlock]**

Executado antes de cada ‘bloco’ de cenário (ex: entre os ‘given’ e ‘when’)

- **[BeforeStep]**

Executado antes de cada passo do cenário

- **[AfterStep]**

Executado após cada passo do cenário

- **[AfterScenarioBlock]**

Executado após cada ‘bloco’ de cenário (ex: entre os ‘given’ e ‘when’)

- **[AfterScenario]**

Executado após cada cenário finalizado.

Equivalente ao *[TestCleanup]* do MsTest V1

- **[AfterFeature]**

Executado uma vez após a finalização de cada Feature

O método deve ser estático.

Equivalente ao *[ClassCleanup]* do MsTest V1

- **[AfterTestRun]**

Executado uma vez antes após a execução de todas as features.

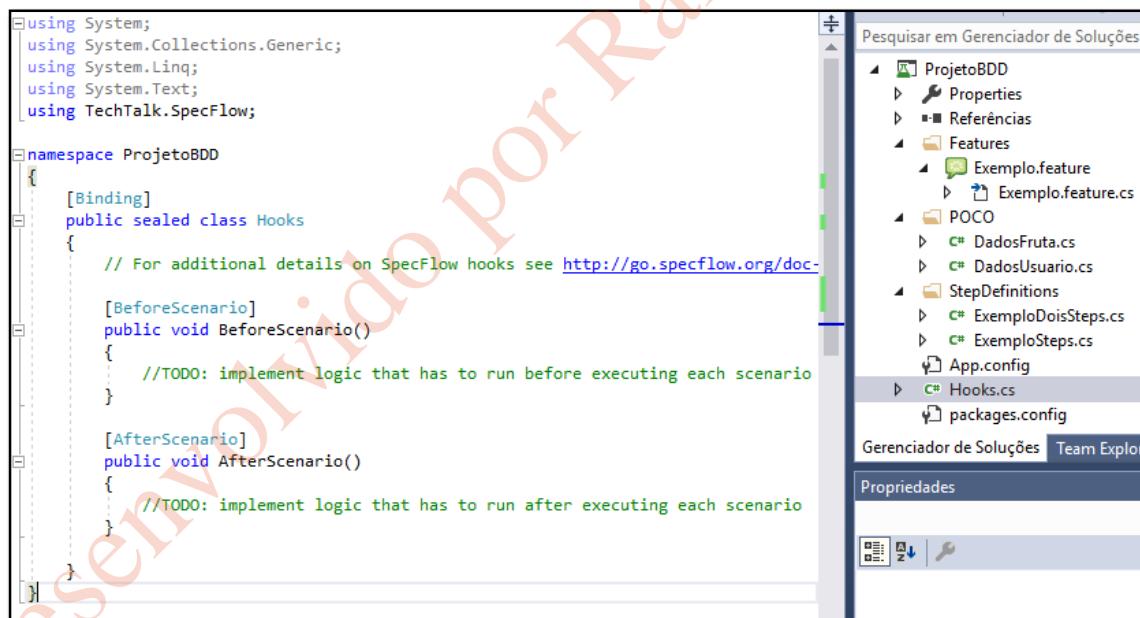
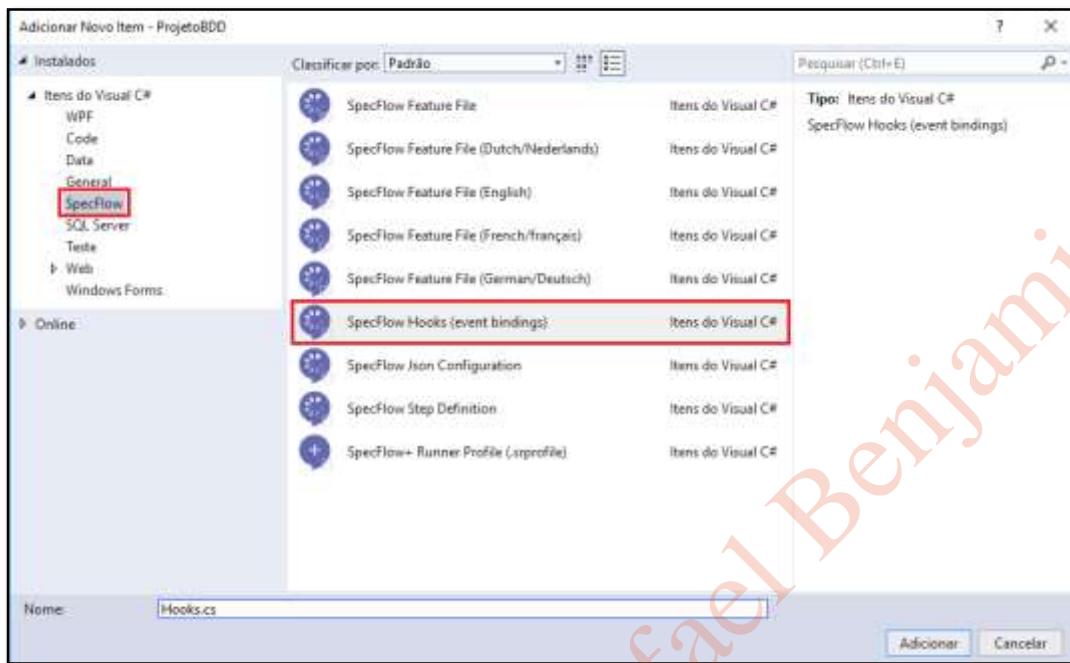
O método deve ser estático.

Obs: Esta chamada é executada pelo evento de descarregamento do assembly, o tempo de execução pode variar de acordo com cada tipo de executor de testes.

Equivalente ao *[AssemblyInitialize]* do MsTest V1

Para adicionar um arquivo de Hooks, clique com o botão direito no **Projeto > Adicionar > Novo Item**

Selecione no menu **Specflow**, o item **Specflow Hooks (event bindings)**



Dentro do BeforeScenario e AfterScenario, implemente para ser exibidas as mensagens:

```
[BeforeScenario]
public void BeforeScenario()
{
    Console.WriteLine("Executando Antes do Cenário");
}

[AfterScenario]
public void AfterScenario()
{
    Console.WriteLine("Executando Após do Cenário");
```

```
}
```

Execute os testes e veja o resultado:

```
Test Name: SubtrairUmNumeroESumarUm
Test Outcome: ✅ Aprovado
Standard Output
-> Using app.config
Executando Antes do Cenário
Dado que liguei a calculadora
Calculadora Ligada
-> done: ExemploSteps.GivenILHaveTurnedACalculatorOn() (0,0s)
Quando subtraio 25
-> done: ExemploDoisSteps.QuandoSubtraio(25) (0,0s)
E somo 10
-> done: ExemploSteps.WhenIAAdd(10) (0,0s)
Então Então o resultado na tela deve ser -15
Nome do cenário: Subtrair um número e somar um
Tipo de passo: Then
-> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(-15) (0,0s)
Executando Após do Cenário
```

Os comandos são executados antes e depois do cenário, semelhante ao que vimos no anterior. Para visualizar todas as opções altere o arquivo hooks para o seguinte:

```
[Binding]
public sealed class Hooks
{
    // For additional details on SpecFlow hooks see http://go.specflow.org/doc-hooks

    [BeforeTestRun]
    public static void BeforeTestRun()
    {
        Console.WriteLine("Executando Antes de todas as Features");
    }

    [AfterTestRun]
    public static void AfterTestRun()
    {
        Console.WriteLine("Executando depois de todas as Features");
    }

    [BeforeFeature]
    public static void BeforeFeature()
    {
        Console.WriteLine("Executando Antes da Feature");
    }

    [AfterFeature]
    public static void AfterFeature()
    {
        Console.WriteLine("Executando Depois da Feature");
    }

    [BeforeScenario]
    public void BeforeScenario()
    {
        Console.WriteLine("Executando Antes do Cenário");
    }
```

```

[AfterScenario]
public void AfterScenario()
{
    Console.WriteLine("Executando Após do Cenário");
}

[BeforeStep]
public void BeforeStep()
{
    Console.WriteLine("Executando Antes de cada Passo");
}

[AfterStep]
public void AfterStep()
{
    Console.WriteLine("Executando Após de cada Passo");
}

[BeforeScenarioBlock]
public void BeforeScenarioBlock()
{
    Console.WriteLine("Executando Antes do bloco de cenário");
}

[AfterScenarioBlock]
public void AfterScenarioBlock()
{
    Console.WriteLine("Executando Após o bloco de cenário");
}

```

-> Using app.config
 Executando Antes de todas as Features
 Executando Antes da Feature
 Executando Antes do Cenário
 Executando Antes do bloco de cenário
 Dado que liguei a calculadora
 Executando Antes de cada Passo
 Calculadora Ligada
 -> done: ExemploSteps.GivenIHaveTurnedACalculatorOn() (0,0s)
 Executando Após de cada Passo
 Executando Após o bloco de cenário
 Executando Antes do bloco de cenário
 Quando subtraio 25
 Executando Antes de cada Passo
 -> done: ExemploDoisSteps.QuandoSubtraio(25) (0,0s)
 Executando Após de cada Passo
 E somo 10
 Executando Antes de cada Passo
 -> done: ExemploSteps.WhenIAdd(10) (0,0s)
 Executando Após de cada Passo
 Executando Após o bloco de cenário
 Executando Antes do bloco de cenário
 Então Então o resultado na tela deve ser -15
 Executando Antes de cada Passo
 Nome do cenário: Subtrair um número e somar um
 Tipo de passo: Then
 -> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(-15) (0,0s)
 Executando Após de cada Passo
 Executando Após o bloco de cenário
 Executando Após do Cenário
 Executando Depois da Feature

Caso não queria ficar vendo estas saídas antes e depois de cada item na execução é só comentar o código com estas ações dentro do Hooks (Comentário geral com /* e */ ou linha a linha com //), irei usar o geral abaixo:

```
[Binding]
public sealed class Hooks
{
    // For additional details on SpecFlow hooks see http://go.specflow.org/doc-hooks
    /*
        [BeforeTestRun]
        public static void BeforeTestRun()
        {
            Console.WriteLine("Executando Antes de todas as Features");
        }

        [AfterTestRun]
        public static void AfterTestRun()
        {
            Console.WriteLine("Executando depois de todas as Features");
        }

        [BeforeFeature]
        public static void BeforeFeature()
        {
            Console.WriteLine("Executando Antes da Feature");
        }

        [AfterFeature]
        public static void AfterFeature()
        {
            Console.WriteLine("Executando Depois da Feature");
        }

        [BeforeScenario]
        public void BeforeScenario()
        {
            Console.WriteLine("Executando Antes do Cenário");
        }

        [AfterScenario]
        public void AfterScenario()
        {
            Console.WriteLine("Executando Após do Cenário");
        }

        [BeforeStep]
        public void BeforeStep()
        {
            Console.WriteLine("Executando Antes de cada Passo");
        }

        [AfterStep]
        public void AfterStep()
        {
            Console.WriteLine("Executando Após de cada Passo");
        }

        [BeforeScenarioBlock]
        public void BeforeScenarioBlock()
        {
            Console.WriteLine("Executando Antes do bloco de cenário");
        }
    */
}
```

```

[AfterScenarioBlock]
public void AfterScenarioBlock()
{
    Console.WriteLine("Executando Após o bloco de cenário");
}
*/
}

```

Podemos obter também informações relevantes sobre nossos testes que são úteis para exibir em relatórios como nome da feature, cenário, tag, ou passo em execução.

No arquivo hooks, abaixo da última função comentada inclua a seguinte função para ser executada após cada passo:

```

[AfterStep]
public void AfterScenario()
{
    string tituloFeature = FeatureContext.Current.FeatureInfo.Title;
    string tituloCenario = ScenarioContext.Current.ScenarioInfo.Title;

    //Necessário que possua uma tag, retorna um array com as tags
    var tagCenario = ScenarioContext.Current.ScenarioInfo.Tags;

    //A informação do passo só é obtida no before ou after Step
    string tipoPasso =
    ScenarioContext.Current.StepContext.StepInfo.StepInstance.Keyword.ToString();
    string descricaoPasso = ScenarioContext.Current.StepContext.StepInfo.Text;

    Console.WriteLine("Titulo da Feature: " + tituloFeature);
    Console.WriteLine("Titulo do Cenario: " + tituloCenario);
    Console.WriteLine("Tag do Cenario: " + tagCenario.First());
    Console.WriteLine("Passo atual: " + tipoPasso + descricaoPasso);
}

```

Faz uso do **ScenarioContext** que vimos anteriormente, mas agora também usa o **FeatureContext** que possui informações da Feature, e para obter informações do Passo usa o **ScenarioContext.Current.StepContext**

Execute algum cenário que possua alguma tag, ou inclua uma tag no cenário a ser executado.

```

Test Name: SomarDoisNumeros
Test Outcome: ✅ Aprovado

Standard Output
-> Using app.config
Dado que liguei a calculadora
Calculadora Ligada
-> done: ExemploSteps.GivenILHaveTurnedACalculatorOn() (0,0s)
Titulo da Feature: Exemplo
Titulo do Cenario: Somar dois números
Tag do Cenario: mytag
Passo atual: Given que liguei a calculadora
Quando somo 50
-> done: ExemploSteps.WhenIAdd(50) (0,0s)
Titulo da Feature: Exemplo
Titulo do Cenario: Somar dois números
Tag do Cenario: mytag
Passo atual: When somo 50
E somo 70
-> done: ExemploSteps.WhenIAdd(70) (0,0s)
Titulo da Feature: Exemplo
Titulo do Cenario: Somar dois números
Tag do Cenario: mytag
Passo atual: When somo 70
Então o resultado na tela deve ser 120
-> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(120) (0,0s)
Titulo da Feature: Exemplo
Titulo do Cenario: Somar dois números
Tag do Cenario: mytag
Passo atual: Then o resultado na tela deve ser 120

```

E novamente, para que estas informações não fiquem se repetindo nos passos seguintes, comente esta função.

5.17 Specflow – Scoped Bindings

Referência: <https://specflow.org/documentation/Scoped-Bindings/>

As bindings possuem um escopo global dentro de nosso projeto, mas também podemos restringir a execução a condições específicas.

Inclua mais duas features: **LoginAdministracao.Feature** e **LoginUsuario.Feature**

Na **Login Administração** atualize a feature para o seguinte:

```
#language: pt

Funcionalidade: Login Administração
Como um Administrador
Quero realizar um Login

@administracao
Cenario: Relizar o login na aplicação A
Dado que acessei minha aplicação
Quando realizar o login
Então devo visualizar a área logada
```

E no **Login Usuario** atualize a feature para o seguinte:

```
#language: pt

Funcionalidade: Login Usuário
Como um usuário
```

Quero realizar um Login

@usuario

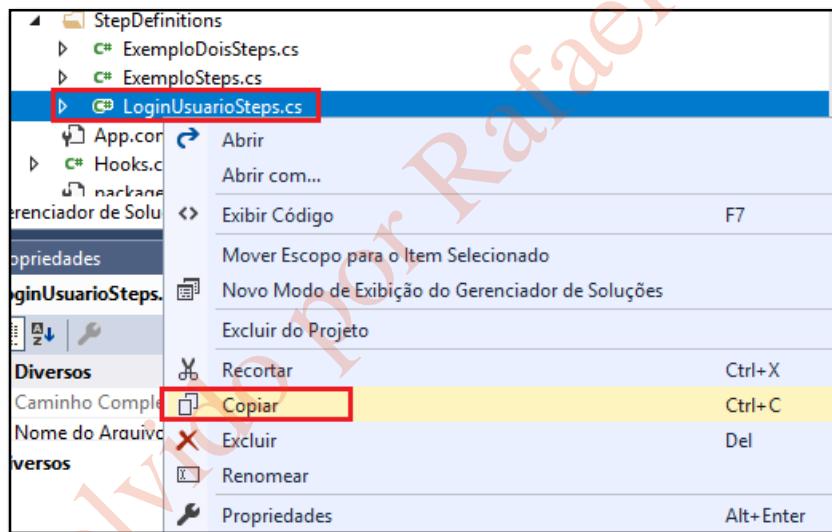
Cenario: Relizar o login na aplicação B
Dado que acessei minha aplicação
Quando realizar o login
Então devo visualizar a área logada

Imagine que estas duas feature foram descritas com as mesmas frases, porém a ação a ser realizada será em aplicações distintas. A solução mais comum é especificar diretamente na frase do passo o que deve ser feito. Ex: *Dado que acessei minha aplicação B, Quando realizar o login de usuário, Então devo visualizar a área logada da aplicação B.*

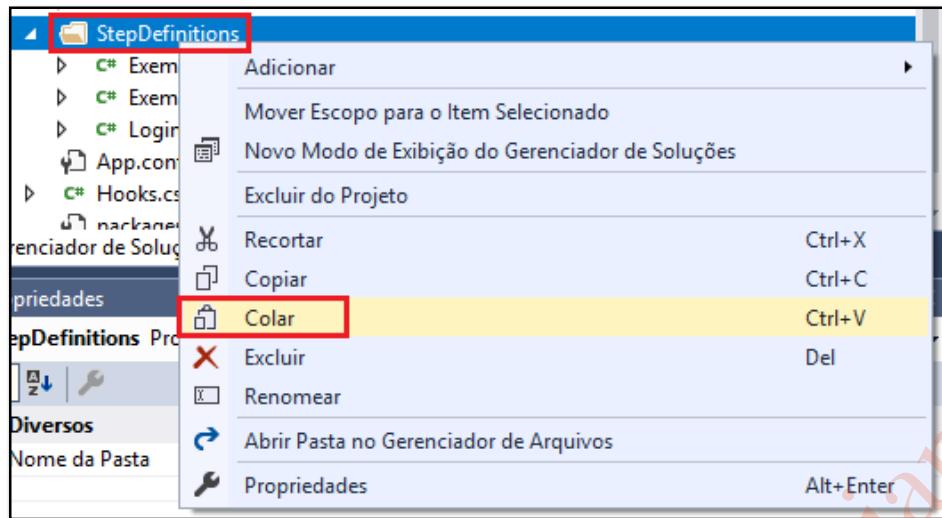
Mas, para fins didáticos vamos ver outra maneira (menos recomendada) de fazer isso. Como a feature é composta por frases pode ser que ocorra de algum dia duas frases iguais existam nas suas feature e precisem ser mantidas sem alteração.

Gere um step definition para o login usuário.

Em seguida duplique este step e altere para login usuário. Para isso clique com o botão direito nesse step criado e selecione copiar.



Clique o botão direito dentro da pasta StepDefinitions e clique Colar.



Renomeie este arquivo de cópia para **LoginAdministradorSteps**. E altere o nome da classe para o mesmo.

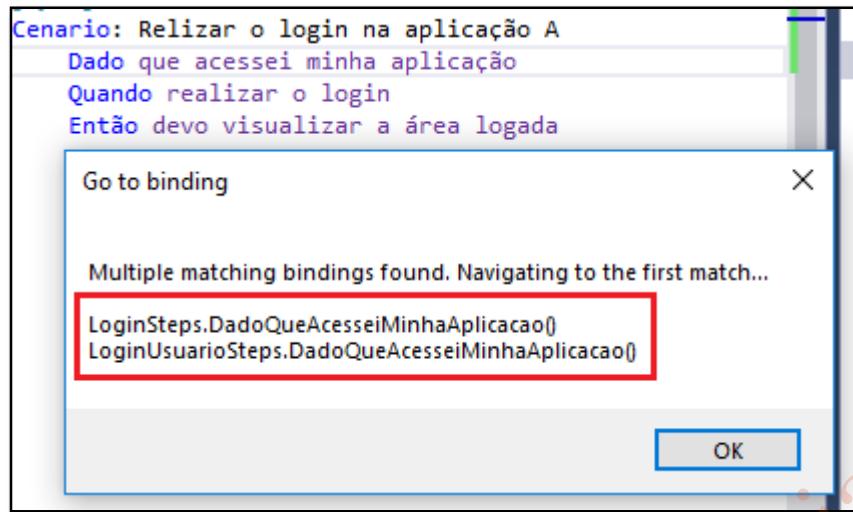
```

1  using System;
2  using TechTalk.SpecFlow;
3
4  namespace ProjetoBDD.StepDefinitions
5  {
6      [Binding]
7      public class LoginAdministradorSteps
8      {
9          [Given(@"que accesei minha aplicação")]
10         public void DadoQueAcesseiMinhaAplicacao()
11         {
12             ScenarioContext.Current.Pending();
13         }
14
15         [When(@"realizar o login")]
16         public void QuandoRealizarOLogin()
17         {
18             ScenarioContext.Current.Pending();
19         }
}

```

Agora devemos ter a implementação de ambos os passos repetidos em dois arquivos de step definition.

Volte nas features, e tente navegar para alguma definição destes novos steps. Será exibido o seguinte alerta:

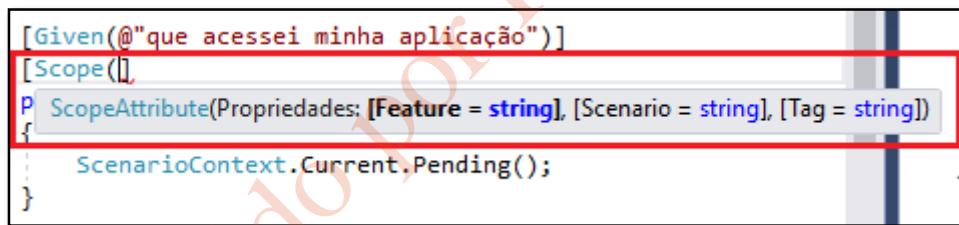


A implementação mesmo passo foi encontrada em dois arquivos diferentes.

Para isso podemos restringir a execução dos steps a:

- Tag
- Feature (usando o título da Feature)
- Cenário (usando o título do Cenário)

Onde podem ser utilizados um ou mais destes atributos em conjunto



Altere sua classe de **LoginUsuarioSteps** para o seguinte:

```
[Binding]
public class LoginUsuarioSteps
{
    [Given(@"que accesei minha aplicação")]
    [Scope()]
    public void DadoQueAcesseiMinhaAplicacao()
    {
        ScenarioContext.Current.Pending();
    }

    [When(@"realizar o login")]
    [Scope(Scenario = "Relizar o login na aplicação B")]
    public void QuandoRealizarOLogin()
    {
        ScenarioContext.Current.Pending();
    }

    [Then(@"devo visualizar a área logada")]
    [Scope(Tag = "usuario")]
    public void EntaoDevoVisualizarAAreaLogada()
    {
        ScenarioContext.Current.Pending();
    }
}
```

```

        Console.WriteLine("Área logada da aplicação B (Usuário)");
    }
}

```

Repare que acima de cada de cada método, inclui o atributo que define o escopo para qual Feature, Cenário ou Tag cada passo deve ser executado. Neste caso fiz o vínculo do primeiro passo com o título da feature:

```

[Given(@"que accesei minha aplicação")]
[Scope(Feature = "Login Usuário")]
public void DadoQueAcesseiMinhaAplicacao()

```

Do segundo passo com o nome do cenário:

```

[When(@"realizar o login")]
[Scope(Scenario = "Relizar o login na aplicação B")]
public void QuandoRealizarOLogin()

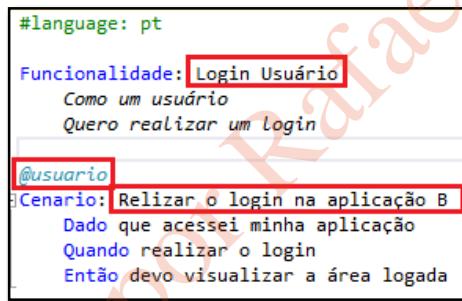
```

E do terceiro passo com a tag do cenário (sem o @):

```

[Then(@"devo visualizar a área logada")]
[Scope(Tag = "usuario")]
public void EntaoDevoVisualizarAAreaLogada()

```



Agora volte a feature de **LoginUsuario** e note que os steps estão em preto, significando que o vinculo com os passos foi criado. Se tentar navegar até sua definição irá ser direcionado para estes passos.

E na outra feature **LoginAdministracao**, repare que os steps também estão sendo direcionado corretamente, já que por eliminação o specflow entende que estes passos serão vinculados aos passos que sobraram, já que os outros serão executados apenas para o escopo definido. Mas neste caso seria bom definir que os passos estão vinculados à uma determinada feature, então atualize sua classe de **LoginAdministradorteps** para o seguinte:

```

[Binding]
public class LoginAdministradorSteps
{
    [Given(@"que accesei minha aplicação")]
    [Scope(Tag = "administracao", Feature = "Login Administração")]
    public void DadoQueAcesseiMinhaAplicacao()
    {
        Console.WriteLine("Acesso a aplicação de Administracão");
    }

    [When(@"realizar o login")]
    [Scope(Feature = "Login Administração")]
    [Scope(Tag = "administracao")]
    public void QuandoRealizarOLogin()
    {
        Console.WriteLine("Login aplicação A");
    }
}

```

```

}

[Then(@"devo visualizar a área logada")]
[Scope(Feature = "Login Administração")]
[Scope(Tag = "administracao")]
[Scope(Scenario = "Relizar o login na aplicação A")]
public void EntaoDevoVisualizarAAreaLogada()
{
    Console.WriteLine("Area logada da aplicacao A (Adm)");
}
}

```

Repare que agora utilizei mais de uma condição para definir o escopo, que também pode ser utilizados em conjunto. Mas todas as condições devem ser existentes para que o escopo seja definido.

Execute ambas as features novamente e veja na saída que os resultados executam os passo corretamente.

<p>Test Name: RelizarOLoginNaAplicacaoB</p> <p>Test Outcome: ✓ Aprovado</p> <p>Standard Output</p> <pre> -> Using app.config Dado que accesei minha aplicação Acesso a aplicacao de Usuario -> done: LoginUsuarioSteps.DadoQueAcesseiMinhaAplicacao() (0,0s) Quando realizar o login Login aplicacao B -> done: LoginUsuarioSteps.QuandoRealizarOLogin() (0,0s) Então devo visualizar a área logada Area logada da aplicacao B (Usuário) -> done: LoginUsuarioSteps.EntaoDevoVisualizarAAreaLogada() (0,0s) </pre>	<p>Test Name: RelizarOLoginNaAplicacaoA</p> <p>Test Outcome: ✓ Aprovado</p> <p>Standard Output</p> <pre> -> Using app.config Dado que accesei minha aplicação Acesso a aplicacao de Administracao -> done: LoginAdministradorSteps.DadoQueAcesseiMinhaAplicacao() (0,0s) Quando realizar o login Login aplicacao A -> done: LoginAdministradorSteps.QuandoRealizarOLogin() (0,0s) Então devo visualizar a área logada Area logada da aplicacao A (Adm) -> done: LoginAdministradorSteps.EntaoDevoVisualizarAAreaLogada() (0,0s) </pre>
---	---

Ainda é possível limitar este escopo dentro dos hooks com o uso de tags. Para isso inclua no cenário da feature **Login Administração** a tag **@web**:

```

@administracao @web
Cenario: Relizar o login na aplicação A
  Dado que accesei minha aplicação
  Quando realizar o login
  Então devo visualizar a área logada

```

E no cenário da feature **Login Usuário** a tag **@servico**:

```

@usuario @servico
Cenario: Relizar o login na aplicação B
  Dado que accesei minha aplicação
  Quando realizar o login
  Então devo visualizar a área logada

```

Antes disso, verifique se seu arquivo Hooks que criamos anteriormente teve todas as suas funções comentadas.

Após a ultima função comentada dentro da classe inclua as seguintes funções:

```

[BeforeScenario("web")]
public static void BeforeWebScenario()
{
    Console.WriteLine("Executando Antes do Cenário com tag WEB");
}

[BeforeScenario("servico")]

```

```

public static void BeforeServiceScenario()
{
    Console.WriteLine("Executando Antes do Cenário com tag SERVICO");
}

```

O que foi feito agora é que temos também dois BeforeScenario, estas duas funções sempre irão ser executadas antes de cada cenário. Porém agora uma irá executar apenas para cenários com a tag “web” e a outra apenas para cenário com a tag “serviço”. Execute as features LoginUsuario e LoginAdminstracao. Também execute alguma feature anterior que fizemos e não possuem estas tags e veja o resultado:

Test Name: RealizarLoginNaAplicacaoA	Test Name: RealizarLoginNaAplicacaoB	Test Name: SomarDoisNumeros
Test Outcome: Aprovado	Test Outcome: Aprovado	Test Outcome: Aprovado
<p>Standard Output</p> <pre> -> Using app.config Executando Antes do Cenário com tag WEB Dado que acessei minha aplicação Acesso a aplicação de Administrador -> done: LoginAdministratorSteps.DadoQueAcesseiMinhaAplicacao() (0.0s) Quando realizar o login Login aplicação A -> done: LoginAdministratorSteps.QuandoRealizarOLogin() (0.0s) Então devo visualizar a área logada Área logada da aplicação A (Adm) -> done: LoginAdministratorSteps.EntaoDevoVisualizarAAreaLogada() (0.0s) </pre>	<p>Standard Output</p> <pre> -> Using app.config Executando Antes do Cenário com tag SERVICO Dado que acessei minha aplicação Acesso a aplicação de Usuário -> done: LoginUserSteps.DadoQueAcesseiMinhaAplicacao() (0.0s) Quando realizar o login Login aplicação B -> done: LoginUserSteps.QuandoRealizarOLogin() (0.0s) Então devo visualizar a área logada Área logada da aplicação B (Usuário) -> done: LoginUserSteps.EntaoDevoVisualizarAAreaLogada() (0.0s) </pre>	<p>Standard Output</p> <pre> -> Using app.config Dado que liguei a calculadora Calculadora Ligada -> done: ExemploSteps.GivenHaveTurnedACalculatorOn() (0.0s) Quando some 50 -> done: ExemploSteps.WhenAdd(50) (0.0s) É soma 70 -> done: ExemploSteps.ThenAdd(70) (0.0s) Então Então o resultado na tela deve ser 120 Nome do cenário: Somar dois números Tipo de passo: Then -> done: ExemploSteps.ThenTheResultShouldBeOnTheScreen(120) (0.0s) </pre>
Sem nenhuma das tags		

5.18 Specflow – Contexto

O contexto é utilizado para agrupar os passos em comum, geralmente utilizados para passos tipo ‘Dado’ que representam a pré condição dos testes. Não é recomendado utilizar para passos mais complexos, e não é uma regra sempre tentar ‘enxutar’ os passos e realizar seu uso. Só use quando notar que os mesmos passos estão repetidos em todos os cenários.

Atualize o LoginUsuário para o seguinte:

```

#language: pt

Funcionalidade: Login Usuário
  Como um usuário
  Quero realizar um login

@usuario @servico
Cenario: Relizar o login na aplicação B
  Dado que acessei minha aplicação
  Quando realizar o login
  Então devo visualizar a área logada

@usuario @servico
Cenario: Relizar o login com usuario invalido na aplicação B
  Dado que acessei minha aplicação
  #Quando realizar o login com um usuario invalido
  #Então devo visualizar a mensagem de erro

@usuario @servico
Cenario: Relizar o login com usuario inativo na aplicação B
  Dado que acessei minha aplicação
  #Quando realizar o login com um usuario inativo
  #Então devo visualizar a mensagem de erro

@usuario @servico
Cenario: Relizar o login com usuario cancelado na aplicação B
  Dado que acessei minha aplicação
  #Quando realizar o login com um usuario cancelado

```

```
#Então devo visualizar a mensagem de erro
```

Não vamos implementar os passos novos, por isso pode deixar comentado. O importante agora é reparar que todos os passos do tipo ‘Dado’ estão se repetindo, neste caso podemos utilizar o contexto e deixar a feature da seguinte forma:

```
#language: pt

Funcionalidade: Login Usuário
  Como um usuário
  Quero realizar um Login

Contexto: Dado que acessei minha aplicação

@usuario @servico
Cenario: Relizar o login na aplicação B
  Quando realizar o login
  Então devo visualizar a área logada

@usuario @servico
Cenario: Relizar o login com usuario invalido na aplicação B
  #Quando realizar o login com um usuario invalido
  #Então devo visualizar a mensagem de erro

@usuario @servico
Cenario: Relizar o login com usuario inativo na aplicação B
  #Quando realizar o login com um usuario inativo
  #Então devo visualizar a mensagem de erro

@usuario @servico
Cenario: Relizar o login com usuario cancelado na aplicação B
  #Quando realizar o login com um usuario cancelado
  #Então devo visualizar a mensagem de erro
```

Agora o que estiver no contexto será executado antes de cada de cada cenário nesta feature. Não é necessário alterar nada dentro do step definition, é só uma maneira de rezudir a escrita. Lembrando, neste caso foi visto que todos os passos possuíam algo em comum.

Se caso um dos cenários apresentasse um ‘Dado’ diferente, não poderíamos utilizar desta forma já que o contexto sempre executa para todos o que estiver declarado nele. E não tente mudar este cenário diferente para se encaixar nos demais apenas para usar o Contexto, ele só deve ser usado caso você veja que há esta possibilidade sem impactar nos cenários já escritos, não é uma obrigação usa-lo.

5.19 Atualizando para Specflow 3

Não compatível com MsTest V1.

<https://specflow.org/2019/updating-to-specflow-3/>

5.20 Refazendo o projeto com BDD

Agora que já sabemos como criar um projeto de automação e vimos os conceitos sobre BDD, vamos atualizar nosso primeiro projeto para formato BDD.

Crie um novo projeto. Vou chamar o meu de **ProjetoAutomacaoBDD**.

Como também usaremos o MsTest V1, siga novamente os passos descritos na seção [Alterando o framework de testes para MsTest Versão 1](#) para configurar o framework de testes removendo o MsTestV2 deste novo projeto.

Depois de configurar, apague a classe **UnitTest1.cs** que vem por padrão no projeto, não iremos criar testes neste formato.

Baixe também os pacotes que utilizamos anteriormente:

- Bogus
- DotNetSeleniumExtras.PageObjects
- DotNetSeleniumExtras.WaitHelpers
- Selenium.Chrome.WebDriver
- Selenium.InternetExplorer.WebDriver
- Selenium.Firefox.WebDriver
- Selenium.Support
- Selenium.WebDriver
- Specflow (**v2.4.1**)
- SpecFlow.Assist.Dynamic (v1.3.1)

Configure o specflow como visto nesta seção: [Specflow – Configuração](#).

Não é necessário instalar o plugin do visual Studio novamente, uma vez instalado ele permanece no visual Studio.

Lembrando que como iremos usar a versão 2.4.1 do specflow, se certifique de que a opção **Enable SpecFlowSingleFileGenerator**, está como **True**.

5.21 Login Feature

Vamos começar pelo login, da maneira simples, inclua um novo arquivo feature chamado **Login.feature**

Botão direito no projeto > Novo Item > Specflow > Specflow Feature File

Antes de copiar o exemplo abaixo, tente escrever da sua forma como seriam os cenários para os testes de Login válido e Login inválido, lembrando que sempre iremos começar com a especificação e depois a implementação.

```
#language: pt

Funcionalidade: Login
  Como um usuário
  Quero realizar o login no site
  Para ter acesso à minha área Logada

Cenário: Login com usuário válido
  Dado que acessei o site de compras
  Quando realizar um login utilizando um usuário válido
  Então devo visualizar a área logada do site

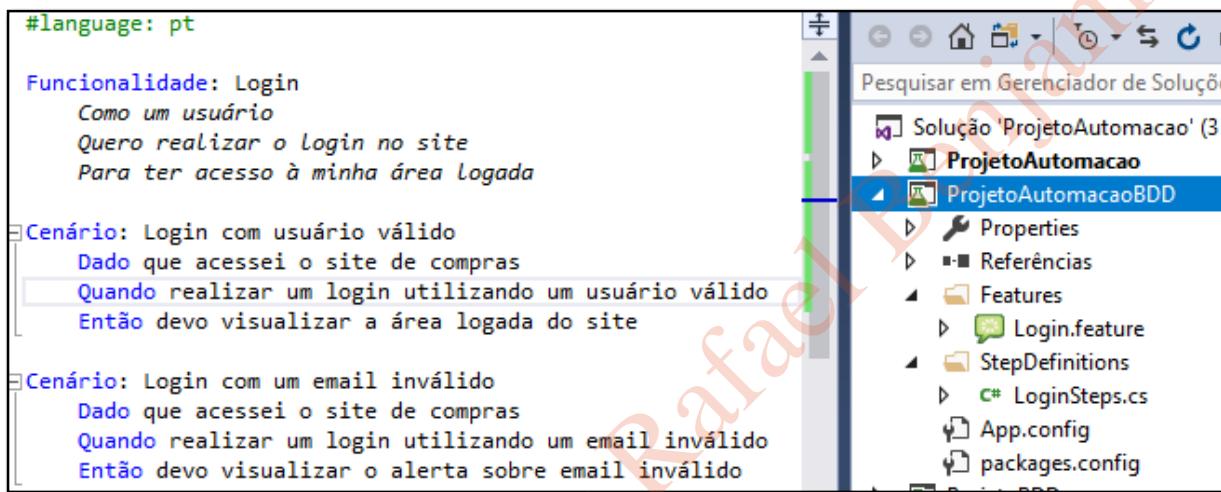
Cenário: Login com um email inválido
  Dado que acessei o site de compras
  Quando realizar um login utilizando um email inválido
  Então devo visualizar o alerta sobre email inválido
```

Muito provavelmente sua maneira de escrever os cenários ficou diferente da minha, este é o primeiro desafio de quando usamos o BDD, a idéia agora é se juntar com os demais interessados e ver qual é a melhor forma de escrever os cenários afim que de todos estejam com o mesmo entendimento sobre o que cada cenário especifica.

Por exemplo, no meu cenário não me refiro à navegar para a página de login após acessar o site, apenas assumo que está implícito este comportamento dentro do passo realizar um login.

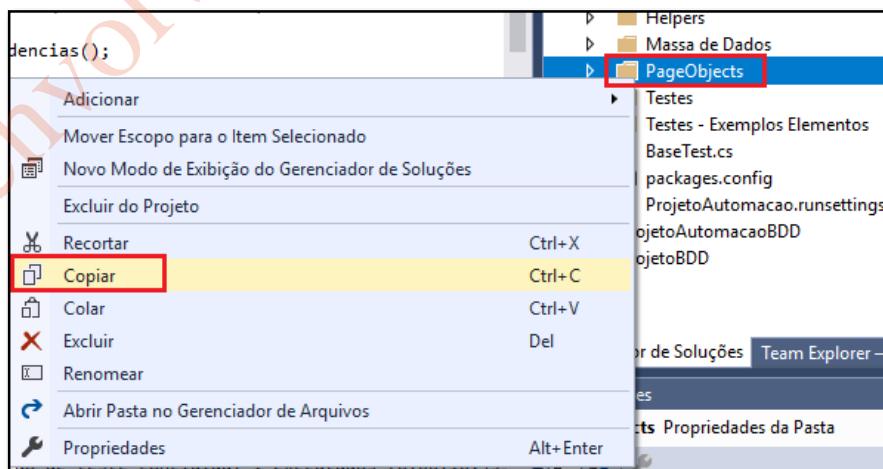
Apenas para dar continuidade irei prosseguir com a implementação do cenário sem alterá-la.

Crie uma pasta de **Features** e outra para **StepDefinitions** em seu projeto. Mova a feature de Login para a pasta Features, e crie a step definition para o login a partir dos steps.



Neste ponto, caso ainda não tivéssemos alguma versão do sistema pronta, poderíamos criar suas classes de Page objects e definir alguns métodos de ação. Após ser liberada uma versão iríamos mapear os objetos dentro de suas respectivas classes de Page objects e fazer ajustes aos métodos. Mas como já temos estas classes prontas vamos reaproveitá-las do primeiro projeto.

Copie a pasta **PageObjects** do outro projeto:



Cole no atual:



Agora temos que atualizar os namespaces destas classes para o projeto atual.

```

1  using OpenQA.Selenium;
2  using SeleniumExtras.PageObjects;
3
4  namespace ProjetoAutomacaoBDD.PageObjects
5  {
6
7      public class BasePage
8      {
9          protected IWebDriver driver;
10
11         public BasePage(IWebDriver driver)
12         {
13             this.driver = driver;
14             PageFactory.InitElements(driver, this);
15         }
16
17     }
18 }

```

No meu caso atualizei de **ProjetoAutomacao.PageObjects** para **ProjetoAutomacaoBDD.PageObjects**

Antes de começar a chamar as pages, precisamos de um driver para abrir nosso navegador e poder passar para as pages.

```

[Binding]
public class LoginSteps
{
    private IWebDriver driver;

    [Given(@"que accesei o site de compras")]
    public void DadoQueAcesseiOSiteDeCompras()
    {
        driver = new ChromeDriver();
        driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    }

    [When(@"realizar um login utilizando um usuário válido")]
    public void QuandoRealizarUmLoginUtilizandoUmUsuarioValido()
    {
        ScenarioContext.Current.Pending();
    }

    [When(@"realizar um login utilizando um email inválido")]
    public void QuandoRealizarUmLoginUtilizandoUmEmailInvalido()
    {
        ScenarioContext.Current.Pending();
    }

    [Then(@"devo visualizar a área logada do site")]
}

```

```

public void EntaoDevoVisualizarAAreaLogadaDoSite()
{
    ScenarioContext.Current.Pending();
}

[Then(@"devo visualizar o alerta sobre email inválido")]
public void EntaoDevoVisualizarOAlertaSobreEmailInvalido()
{
    ScenarioContext.Current.Pending();
}
}

```

Tente executar algum dos testes de login, o navegador deve abrir e acessar o site. Com isso temos o primeiro passo feito.

Agora vamos implementar o passo para o login válido usando os métodos que criamos para a Page, irei utilizar a forma funcional, caso queria user a estrutural.

```

[When(@"realizar um login utilizando um usuário válido")]
public void QuandoRealizarUmLoginUtilizandoUmUsuarioValido()
{
    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar("teste@testing.com", "1234qwer");
}

```

Porém note que email e senha deveriam ser parametrizados, desta forma estão como uma prática ruim, sendo hard coded no passo. Para melhorar isso podemos fazer o uso de tabelas, alterando o cenário para o seguinte:

Cenário: Login com usuário válido
Dado que accesei o site de compras
Quando realizar um login utilizando um usuário válido

Usuario	Senha
teste@testing.com	1234qwer

Então devo visualizar a área logada do site

E no passo incluindo o parâmetro Table e realizando a leitura usando o Assist.Dynamic:

```

[When(@"realizar um login utilizando um usuário válido")]
public void QuandoRealizarUmLoginUtilizandoUmUsuarioValido(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

```

Execute o teste de login válido novamente e veja que agora o login é realizado, mas ainda falta a validação, que pode ser feita desta forma:

```

[Then(@"devo visualizar a área logada do site")]
public void EntaoDevoVisualizarAAreaLogadaDoSite()
{
    HomeLogadaPage home = new HomeLogadaPage(driver);
    Assert.IsNotNull(home.VerificaHomeLogada());
}

```

```

        driver.Quit();
    }
}

```

Agora ao executar o teste deve passar.

A implementação para o login inválido é seguindo a mesma idéia, porém ao final posso especificar a frase a ser validada na feature, ficando desta forma por exemplo:

```

Cenário: Login com um email inválido
  Dado que acessei o site de compras
  Quando realizar um login utilizando um email inválido
    | Usuario | Senha |
    | teste@a.com | 1234qwer |
  Então devo visualizar o alerta sobre email inválido
  """
  There is 1 error Authentication failed.
  """

```

E a implementação dos passos:

```

[When(@"realizar um login utilizando um email inválido")]
public void QuandoRealizarUmLoginUtilizandoUmEmailInvalido(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre email inválido")]
public void EntaoDevoVisualizarOAlertaSobreEmailInvalido(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
    driver.Quit();
}

```

No passo em que fazemos a leitura dos dados das tabelas para login válido e inválido, repare que foi utilizada exatamente a mesma lógica, neste caso seria mais simples escrever um passo com as mesmas ações e simplificar nossa implementação... bem, talvez não.

Imagine que simplificássemos os dois steps para algo como “**Quando realizar um login**”, os cenários ficariam assim:

```

Cenário: Login com usuário válido
  Dado que acessei o site de compras
  Quando realizar um login
    | Usuario | Senha |
    | teste@testing.com | 1234qwer |
  Então devo visualizar a área logada do site

Cenário: Login com um email inválido

```

```

Dado que acessei o site de compras
Quando realizar um login
| Usuario | Senha |
| teste@a.com | 1234qwer |
Então devo visualizar o alerta sobre email inválido
"""
There is 1 error Authentication failed.
"""

```

Execute o teste, deve passar também.

Pronto, agora podemos resumir em um único step a mesma ação... porém a idéia principal que é ter uma especificação que possa ser lida perdeu valor. Antes tínhamos uma especificação onde estava claro qual tipo de dado deveria estar ali. Mesmo tendo o dado sendo apresentado pela tabela abaixo, não fica claro que tipo de dado deve estar ali, portanto sempre entre em acordo com seu time antes de tomar este tipo de decisão. Este é um exemplo simples de login e talvez o próprio nome do cenário já seja o suficiente, mas em cenários maiores onde se usam mais dados talvez isso não ocorra com a mesma facilidade.

Inclua mais três cenários negativos:

```

Cenário: Login sem preencher o email
Dado que acessei o site de compras
Quando realizar um login utilizando um email vazio
| Usuario | Senha |
|          | 1234qwer |
Então devo visualizar o alerta sobre email vazio
"""
There is 1 error An email address required.
"""

Cenário: Login com uma senha inválida
Dado que acessei o site de compras
Quando realizar um login utilizando uma senha inválida
| Usuario | Senha |
| teste@a.com | 1 |
Então devo visualizar o alerta sobre uma senha inválida
"""
There is 1 error Invalid password.
"""

Cenário: Login sem preencher a senha
Dado que acessei o site de compras
Quando realizar um login utilizando uma senha vazia
| Usuario | Senha |
| teste@a.com | |
Então devo visualizar o alerta sobre uma senha vazia
"""
There is 1 error Password is required.
"""

```

E suas implementações:

```

[When(@"realizar um login utilizando um email vazio")]
public void QuandoRealizarUmLoginUtilizandoUmEmailVazio(Table table)
{
    dynamic user = table.CreateDynamicInstance();
    string usuario = user.Usuario;
}

```

```

        string senha = user.Senha;

        HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
        home.AcessarLogin()
            .Logar(usuario, senha);
    }

[Then(@"devo visualizar o alerta sobre email vazio")]
public void EntaoDevoVisualizarOAlertaSobreEmailVazio(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
    driver.Quit();
}

[When(@"realizar um login utilizando uma senha inválida")]
public void QuandoRealizarUmLoginUtilizandoUmaSenhaInvalida(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre uma senha inválida")]
public void EntaoDevoVisualizarOAlertaSobreUmaSenhaInvalida(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
    driver.Quit();
}

[When(@"realizar um login utilizando uma senha vazia")]
public void QuandoRealizarUmLoginUtilizandoUmaSenhaVazia(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre uma senha vazia")]
public void EntaoDevoVisualizarOAlertaSobreUmaSenhaVazia(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
    driver.Quit();
}

```

Ainda existe uma opção para agrupar os cenários negativos. Podemos utilizar o esquema do cenário como no exemplo abaixo:

#Cenarios negativos com esquema do cenário

Esquema do Cenário: Login cenários negativos

Dado que acessei o site de compras

Quando realizar um login com usuário <usuario> e senha <senha>

Então devo visualizar o alerta com a mensagem <mensagem>

Exemplos:

Tipo	usuario	senha	mensagem
Email Invalido	"teste@a.com"	"12312313"	"There is 1 error Authentication failed."
Email Vazio	"	"124qewr"	"There is 1 error An email address required."
Senha Invalida	"teste@testing.com"	"1"	"There is 1 error Invalid password."
Senha Vazia	"teste@testing.com"	" "	"There is 1 error Password is required."

Note que nos gerenciador de testes, o nome é exibido com o nome do cenário seguido do texto exibido na primeira coluna 'Tipo'. Por mais que a primeira coluna não seja utilizada, ela pode ser útil para referenciar o cenário e auxiliar na sua identificação.

- ! LoginCenariosNegativos_EmailInvalido
- ! LoginCenariosNegativos_EmailVazio
- ! LoginCenariosNegativos_SenhaInvalida
- ! LoginCenariosNegativos_SenhaVazia

E sua implementação:

```
[When(@"realizar um login com usuário ""(.*)"" e senha """(.*)""")]
public void QuandoRealizarUmLoginComUsuarioESenha(string usuario, string senha)
{
    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta com a mensagem """(.*)""")]
public void EntaoDevoVisualizarOAlertaComAMensagem(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
    driver.Quit();
}
```

5.22 Centralizando o driver no Hooks

Temos a primeira feature criada, mas repare que o driver existe apenas dentro da classe de step definition que criamos. Vamos manter o driver em apenas uma classe semelhante a como fizemos no primeiro projeto com a classe BaseTest. Crie uma nova classe hooks:

Clique com o botão direito no projeto > Adicionar > Novo Item > Specflow > Specflow Hooks

Seguindo a linha do primeiro projeto, vamos fazer com que o driver seja aberto e fechado após cada teste, neste caso após cada cenário.

Aproveitando o que já foi feito, copie o método CriaBrowser que fizemos dentro do BaseTest abaixo do AfterScenario.

Ajuste os imports necessários e crie a variável driver para a classe.

Dentro do BeforeScenario, inclua a chamada a para a função CriaBrowser e dentro do AfterScenario inclua o driver.Quit().

Até este ponto o Hooks deve estar desta forma:

```
[Binding]
public sealed class Hooks
{
    IWebDriver driver;

    [BeforeScenario]
    public void BeforeScenario()
    {
        driver = CriaDriver("CHROME");
        driver.Manage().Window.Maximize();
    }

    [AfterScenario]
    public void AfterScenario()
    {
        driver.Quit();
        driver.Dispose();
    }

    public IWebDriver CriaDriver(string browser)
    {
        switch (browser.ToUpper())
        {
            case "FIREFOX":
                driver = new FirefoxDriver();
                return driver;
            case "CHROMEHEADLESS":
                var options = new ChromeOptions();
                options.AddArgument("headless");
                driver = new ChromeDriver(options);
                return driver;
            case "CHROME":
                driver = new ChromeDriver();
                return driver;
            case "IE":
                driver = new InternetExplorerDriver();
                return driver;
            default:
                throw new AssertFailedException("Driver informado: " + browser + " Informe um
driver valido");
        }
    }
}
```

Como o specflow não consegue trabalhar da mesma forma com herança, iremos compartilhar o driver usando o Context Injection.

Iremos compartilhar a instância do driver criado aqui, mas agora não iremos criar uma classe POCO para driver.

Referência: <http://www.marcusoft.net/2013/04/ContextInjectionSpecFlow.html>

Abaixo da instância do driver, crie esta variável:

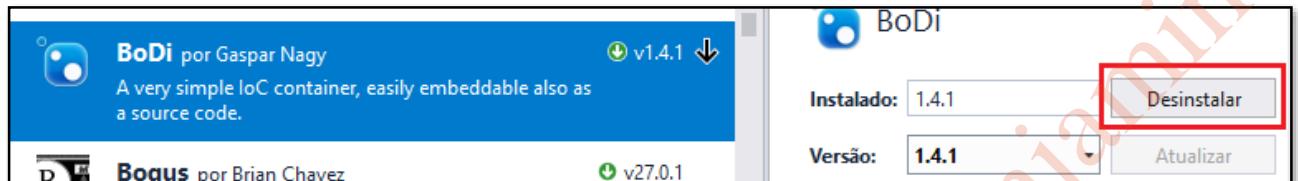
```
private readonly IObjectContainer objectContainer;
```

Ela irá precisar do import `using BoDi;`

Caso o erro abaixo seja exibido:



Desinstale o pacote 'BoDi'.



Vamos utilizar um novo comando, útil para compartilhar objetos com tipos já existentes como driver. Neste comando temos que informar o tipo da variável que será compartilhada e a variável como seu parâmetro:

```
objectContainer.RegisterInstanceAs<Tipo>(variavel);
```

Dentro do BeforeScenario, logo após a criação do driver inclua a linha:

```
objectContainer.RegisterInstanceAs<IWebDriver>(driver);
```

Esta linha é que enviara o nosso driver criado para a variável do tipo `objectContainer`, o qual por sua vez será compartilhado pelo construtor para as demais classes de stepdefinition.

E finalmente vamos incluir o construtor do hooks para compartilhar este objeto:

```
public Hooks(IObjectContainer objectContainer)
{
    this.objectContainer = objectContainer;
}
```

A classe Hooks ficou desta forma:

```
[Binding]
public sealed class Hooks
{
    private IWebDriver driver;
    private readonly IObjectContainer objectContainer;

    public Hooks(IObjectContainer objectContainer)
    {
        this.objectContainer = objectContainer;
    }

    [BeforeScenario]
    public void BeforeScenario()
    {
        driver = CriaDriver("CHROME");
        driver.Manage().Window.Maximize();
        objectContainer.RegisterInstanceAs<IWebDriver>(driver);
    }

    [AfterScenario]
```

```

public void AfterScenario()
{
    driver.Quit();
    driver.Dispose();
}

public IWebDriver CriaDriver(string browser)
{
    switch (browser.ToUpper())
    {
        case "FIREFOX":
            driver = new FirefoxDriver();
            return driver;
        case "CHROMEHEADLESS":
            var options = new ChromeOptions();
            options.AddArgument("headless");
            driver = new ChromeDriver(options);
            return driver;
        case "CHROME":
            driver = new ChromeDriver();
            return driver;
        case "IE":
            driver = new InternetExplorerDriver();
            return driver;
        default:
            throw new AssertFailedException("Driver informado: " + browser + " Informe um
driver valido");
    }
}
}

```

Agora para usar este driver, vamos criar o construtor na classe de step definition do login, mas agora no lugar da variável do tipo `ObjectContainer`, este construtor deve receber o tipo que foi definido quando usamos o comando `objectContainer.RegisterInstanceAs<Tipo>(variavel)`. Neste caso o tipo é `IWebDriver`.

Após receber este tipo, iremos armazena-lo em uma variável com visibilidade para a classe e usa-la normalmente.

```

private IWebDriver driver;

public LoginSteps(IWebDriver driver)
{
    this.driver = driver;
}

```

Agora podemos remover as chamadas de criação (`driver = new Chromedriver`) e encerramento (`driver.quit`) de driver, deixando a classe `LoginSteps` desta forma:

```

[Binding]
public class LoginSteps
{
    private IWebDriver driver;

    public LoginSteps(IWebDriver driver)
    {
        this.driver = driver;
    }

    [Given(@"que accesei o site de compras")]
    public void DadoQueAcesseiOSiteDeCompras()
    {

```

```

        driver.Navigate().GoToUrl("http://automationpractice.com/index.php");
    }

[When(@"realizar um login utilizando um usuário válido")]
public void QuandoRealizarUmLoginUtilizandoUmUsuarioValido(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[When(@"realizar um login utilizando um email inválido")]
public void QuandoRealizarUmLoginUtilizandoUmEmailInvalido(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar a área logada do site")]
public void EntaoDevoVisualizarAAreaLogadaDoSite()
{
    HomeLogadaPage home = new HomeLogadaPage(driver);
    Assert.IsNotNull(home.VerificaHomeLogada());
}

[Then(@"devo visualizar o alerta sobre email inválido")]
public void EntaoDevoVisualizarOAlertaSobreEmailInvalido(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
}

[When(@"realizar um login utilizando um email vazio")]
public void QuandoRealizarUmLoginUtilizandoUmEmailVazio(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre email vazio")]
public void EntaoDevoVisualizarOAlertaSobreEmailVazio(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
}

```

```

[When(@"realizar um login utilizando uma senha inválida")]
public void QuandoRealizarUmLoginUtilizandoUmaSenhaInvalida(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre uma senha inválida")]
public void EntaoDevoVisualizarOAlertaSobreUmaSenhaInvalida(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
}

[When(@"realizar um login utilizando uma senha vazia")]
public void QuandoRealizarUmLoginUtilizandoUmaSenhaVazia(Table table)
{
    dynamic user = table.CreateDynamicInstance();

    string usuario = user.Usuario;
    string senha = user.Senha;

    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta sobre uma senha vazia")]
public void EntaoDevoVisualizarOAlertaSobreUmaSenhaVazia(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
}

[When(@"realizar um login com usuário ""(.*)"" e senha """(.*)""")]
public void QuandoRealizarUmLoginComUsuarioESenha(string usuario, string senha)
{
    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}

[Then(@"devo visualizar o alerta com a mensagem """(.*)""")]
public void EntaoDevoVisualizarOAlertaComAMensagem(string message)
{
    Assert.AreEqual(message, new LoginPage(driver)
        .VerificarMensagemAlertaErro());
}
}

```

Tente executar algum teste novamente e veja seu funcionamento.

Com isso qualquer outra nova classe de steps pode chamar o driver criado no hooks usando este construtor.

5.23 Incluindo o arquivo .runsettings

Vamos incluir o arquivo runsettings neste projeto para deixar os parâmetros de configuração externos como já foi explicado no capítulo anterior sobre o [arquivo de configurações runsettings](#).

Inclua um novo arquivo de runsettings neste projeto. Irei chamar o meu de **AutomacaoBDD.runsettings** e terá o mesmo conteúdo do que criamos no capítulo anterior.

```
<?xml version="1.0" encoding="utf-8" ?>
<RunSettings>
    <TestRunParameters>
        <Parameter name="browser" value="Chrome" />
    </TestRunParameters>
</RunSettings>
```

Aponte seu projeto para ler este arquivo pelo menu Teste > Configurações de Teste > Seleccione o arquivo .runsettings criado em seu projeto.

E atualize o BeforeScenario para ler a propriedade TestContext. Ela é inicializada como um parâmetro na função:

```
[BeforeScenario]
public void BeforeScenario(TestContext test)
{
    string browser = test.Properties["browser"].ToString();
    driver = CriaDriver(browser);
    driver.Manage().Window.Maximize();
    objectContainer.RegisterInstanceAs<IWebDriver>(driver);
}
```

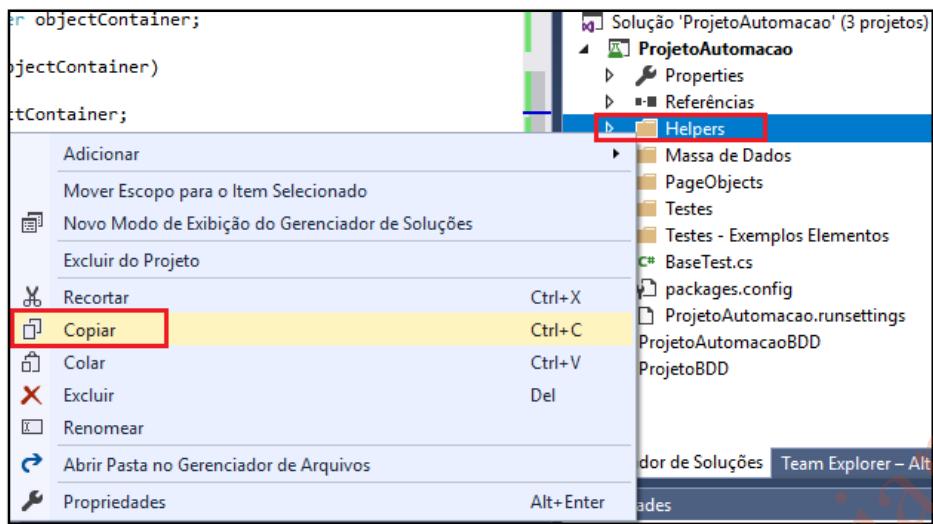
Também poderíamos inicializar outros contexts como o ScenarioContext e FeatureContext nesta função.

5.24 Screenshots

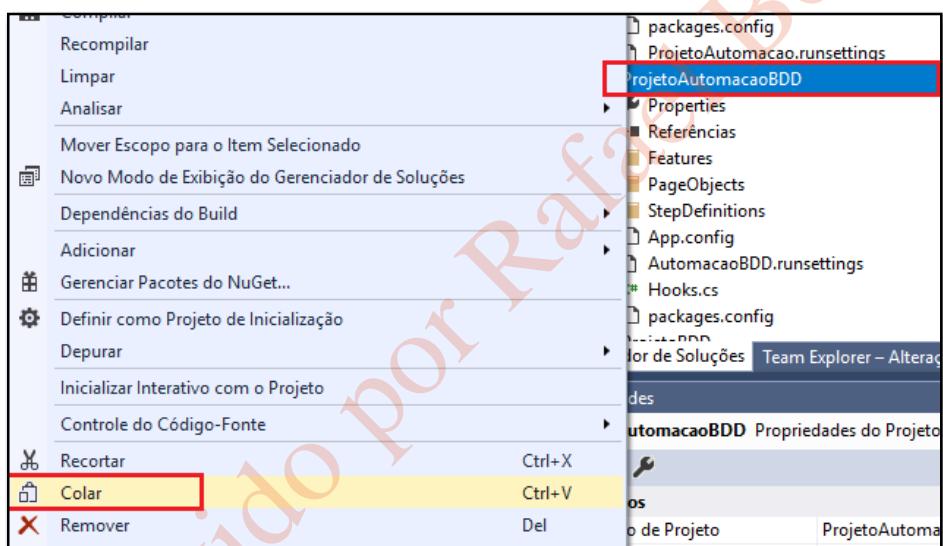
Já temos os testes sendo executados, mas ainda não temos registro de suas evidências. Como também já fizemos a classe responsável por esta parte vamos copiá-la do outro projeto e inclui-la aqui como fizemos com as classes de Page objects.

A classe é **ScreenshotHelper** e está dentro da pasta **Helpers**.

Copie a pasta do outro projeto:



E cole no projeto atual:



Após colar lembre-se de atualizar seu namespace para o projeto atual.

Da mesma forma que fizemos no outro projeto, temos que criar uma pasta de evidências antes da execução dos testes. Podemos incluir a função que cria esta pasta no BeforeTestRun que é executado antes de todas as features.

Em seu arquivo Hooks, inclua:

```
[BeforeTestRun]
public static void BeforeTestRun()
{
    ScreenshotHelper.CriaPastaEvidencias();
}
```

Caso queira que sempre seja registrada uma evidência após o final de cada cenário, podemos incluir a captura no AfterScenario do arquivo hooks desta forma:

```
[AfterScenario]
public void AfterScenario(TestContext test)
```

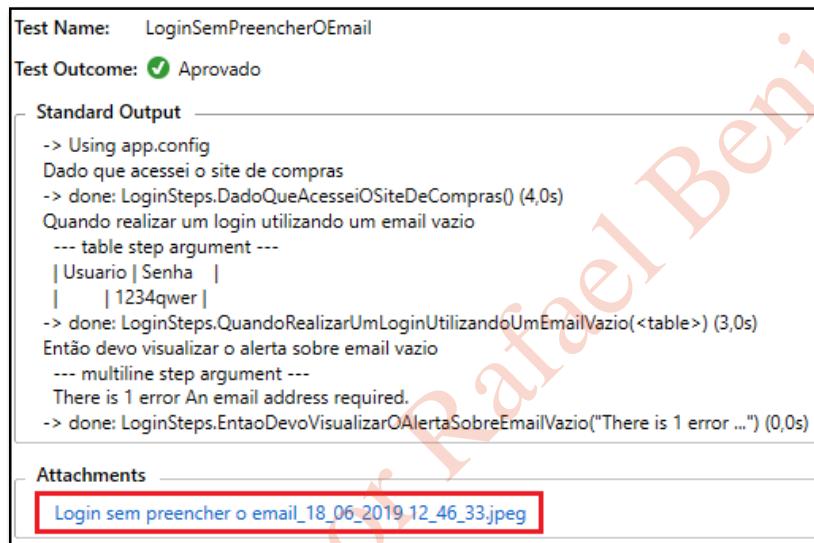
```

    {
        test.AddResultFile(ScreenshotHelper.TiraPrint(ScenarioContext.Current.ScenarioInfo.Title,
driver));
        driver.Quit();
        driver.Dispose();
    }

```

O TestContext que irá anexar as imagens no resultado do teste, foi inicializado como um parâmetro desta função, e para obter o nome do cenário que é usado para armazenar o nome do cenário usei o ScenarioContext.Current.ScenarioInfo.Title.

Agora após a execução de um teste uma evidência com o nome do cenário é anexada:



Além desta forma, temos outra opção. Poderíamos registrar uma evidência ao final de cada passo usando o AfterStep:

```

[AfterStep]
public void AfterStepWeb(ScenarioContext scenario, TestContext test)
{
    string tipoPasso =
ScenarioContext.Current.StepContext.StepInfo.StepInstance.Keyword.ToString();
    string descricaoPasso = ScenarioContext.Current.StepContext.StepInfo.Text;
    test.AddResultFile(ScreenshotHelper.TiraPrint(tipoPasso + descricaoPasso, driver));
}

```

Execute o cenário de Login válido e veja o resultado:

Test Name: LoginComUsuarioValido

Test Outcome: Aprovado

Standard Output

```
-> Using app.config
Dado que accesei o site de compras
-> done: LoginSteps.DadoQueAcesseiOSiteDeCompras() (5,0s)
Quando realizar um login utilizando um usuário válido
--- table step argument ---
| Usuario | Senha |
| teste@testing.com | 1234qwer |
-> done: LoginSteps.QuandoRealizarUmLoginUtilizandoUmUsuarioValido(<table>) (3,7s)
Então devo visualizar a área logada do site
-> done: LoginSteps.EntaoDevoVisualizarAAreaLogadaDoSite() (0,0s)
```

Attachments

[Dado que accesei o site de compras_18_06_2019 23_04_56.jpeg](#)

[Quando realizar um login utilizando um usuário válido_18_06_2019 23_05_00.jpeg](#)

[Então devo visualizar a área logada do site_18_06_2019 23_05_00.jpeg](#)

Agora tente executar algum dos testes que estavam no esquema do cenário, como o *LoginCenariosNegativos_EmailInvalido*:

Test Name: LoginCenariosNegativos_EmailInvalido

Test Outcome: Com falha

Message: Test method ProjetoAutomacaoBDD.Features.LoginFeature.LoginCenariosNegativos_EmailInvalido threw exception:
System.ArgumentException: Caracteres inválidos no caminho.

Standard Output

```
-> Using app.config
Dado que accesei o site de compras
-> done: LoginSteps.DadoQueAcesseiOSiteDeCompras() (4,4s)
Quando realizar um login com usuário "teste@a.com" e senha "12312313"
-> done: LoginSteps.QuandoRealizarUmLoginComUsuarioESenha("teste@a.com", "12312313") (2,8s)
```

Attachments

[Dado que accesei o site de compras_18_06_2019 23_08_55.jpeg](#)

Veja que ao tentar salvar a imagem para o passo **Quando realizar um login com usuário "teste@a.com" e senha "12312313"**, foi exibido o erro: **Caracteres inválidos no caminho**.

Repare que este passo possui aspas duplas, que é um caractere inválido para usar em nomes de arquivos no Windows:

Os nomes de arquivo não podem conter nenhum dos seguintes caracteres:
 \ / : * ? < > |

Neste caso vou optar por dar um **replace** sempre que encontrar um destes caracteres na minha função que salva os screenshots.

```
public static string TiraPrint(string nomeDaImagem, IWebDriver driver)
{
    StringBuilder nome = new StringBuilder(nomeDaImagem);
    nome.Replace("\\", "");
    nome.Replace("/", "");
    nome.Replace(":", "");
    nome.Replace("*", "");
    nome.Replace("?", "");
    nome.Replace("\", "");
    nome.Replace("<", "");
    nome.Replace(">", "");
    nome.Replace("|", "");
```

```

StringBuilder timeStamp = new StringBuilder(DateTime.Now.ToString());
timeStamp.Replace("/", "_");
timeStamp.Replace(":", "_");

string caminhoAssembly = System.Reflection.Assembly.GetCallingAssembly().CodeBase;

caminhoAssembly = new Uri(caminhoAssembly).LocalPath;
string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../.."));
string caminhoDaFoto = novoCaminho + "Evidencias\" + nome + "_" + timeStamp + ".jpeg";

ITakesScreenshot camera = driver as ITakesScreenshot;
Screenshot foto = camera.GetScreenshot();

foto.SaveAsFile(caminhoDaFoto, ScreenshotImageFormat.Jpeg);

return caminhoDaFoto;
}

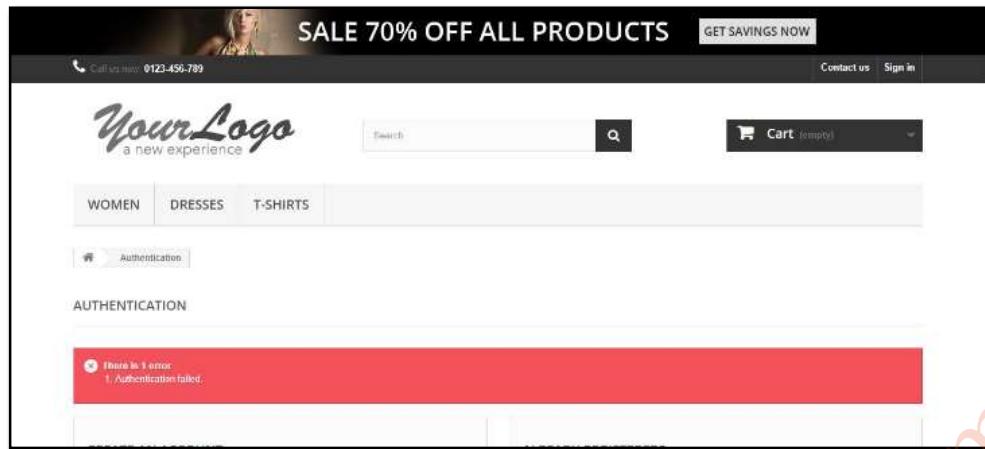
```

Após atualizar minha função e executar o teste novamente, agora podemos ter os prints de cada passo:

Test Name:	LoginCenariosNegativos_EmailInvalido
Test Outcome:	Aprovado
Standard Output	
<pre> -> Using app.config Dado que accesei o site de compras -> done: LoginSteps.DadoQueAcesseiOSiteDeCompras() (4,5s) Quando realizar um login com usuário "teste@a.com" e senha "12312313" -> done: LoginSteps.QuandoRealizarUmLoginComUsuarioESenha("teste@a.com", "12312313") (3,3s) Então devo visualizar o alerta com a mensagem "There is 1 error Authentication failed." -> done: LoginSteps.EntaoDevoVisualizarOAlertaComAMensagem("There is 1 error ...") (0,0s) </pre>	
Attachments	
<p>Dado que accesei o site de compras_18_06_2019 23_26_27.jpeg</p> <p>Quando realizar um login com usuário teste@a.com e senha 12312313_18_06_2019 23_26_31.jpeg</p> <p>Então devo visualizar o alerta com a mensagem There is 1 error Authentication failed._18_06_2019 23_26_31.jpeg</p>	

Obs: Tome cuidado com estes nomes extensos, lembre-se que o Windows possui um limite de 260 caracteres para arquivos, e caso seu projeto esteja em alguma pasta que já possua um caminho grande isto pode ser uma limitação.

Porém utilizar desta forma nem sempre garante uma evidência adequada. A imagem do passo “Quando realizar um login com usuário...” Não capturou os campos email e senha preenchidos.



Para contornar... Poderia quebrar ainda mais os passos:

De:

Esquema do Cenário: Login cenários negativos
Dado que acessei o site de compras
Quando realizar um login com usuário <usuario> e senha <senha>
Então devo visualizar o alerta com a mensagem <mensagem>

Para:

Esquema do Cenário: Login cenários negativos
Dado que acessei o site de compras
Quando informar o usuário <usuario> e senha <senha>
E realizar o login
Então devo visualizar o alerta com a mensagem <mensagem>

Assim divido meu passo mantendo o preenchimento dos campos em um passo, e ação de clicar no passo 'realizar o login' em outro. Mas caso isso não seja possível... Poderia incluir a chamada para função que tira o screenshot após minha ação dentro do step definition, que está desta forma:

```
[When(@"realizar um login com usuário \"(.*)\" e senha \"(.*)\"")]
public void QuandoRealizarUmLoginComUsuarioESenha(string usuario, string senha)
{
    HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(usuario, senha);
}
```

Aqui temos outro ponto...da forma como fiz a chamada da função para logar, criei ela usando a forma funcional e por isso os passos de preencher o email, preencher a senha e clicar no botão para logar estão juntos na função 'Logar'. Posso chamar a função que captura os screenshots dentro do Page objects... Mas para que a imagem seja anexada ao resultado, preciso passar o **Test Context** como parâmetro, e sua instância é criada somente no arquivo hooks.

E para que este seja lido em um step context temos que move-lo através do [Context Injection](#) do arquivo hooks para nossa classe de step definition.

Primeiro vamos criar uma variável a nível de classe dentro do Hooks;

```
private TestContext testContext;
```

Depois vamos atualizar essa variável com o valor que é criado no beforeScenario

```
[BeforeScenario]
```

```

public void BeforeScenario(TestContext test)
{
    string browser = test.Properties["browser"].ToString();
    driver = CriaDriver(browser);
    driver.Manage().Window.Maximize();
    objectContainer.RegisterInstanceAs<IWebDriver>(driver);
    testContext = test;
}

```

E finalmente, vamos incluir outro construtor para as classes que queira receber alem do driver, o testContext:

```

public Hooks(IObjectContainer objectContainer, TestContext test)
{
    this.objectContainer = objectContainer;
    this.testContext = test;
}

```

O arquivo hooks fica então desta forma com dois construtores, caso só chamando no meu step definition o que caber a ele:

```

[Binding]
public sealed class Hooks
{
    private IWebDriver driver;
    private TestContext testContext;
    private readonly IObjectContainer objectContainer;

    public Hooks(IObjectContainer objectContainer)
    {
        this.objectContainer = objectContainer;
    }

    public Hooks(IObjectContainer objectContainer, TestContext test)
    {
        this.objectContainer = objectContainer;
        this.testContext = test;
    }

    [BeforeTestRun]
    public static void BeforeTestRun()
    {
        ScreenshotHelper.CriaPastaEvidencias();
    }

    [BeforeScenario]
    public void BeforeScenario(TestContext test)
    {
        string browser = test.Properties["browser"].ToString();
        driver = CriaDriver(browser);
        driver.Manage().Window.Maximize();
        objectContainer.RegisterInstanceAs<IWebDriver>(driver);
        testContext = test;
    }

    [AfterScenario]
    public void AfterScenario(TestContext test)
    {

//test.AddResultFile(ScreenshotHelper.TiraPrint(ScenarioContext.Current.ScenarioInfo.Title, driver));
        driver.Quit();
        driver.Dispose();
    }
}

```

```

    }

    [AfterStep]
    public void AfterStepWeb(ScenarioContext scenario, TestContext test)
    {
        string tipoPasso =
ScenarioContext.Current.StepContext.StepInfo.StepInstance.Keyword.ToString();
        string descricaoPasso = ScenarioContext.Current.StepContext.StepInfo.Text;
        test.AddResultFile(ScreenshotHelper.TiraPrint(tipoPasso + descricaoPasso, driver));
    }

    public IWebDriver CriaDriver(string browser)
    {
        switch (browser.ToUpper())
        {
            case "FIREFOX":
                driver = new FirefoxDriver();
                return driver;
            case "CHROMEHEADLESS":
                var options = new ChromeOptions();
                options.AddArgument("headless");
                driver = new ChromeDriver(options);
                return driver;
            case "CHROME":
                driver = new ChromeDriver();
                return driver;
            case "IE":
                driver = new InternetExplorerDriver();
                return driver;
            default:
                throw new AssertFailedException("Driver informado: " + browser + " Informe um
driver valido");
        }
    }
}

```

Agora na classe LoginSteps atualizo para o seguinte, crio uma variável a nível de classe para o testContext, e altero o construtor para receber o novo parâmetro do testContext. Caso não houver essa necessidade em algum novo step definition posso manter o construtor que recebe apenas o driver.

```

private IWebDriver driver;
private TestContext test;

public LoginSteps(IWebDriver driver, TestContext test)
{
    this.driver = driver;
    this.test = test;
}

```

Na minha chamada da função para realizar o login agora fica desta forma, criando um novo parâmetro para receber este testContext:

```

HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
home.AcessarLogin()
    .Logar(usuario, senha, test);

```

E dentro da função .Logar() , finalmente com este novo parâmetro do testContext é possível anexar a evidência tirada após o preenchimento dos campos, e antes do clique no botão:

```

public HomeLogadaPage Logar(string email, string senha, TestContext test)
{

```

```

        txtEmail.SendKeys(email);
        txtSenha.SendKeys(senha);
        test.AddResultFile(ScreenshotHelper.TiraPrint("Campos Preenchidos", driver));
        btnSignIn.Click();
        return new HomeLogadaPage(driver);
    }
}

```

Caso você esteja utilizando a forma estrutual fica assim, sem a necessidade de incluir a função de print dentro da Page:

```

HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
home.AcessarLogin()
    .PreencherEmail(usuario)
    .PreencherSenha(senha);
test.AddResultFile(ScreenshotHelper.TiraPrint("Campos Preenchidos", driver));
LoginPage login = new LoginPage(driver);
login.ClicarSignIn();

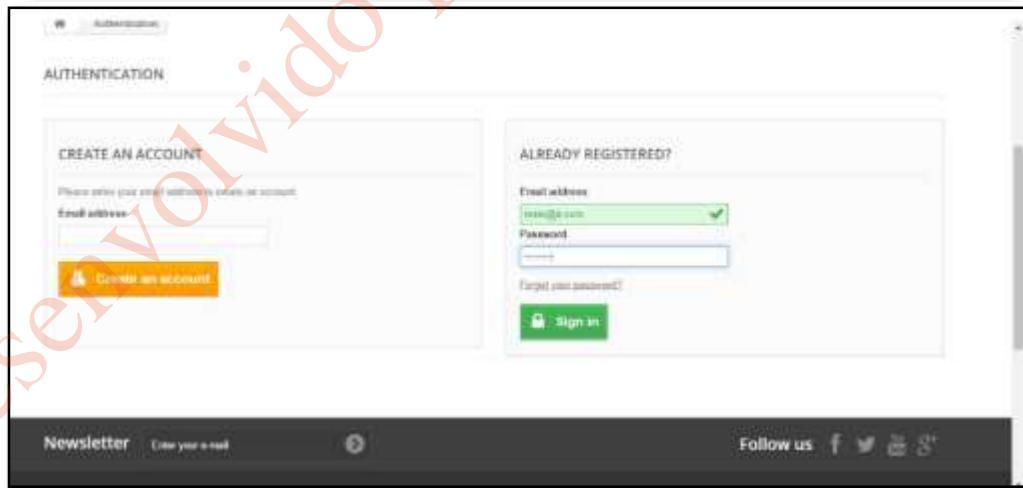
```

E agora após a execução do teste:

Test Name: LoginCenariosNegativos_EmailInvalido

Test Outcome: Aprovado

- Standard Output
 - > Using app.config
 - Dado que accesei o site de compras
 - > done: LoginSteps.DadoQueAcesseiOSiteDeCompras() (4,2s)
 - Quando realizar um login com usuário "teste@a.com" e senha "12312313"
 - > done: LoginSteps.QuandoRealizarUmLoginComUsuarioESenha("teste@a.com", "12312313") (4,0s)
 - Então devo visualizar o alerta com a mensagem "There is 1 error Authentication failed."
 - > done: LoginSteps.EntaoDevoVisualizarOAlertaComAMensagem("There is 1 error ...") (0,0s)
- Attachments
 - Dado que accesei o site de compras_19_06_2019 00_36_36.jpeg
 - Campos Preenchidos_19_06_2019 00_36_39.jpeg
 - Quando realizar um login com usuário teste@a.com e senha 12312313_19_06_2019 00_36_40.jpeg
 - Então devo visualizar o alerta com a mensagem There is 1 error Authentication failed._19_06_2019 00_36_41.jpeg



5.25 Lendo de um arquivo Excel

Para exemplificar um caso aonde os dados são lidos de uma fonte externa, neste caso um arquivo excel já que não temos um banco de dados disponível. Inclua mais dois cenários na feature de login como exemplos:

```
#Lendo os dados pelo excel
Cenário: Login com usuário válido, dados do Excel
    Dado que posso um usuário "válido"
    E que accessei o site de compras
    Quando realizar um login com os dados que foram lidos
    Então devo visualizar a área logada do site

Cenário: Login com um email inválido, dados do Excel
    Dado que posso um usuário "inválido"
    E que accessei o site de compras
    Quando realizar um login com os dados que foram lidos
    Então devo visualizar o alerta sobre email inválido
    """
        There is 1 error Authentication failed.
    """
```

Note que inclui um novo passo onde irei ler os dados da fonte externa, e mudei o passo para realizar o login já que o passo antigo está lendo os dados de uma tabela. Mudei este passo para que ele pegue os dados que iremos ler no primeiro passo. E para tentar não misturar com os exemplos anteriores, gere uma nova classe de stepDefinition para estes dois passos, **LoginExcelSteps**.

Os passos em que é feita a navegação e a validação não irão ser alterados, já que não compartilham nenhum dado e são independentes.

```
[Binding]
public class LoginExcelSteps
{
    [Given(@"que posso um usuário ""(.*)""")]
    public void DadoQuePossuoUmUsuario(string p0)
    {
        ScenarioContext.Current.Pending();
    }

    [When(@"realizar um login com os dados que foram lidos")]
    public void QuandoRealizarUmLoginComOsDadosQueForamLidos()
    {
        ScenarioContext.Current.Pending();
    }
}
```

Agora para ler os dados do excel, crie uma classe dentro de sua pasta Helper com o nome **ExcelReaderHelper**, e substitua seu conteúdo pelo abaixo (Não se esqueça de atualizar o namespace para ficar de acordo com seu projeto).

```
using System;
using System.Data;
using System.Data.OleDb;
using System.IO;

namespace ProjetoAutomacaoBDD.Helpers
{
    public class ExcelReaderHelper
    {

        /// <summary>
        /// Execute the query in the file to retreive the specified data
        /// </summary>
        /// <param name="excelFilePath">Path to the excel file</param>
        /// <param name="queryCommand">The query to be executed in the sheet</param>
```

```

/// <returns>The result dataset</returns>
public static DataSet QueryFromSheet(string excelFilePath, string queryCommand)
{
    return SheetReader(excelFilePath, "default", queryCommand);
}

/// <summary>
/// Read the specified file
/// </summary>
/// <param name="excelFilePath">Path to the excek file</param>
/// <param name="sheetName">Name of the sheet to be read</param>
/// <param name="queryCommand">The query to be executed in the sheet</param>
/// <returns>The result dataset</returns>
public static DataSet SheetReader(string excelFilePath, string sheetName, string
queryCommand)
{
    DataSet dataSet;
    dataSet = new DataSet();

    if (!File.Exists(excelFilePath))
        throw new Exception(string.Format("Arquivo não encontrado: {0}", excelFilePath), new
FileNotFoundException());

    string connectionString =
    string.Format("Provider=Microsoft.ACE.OLEDB.12.0;Data Source={0};Extended
Properties=\"Excel 12.0 Xml;HDR=YES\";", excelFilePath);

    using (var connection = new OleDbConnection(connectionString))
    {
        try
        {
            connection.Open();
            var command = new OleDbCommand(queryCommand, connection);
            OleDbDataAdapter adapter = new OleDbDataAdapter(command);
            adapter.Fill(dataSet, sheetName);
            if (adapter == null)
                throw new Exception(string.Format("Nenhum dado retornado do arquivo: {0}", excelFilePath));
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            connection.Close();
        }
    }
    return dataSet;
}
}

```

É o código de uma classe genérica para ler dados do excel (achei na internet ☺) diferente do que fizemos no projeto anterior esta não tem suporte ao data driven, já que não está vinculada ao framework de testes.

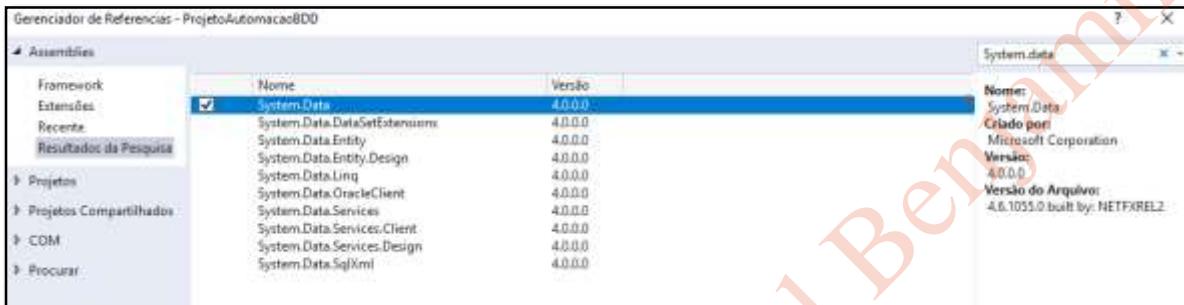
Obs: O specflow possui uma biblioteca que permite ler do excel, permite inclusive manter todo o arquivo feature dentro do excel. Porém como é paga vamos utilizar esta genérica. Ref: <https://specflow.org/plus/excel/>

Após colar esta classe em seu projeto, note que o `using System.Data.OleDb;` está com erro.

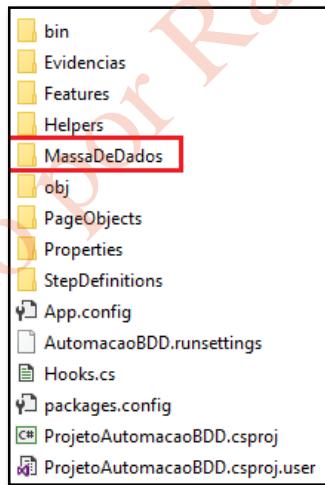
```
using System;
using System.Data;
using System.Data.OleDb; // Erro
using System.IO;
```

Para corrigir, clique com o botão direito em Referências > Adicionar Referência

E procure por System.Data. Após adicionar o erro será corrigido.



Agora abra a pasta do projeto no Windows, e crie uma pasta chamada 'MassaDeDados'. Dentro desta pasta crie um arquivo excel chamado 'Login'



Dentro deste arquivo crie as colunas, 'Tipo', 'Usuario' e 'Senha'. Na coluna tipo informaremos o tipo definido no primeiro passo da feature. E seu respectivo usuário e senha.

A	B	C
Tipo	Usuario	Senha
válido	teste@testing.com	1234qwer
inválido	teste@a.com	1234qwer

Agora precisamos ler os dados da planilha. Para isso iremos utilizar a função da classe ExcelReaderHelper.

```
ExcelReaderHelper.QueryFromSheet()
```

Esta função pede dois parâmetros, o primeiro é o caminho onde a planilha se encontra, e o segundo a query que iremos executar para buscar o dado dentro dela.

Para ler o caminho da planilha, irei utilizar os mesmos passos para obter o caminho da pasta de evidências, so que agora apontando para a pasta de massa de dados:

```
string caminhoAssembly = System.IO.Directory.GetCurrentDirectory();
caminhoAssembly = new Uri(caminhoAssembly).LocalPath;
string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../"));
string caminhoPlanilha = novoCaminho + "MassaDeDados\\" + "Login.xlsx";
```

E a query é semelhante a que usamos em um banco de dados:

```
string query = "SELECT * FROM [Plan1$] where [Tipo] = '" + tipo + "'";
```

Iremos filtrar a linha de acordo com a coluna Tipo, e este é recebido pelo parâmetro do primeiro passo. A ‘tabela’ é na verdade o nome da aba onde estão os dados.

Após o retorno da query pela função, ela irá retornar um DataSet, que é um conjunto de tabelas. Mas como nossa query irá retornar uma única linha será mais simples de trabalhar com esse dataset.

Para não ficar lendo este dataset n vezes, iremos alimentar seus dados em um objeto, que por sua vez poderá ser utilizado em toda a classe, ou compartilhado para outra através do context injection.

Crie uma nova classe dentro de helpers chamada **LoginFieldsHelper**. Dentro dela inclua os campos que temos na planilha com exceção do tipo, que não iremos utilizar nos testes, ele só foi utilizado para servir de referência na hora de buscar o registro da planilha.

```
public class LoginFieldsHelper
{
    public string usuario;
    public string senha;
}
```

Agora na classe LoginExcelSteps, inclua uma variável de classe para o LoginFieldsHelper:

```
LoginFieldsHelper loginFields;
```

E com o resultado o data set iremos alimentá-la dentro do step:

```
DataSet ds = ExcelReaderHelper.QueryFromSheet(caminhoPlanilha, query);

loginFields = new LoginFieldsHelper();
loginFields.usuario = ds.Tables[0].Rows[0]["Usuario"].ToString();
loginFields.senha = ds.Tables[0].Rows[0]["Senha"].ToString();
```

Como o retorno será sempre uma tabela e uma linha, podemos deixar fixo no índice zero a tabela e a linha.

Visualização do dataset pelo modo debug:

<code>DataSet ds = ExcelReaderHelper.QueryFromSheet(caminhoPlanilha, query);</code>
DataSet Visualizador
Tabela: default

E os parâmetros recebidos pelo “Logar” agora serão as propriedades desta classe LoginFieldsHelper contendo os dados lidos da query.

```
HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
    home.AcessarLogin()
        .Logar(loginFields.usuario, loginFields.senha, testContext);
```

A classe de step então fica da seguinte forma:

```
namespace ProjetoAutomacaoBDD.StepDefinitions
{
    [Binding]
    public class LoginExcelSteps
    {
        private IWebDriver driver;
        private TestContext test;
        LoginFieldsHelper loginFields;

        public LoginExcelSteps(IWebDriver driver, TestContext test)
        {
            this.driver = driver;
            this.test = test;
        }

        [Given(@"que possuo um usuário ""(.*)""")]
        public void DadoQuePossuoUmUsuario(string tipo)
        {
            string caminhoAssembly = System.IO.Directory.GetCurrentDirectory();
            caminhoAssembly = new Uri(caminhoAssembly).LocalPath;
            string novoCaminho = Path.GetFullPath(Path.Combine(caminhoAssembly, @"../../"));
            string caminhoPlanilha = novoCaminho + "MassaDeDados\\" + "Login.xlsx";

            string query = "SELECT * FROM [Plan1$] where [Tipo] = '" + tipo + "'";

            DataSet ds = ExcelReaderHelper.QueryFromSheet(caminhoPlanilha, query);

            loginFields = new LoginFieldsHelper();
            loginFields.usuario = ds.Tables[0].Rows[0]["Usuario"].ToString();
            loginFields.senha = ds.Tables[0].Rows[0]["Senha"].ToString();
        }

        [When(@"realizar um login com os dados que foram lidos")]
        public void QuandoRealizarUmLoginComOsDadosQueForamLidos()
        {

            HomeNaoLogadaPage home = new HomeNaoLogadaPage(driver);
            home.AcessarLogin()
                .Logar(loginFields.usuario, loginFields.senha, test);
        }
    }
}
```

```
}
```

5.26 Exercicio - Fale Conosco

Usando o que foi apresentado anteriormente, crie uma feature com os testes criados para a página Contact Us. Um cenário enviando uma mensagem válida, e outro enviando uma mensagem inválida.

Criada a feature:

```
#language: pt

Funcionalidade: FaleConosco
    Como um usuário
    Quero poder entrar em contato com a empresa
    Para poder esclarecer dúvidas sobre o site ou pedidos

Cenário: Enviar uma mensagem válida
    Dado que acessei o site de compras
    Quando acessar a página Contact Us
    E realizar o envio de uma mensagem válida
    Então devo visualizar a mensagem de sucesso
    """
    Your message has been successfully sent to our team.
    """

Cenário: Enviar uma mensagem válida informando um email inválido
    Dado que acessei o site de compras
    Quando acessar a página Contact Us
    E realizar o envio de uma mensagem informando um email inválido
    Então devo visualizar a mensagem de email inválido
    """
    There is 1 error Invalid email address.
    """
```

E a implementação de seu step definition:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using ProjetoAutomacaoBDD.Helpers;
using ProjetoAutomacaoBDD.PageObjects;
using TechTalk.SpecFlow;

namespace ProjetoAutomacaoBDD
{
    [Binding]
    public class FaleConoscoSteps
    {
        private IWebDriver driver;
        private TestContext test;
        string textoRetornado;

        public FaleConoscoSteps(IWebDriver driver, TestContext test)
        {
            this.driver = driver;
            this.test = test;
        }
    }
```

```

[When(@"acessar a página Contact Us")]
public void QuandoAcessarAPaginaContactUs()
{
    HomeNaoLogadaPage homePage = new HomeNaoLogadaPage(driver);
    homePage.AcessarContactUs();
}

[When(@"realizar o envio de uma mensagem válida")]
public void QuandoRealizarOEnvioDeUmaMensagemValida()
{
    ContactUsPage contactPage = new ContactUsPage(driver);
    contactPage.PreencherComboAssunto("Customer service")
        .PreencherEmail("teste@teste.com")
        .PreencherNumeroPedido("12346579")
        .PreencherMensagem("Mensagem de Teste");

    test.AddResultFile(ScreenshotHelper.TiraPrint("Campos preenchidos", driver));

    textoRetornado = contactPage.ClicarEnviar()
        .VerificarMensagemSucesso();
}

[When(@"realizar o envio de uma mensagem informando um email inválido")]
public void QuandoRealizarOEnvioDeUmaMensagemInformandoUmEmailInvalido()
{
    ContactUsPage contactPage = new ContactUsPage(driver);

    contactPage.PreencherComboAssunto("Customer service")
        .PreencherEmail("emailinvalido")
        .PreencherNumeroPedido("12346579")
        .PreencherMensagem("Mensagem de Teste");

    test.AddResultFile(ScreenshotHelper.TiraPrint("Campos preenchidos", driver));

    textoRetornado = contactPage.ClicarEnviar()
        .VerificarMensagemAlertaErro();
}

[Then(@"devo visualizar a mensagem de sucesso")]
public void EntaoDevoVisualizarAMensagemDeSucesso(string mensagemEsperada)
{
    Assert.AreEqual(mensagemEsperada, textoRetornado);
}

[Then(@"devo visualizar a mensagem de email inválido")]
public void EntaoDevoVisualizarAMensagemDeEmailInvalido(string mensagemEsperada)
{
    Assert.AreEqual(mensagemEsperada, textoRetornado);
}
}

```

5.27 Exercicio – Cadastro

Também crie a feature com o cenário de cadastro válido.

```
#language: pt

Funcionalidade: Cadastro
```

*Como um novo usuário
Quero poder me cadastrar no site*

Cenário: Realizar um cadastro

Dado que possuo um usuário não cadastrado no site
E que accesei o site de compras
Quando acessar a página de login
E informar o email do novo usuário
E realizar seu cadastro
Então devo visualizar a área logada do site

E a implementação de seu step definition:

```
using Bogus;
using Bogus.DataSets;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using ProjetoAutomacaoBDD.Helpers;
using ProjetoAutomacaoBDD.PageObjects;
using System;
using System.Globalization;
using TechTalk.SpecFlow;

namespace ProjetoAutomacaoBDD
{
    [Binding]
    public class CadastroSteps
    {
        private IWebDriver driver;
        private TestContext test;
        string email;
        string titulo;
        string nome;
        string sobreNome;
        string senha;
        DateTime dataNascimento;
        string diaNascimento;
        string mesNascimento;
        string anoNascimento;
        string empresa;
        string endereco;
        string complemento;
        string cidade;
        string cep;
        string pais;
        string informacoesAdicionais;
        string telefone;
        string celular;
        string identificacaoEndereco;

        public CadastroSteps(IWebDriver driver, TestContext test)
        {
            this.driver = driver;
            this.test = test;
        }

        [Given(@"que possuo um usuário não cadastrado no site")]
        public void DadoQuePossuoUmUsuarioNaoCadastradoNoSite()
        {
            //Criando os dados necessários para o teste
            Faker gerador = new Faker("en_US");
```

```

Name.Gender genero;
genero = gerador.Person.Gender;
if (genero.ToString() == "Male")
{
    titulo = "Mr.";
}
else
{
    titulo = "Mrs.";
}
nome = gerador.Name.FirstName(genero);
sobreNome = gerador.Name.LastName(genero);
email = gerador.Internet.Email(nome, sobreNome);
senha = "1234qwer";
dataNascimento = gerador.Date.Past(70, DateTime.Now.AddYears(-18));
diaNascimento = dataNascimento.Day.ToString();
mesNascimento = new CultureInfo("en-US").DateTimeFormat.GetMonthName(dataNascimento.Month);
anoNascimento = dataNascimento.Year.ToString();
empresa = gerador.Company.CompanyName();
endereco = gerador.Address.StreetAddress();
complemento = gerador.Address.SecondaryAddress();
cidade = gerador.Address.City();
cep = gerador.Address.ZipCode("#####");
pais = "United States";
informacoesAdicionais = gerador.Lorem.Paragraph(1);
telefone = gerador.Phone.PhoneNumber("(###) ###-####");
celular = gerador.Phone.PhoneNumber("(###) #####-####");
identificacaoEndereco = gerador.Address.StreetSuffix();
}

[When(@"acessar a página de login")]
public void QuandoAcessarAPaginaDeLogin()
{
    HomeNaoLogadaPage homenaoLogadaPage = new HomeNaoLogadaPage(driver);
    homenaoLogadaPage.AcessarLogin();
}

[When(@"informar o email do novo usuário")]
public void QuandoInformarOEmailDoNovoUsuario()
{
    LoginPage loginPage = new LoginPage(driver);
    loginPage.PreencherNovoEmail(email)
        .ClicarCriarConta();
}

[When(@"realizar seu cadastro")]
public void QuandoRealizarSeuCadastro()
{
    //Forma Estrutural
    CadastroPage cadastroPage = new CadastroPage(driver);
    cadastroPage.SelecionarTitulo(titulo)
        .PreencherNomeInformacoesPessoais(nome)
        .PreencherSobreNomeInformacoesPessoais(sobreNome)
        .ClicarCampoEmail()
        .PreencherCampoSenha(senha)
        .SelecionarDiaNascimento(diaNascimento)
        .SelecionarMesNascimento(mesNascimento)
        .SelecionarAnoNascimento(anoNascimento)
        .SelecionarRecebimentoNewsletter()
        .SelecionarRecebimentoOfertasEspeciais();
}

```

```
    test.AddResultFile(ScreenshotHelper.TiraPrint("Dados Pessoais preechidos", driver));

    cadastroPage.PreencherEmpresa(empresa)
        .PreencherEndereco(endereco)
        .PreencherComplemento(complemento)
        .PreencherCidade(cidade)
        .SelecionarEstado();

    test.AddResultFile(ScreenshotHelper.TiraPrint("Dados Endereço parte um preechidos",
driver));

    cadastroPage.PreencherCep(cep)
        .SelecionarPais(pais)
        .PreencherInformacoesAdicionais(informacoesAdicionais)
        .PreencherTelefone(telefone)
        .PreencherCelular(celular)
        .PreencherIdentificacaoEndereco(identificacaoEndereco);

    test.AddResultFile(ScreenshotHelper.TiraPrint("Dados Endereço parte dois preechidos",
driver));

    cadastroPage.ClicarRegistrar();
}
}
```

Desenvolvido por Rafael Beniamim

6 Módulo 3 - Integração com Azure DevOps

Com nosso projeto já estruturado, temos que definir um local para que stakeholders do projeto possam executar os testes da forma menos técnica possível. Como já dito anteriormente, a idéia é associar nossos scripts de teste automatizado com os casos de teste que existem dentro do Azure DevOps (Antigo TFS/VSTS). Outra vantagem é que os relatórios poderão ser extraídos diretamente da ferramenta.

Estes testes também podem ser executados em uma pipeline de integração contínua, que é basicamente realizar os testes logo após que um build da aplicação seja feita. Assim facilitando a detecção de possíveis erros com mais agilidade e garantindo que os testes sempre sejam executados.

6.1 Criar conta no Azure DevOps

Acesse o site do Azure Dev Ops: <https://azure.microsoft.com/pt-br/services/devops/>

Caso não tenha uma conta Microsoft clique em **Início gratuito**, e siga as instruções.

Caso possua alguma conta Microsoft clique em **Entrar no Azure DevOps**, e realize o login.

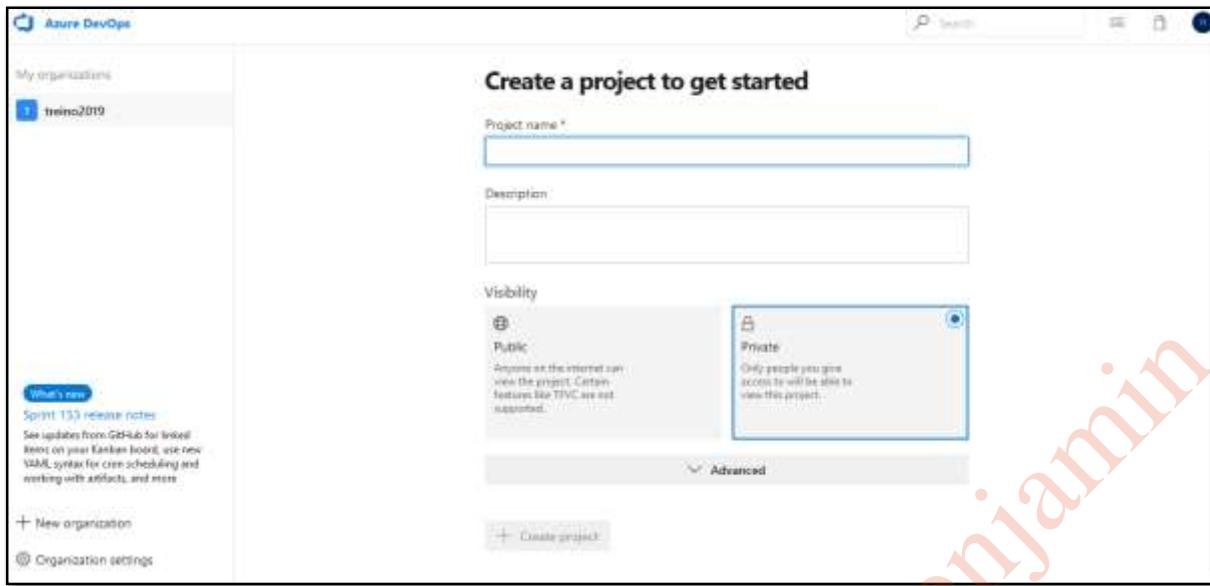
Obs: O módulo de testes que iremos utilizar não é gratuito, mas possui um período de avaliação de 30 dias. Após isto ele não poderá ser reutilizado.

Caso queira explorar suas funcionalidades novamente será necessário criar outra conta da Microsoft para utilizar em um novo período de 30 dias.



6.2 Criando o primeiro projeto

Após seu login, você será direcionado para a página inicial onde criaremos o primeiro projeto. Vamos criar um projeto simples para exemplificar, mas em um projeto real algumas configurações como a ferramenta de versionamento e tipo de projeto podem estar diferentes.



Inclua um nome para seu projeto, se quiser uma descrição. Mantenha seu projeto como Private, já que é apenas para aprendizado.

Clique em **Advanced**

Em **Version Control**, selecione **Team Foundation Version Control** que é um pouco mais simples para aprendizado e já possui integração pelo visual Studio.

Documentação da Microsoft sobre TFVC: <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/?view=azure-devops>

Obs: Versionamento por **Git** também é suportado pelo visual Studio, mas não é o foco de aprendizado neste momento. É muito popular atualmente e recomendo que incluam no seu backlog de estudos. Existem diversos cursos gratuitos para aprender o básico sobre git, irei deixar sugestões abaixo:

Cursos gratuitos na udemy:

<https://www.udemy.com/courses/search/?src=ukw&q=git&p=1&price=price-free>

Cursos gratuitos no youtube:

https://www.youtube.com/results?search_query=curso+git

Em **Work item process**, selecione **Agile**.

Aqui existem 4 tipos de projeto *Basic*, *Agile*, *Scrum* e *CMMI*. Mais informações sobre cada tipo no link:

<https://docs.microsoft.com/pt-pt/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops>

Após preencher os campos clique em **Create Project**

Project name *

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

Advanced

Version control [?](#)

Work item process [?](#)

Team Foundation Version Control Agile

[+ Create project](#)

Projeto criado:

The screenshot shows the Azure DevOps Project Overview page for 'ProjetoTeste'. The left sidebar has 'ProjetoTeste' selected. The main area displays the project name 'ProjetoTeste' and its description 'Projeto para aprendizado'. It includes sections for 'About this project', 'Project stats' (Last 7 days), and 'Members'. The 'Project stats' section shows 1 changeset by 1 author in the last 7 days. The 'Members' section shows one member, 'Tl'.

6.3 Visão Geral do Projeto

Uma breve introdução sobre os menus, estes menus são específicos para o projeto do tipo **Agile**, mas no geral os demais tipos possuem uma visão bem semelhante.

- **Overview**

Possui o resumo do projeto.

O submenu **Dashboards** permite incluir algumas visões de relatórios comuns do projeto.

O submenu **Wiki** permite a criação de uma Wiki local para compartilhamento de informações.

- **Work Items**

Lista todos os itens criados no projeto, pode ser customizado.

O submenu **Boards** disponibiliza uma tabela de kanban para organizar as atividades. O submenu **Backlogs** permite organizar as atividades previstas.

O submenu **Sprints** permite definir as sprints de acordo com as atividades já apontadas.

O submenu **Queries** permite criar queries para busca de workitens, como por exemplo, bugs, atividades, etc..

- **Repos**

O submenu **Files** armazena os repositórios de código fonte, é aqui o lugar onde iremos salvar nosso projeto de automação. Pode ser compartilhado para armazenar outras soluções também relativas ao projeto, como o código da aplicação em desenvolvimento.

O submenu **Changesets** permite visualizar o histórico de alterações realizadas nos código fonte.

O submenu **Shelvesets** permite armazenar comits exclusivos do seu usuário no servidor.

- **Pipelines**

O submenu **Builds** será utilizado para compilar os projetos e gerar as dlls.

O submenu **Releases** é a próxima fase depois dos builds, aqui usaremos o produto gerado para associar aos casos de teste. Em uma aplicação de desenvolvimento é aqui que onde é configurado o deploy em ambientes.

O submenu **Library** é usado para armazenar arquivos importantes e armazenar as variáveis de grupo, que são variáveis utilizadas no build ou release.

O submenu **Task groups** é utilizado para criar ‘templates’ de ações que são utilizadas no build ou release.

O submenu **Deployment Group** serve para agrupa diversos agents de deploy, que são os ambientes físicos utilizados para realizar o build e a release.

- **Test Plans**

Este menu deve vir bloqueado, mas iremos realizar seu desbloqueio e ver em mais detalhes no tópico seguinte.

O submenu **Test Plans** armazena os testes e suas suítes.

O submenu **Runs** armazena as execuções dos testes.

O submenu **Load Test** permite a criação de testes de carga.

- **Artifacts**

Armazena os itens gerados em seu repositório após a compilação.

6.4 Criando os casos de Teste

Para criar os casos de teste, devemos acessar o menu **Test Plans**, que por padrão é bloqueado para os usuário do plano gratuito do Azure DevOps.

Para desbloquear siga os passos deste link: <https://docs.microsoft.com/en-us/azure/devops/organizations/billing/try-additional-features-vs?view=azure-devops>

Boards, Repos and Test Plans	Free	Paid
Basic users	5	0
Basic + Test Plans	Trial expires in 31 days	0

Após o desbloqueio, acesse o menu **Test Plans** novamente e veja que agora está sendo carregado.

The screenshot shows the Azure DevOps interface for the 'ProjetoTeste' project. On the left, there's a sidebar with icons for Boards, Repos, Pipelines, Test Plans, and Artifacts. The 'Test Plans' icon is selected. The main area is titled 'Test Plans' and shows a table with columns: Title, State, Area Path, and Iteration Path. There are two rows: one for 'My Favorites' (empty) and one for 'ProjetoTeste Team'. A red watermark 'Desenvolvido por Rafael Beliamini' is diagonally across the page.

Após isto podemos criar nossos casos de teste. Irei criar uma suíte estática simples contendo os 5 testes que fizemos nos projetos. Em um projeto real, também podemos realizar a associação a casos de teste já existentes.

Documentação de quickstart da Microsoft:

Criar test Plans: <https://docs.microsoft.com/pt-br/azure/devops/test/create-a-test-plan?view=azure-devops>

Criar Test Cases: <https://docs.microsoft.com/pt-br/azure/devops/test/create-test-cases?view=azure-devops>

The screenshot shows the Microsoft Test Manager application. On the left, there's a navigation pane with 'Test Plans > Regressão Loja'. Below it, a tree view shows 'Regressão Loja' expanded, with 'Geral [5]' selected. The main area is titled 'Test suite: Geral (Suite ID: 3)' and shows a table of test cases. The table has columns for 'Outcome', 'Order', 'ID', 'Title', and 'Configuration'. There are 5 rows, all marked as 'Active':

Outcome	Order	ID	Title	Configuration
Active	1	4	Login Válido	Windows ...
Active	2	5	Login Inválido	Windows ...
Active	3	6	Fale Conosco Válido	Windows ...
Active	4	7	Fale Conosco Inválido	Windows ...
Active	5	8	Cadastro	Windows ...

Criei os casos de teste apenas como título e sem incluir a descrição dos passos, já que para a automação o que importa é ter um caso de teste para ser associado.

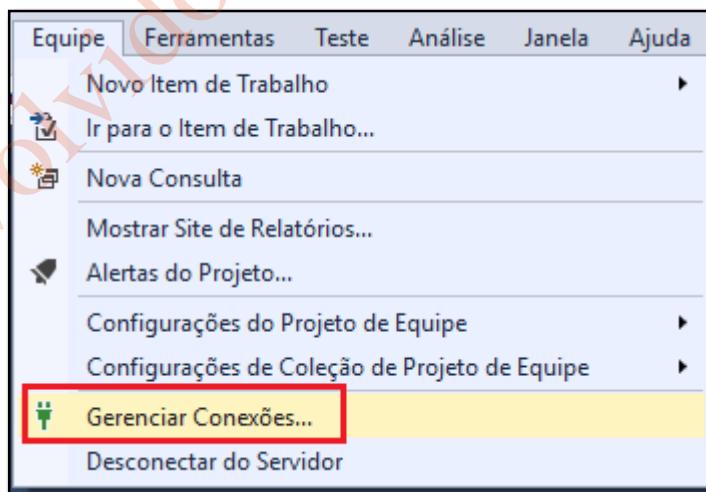
Obs: Mesmo com o BDD, o texto descrito no nosso cenário não é automaticamente atualizado aqui nos testes. Portanto caso for utilizar com lembre-se sempre de que após atualizar um cenário dentro do arquivo feature em seu projeto de automação, será necessário atualizar a descrição do caso de teste aqui também.

6.5 Subindo o projeto no repositório

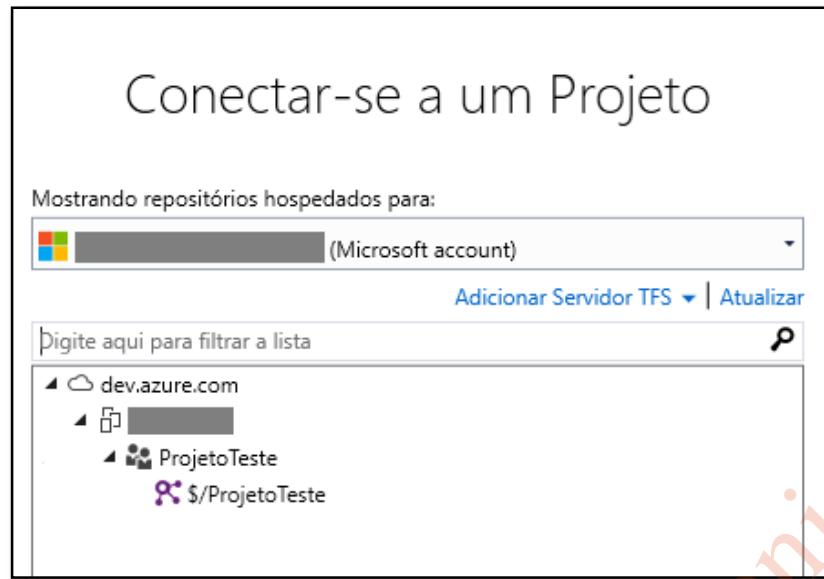
6.5.1 Conectando com o repositório do Azure DevOps

Agora precisamos subir nosso projeto de automação dentro do projeto criado no Azure DevOps.

Primeiro o Visual Studio. Em seguida clique no menu **Equipe > Gerenciar Conexões**



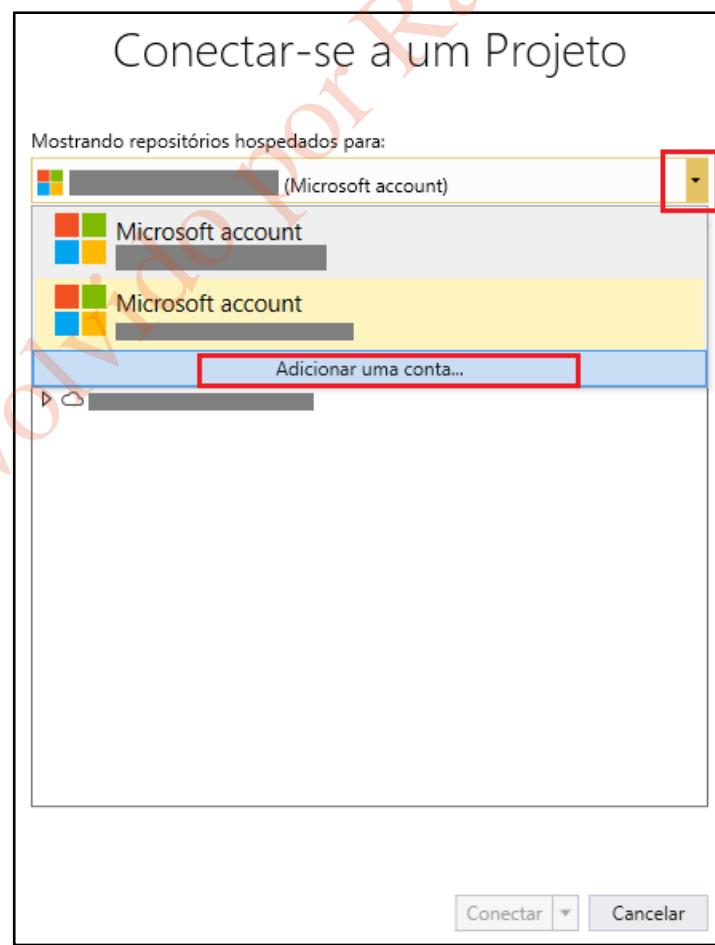
Caso seu Visual Studio já esteja conectado com a mesma conta na qual o projeto foi criado, o projeto será listado automaticamente:



Caso seu Visual Studio não esteja utilizando o mesmo email que o utilizado para criar seu projeto no azure devops, existem duas opções:

- 1 – Adicionar a mesma conta que você está utilizando clique na combo e clique em **Adicionar uma conta...**

Após incluir a conta o projeto será listado.



2 – Dar permissão para o email que está sendo usado em seu Visual Studio para ter acesso ao projeto.

Para dar permissão, Clique em **Project Settings** que está no canto inferior esquerdo do seu projeto no Azure DevOps, Clique em **Teams**, e Clique em **Add**. Insira o email Microsoft e clique em **Save Changes**.

The screenshot shows the 'Project Settings' interface for a project named 'ProjetoTeste'. The left sidebar has a 'Teams' tab selected, which is highlighted with a red box. The main area shows a 'Team Profile' for 'ProjetoTeste Team'. It includes fields for 'Name' (ProjetoTeste Team), 'Description' (The default project team, ...), 'Administrators' (with a '+ Add' link), and sections for 'Manage other settings for this team', 'Notifications', 'Dashboards', and 'Iterations and areas'. On the right, under 'Members', there is a table with two entries. The first entry has a 'Display Name' and a 'Username Or Scope'. A red box highlights the '+ Add...' button. The second entry also has a 'Display Name' and a 'Username Or Scope'. At the bottom right of the page, there is a 'membership' dropdown menu.

Agora ao acessar gerenciar conexões no visual Studio o projeto deve ser listado.

Selecione o projeto e clique em **Conectar**.

A opção para configurar seu espaço de trabalho será exibida. No primeiro item pode manter o que vem selecionado, que é nível dentro do projeto que será mapeado, neste caso vamos mapear deste a raiz. E na segunda opção é o local dentro do seu computador onde serão salvos os itens baixados da primeira opção, caso queira pode apontar para outra pasta que seja melhor para você. Anote este local iremos precisar dele.

Após configurado, clique em **Mapear e Obter**.

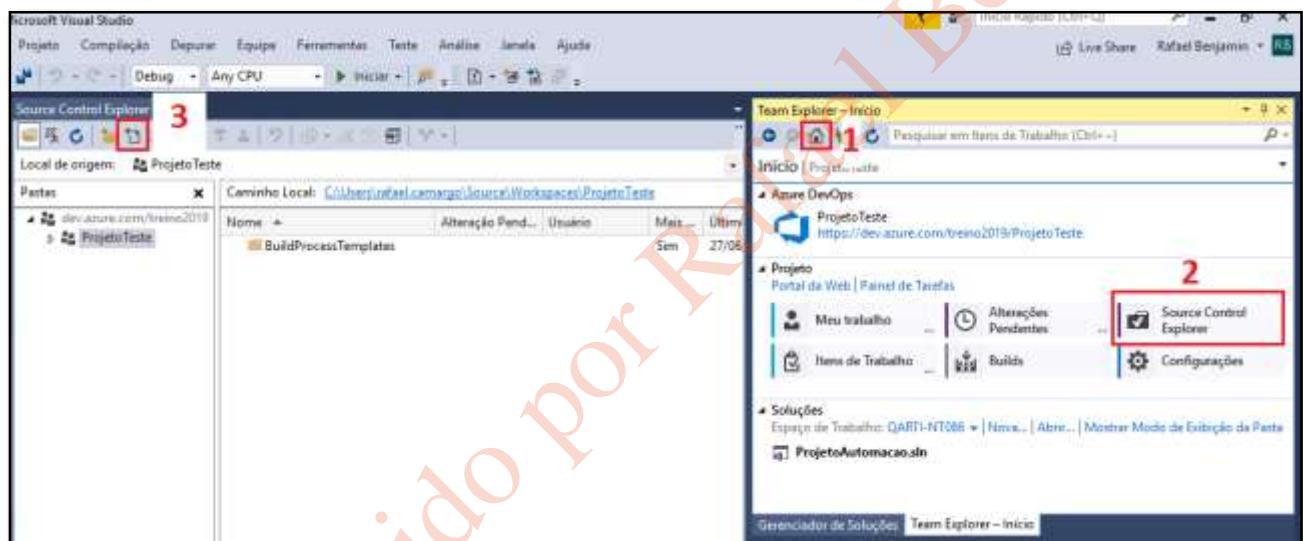
The screenshot shows the 'Team Explorer - Início' window. It displays a list of workspaces under 'Azure DevOps'. One workspace, 'ProjetoTeste', is selected and its URL is shown as 'https://dev.azure.com/treino2019/ProjetoTeste'. Below this, under 'Configurar Espaço de Trabalho', there are two paths: '\$/ProjetoTeste' and 'C:\Users\rafael.camargo\Source\Workspaces\ProjetoTeste'. Both paths are highlighted with a red box. At the bottom of the window, there are three buttons: 'Mapear e Obter' (which is highlighted with a blue box), 'Cancelar', and 'Avançado...'. The window has a yellow header bar with icons for back, forward, search, and refresh.

Referência: <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/set-up-team-foundation-version-control-your-dev-machine?view=azure-devops>

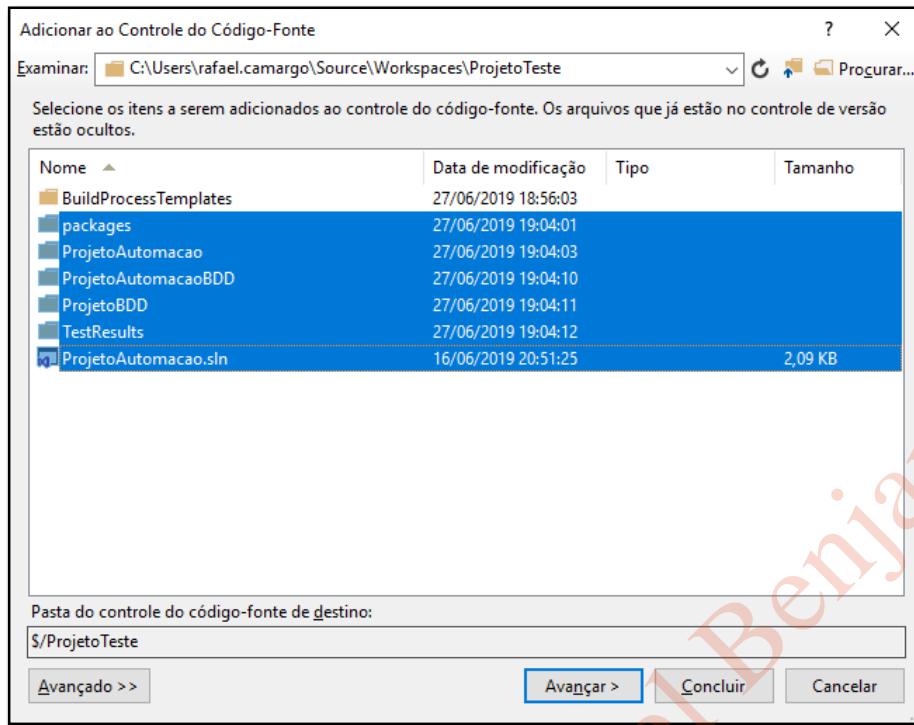
Agora copie os arquivos do projeto de automação para esta pasta que mapeamos.

Nome	Data de modificaç...	Tipo	Tamanho
\$tf	27/06/2019 19:05	Pasta de arquivos	
.vs	27/06/2019 19:03	Pasta de arquivos	
BuildProcessTemplates	27/06/2019 18:56	Pasta de arquivos	
packages	27/06/2019 19:04	Pasta de arquivos	
ProjetoAutomacao	27/06/2019 19:04	Pasta de arquivos	
ProjetoAutomacaoBDD	27/06/2019 19:04	Pasta de arquivos	
ProjetoBDD	27/06/2019 19:04	Pasta de arquivos	
TestResults	27/06/2019 19:04	Pasta de arquivos	
ProjetoAutomacao.sln	16/06/2019 20:51	Visual Studio Solu...	3 KB

Agora precisamos incluir estas pastas dentro do rastreamento. Para isto accese o visual Studio novamente, clique em **Início (ícone de casinha)** no Team Explorer, depois em **Source Control Explorer** e finalmente no ícone de **Adicionar**, conforme figura abaixo:

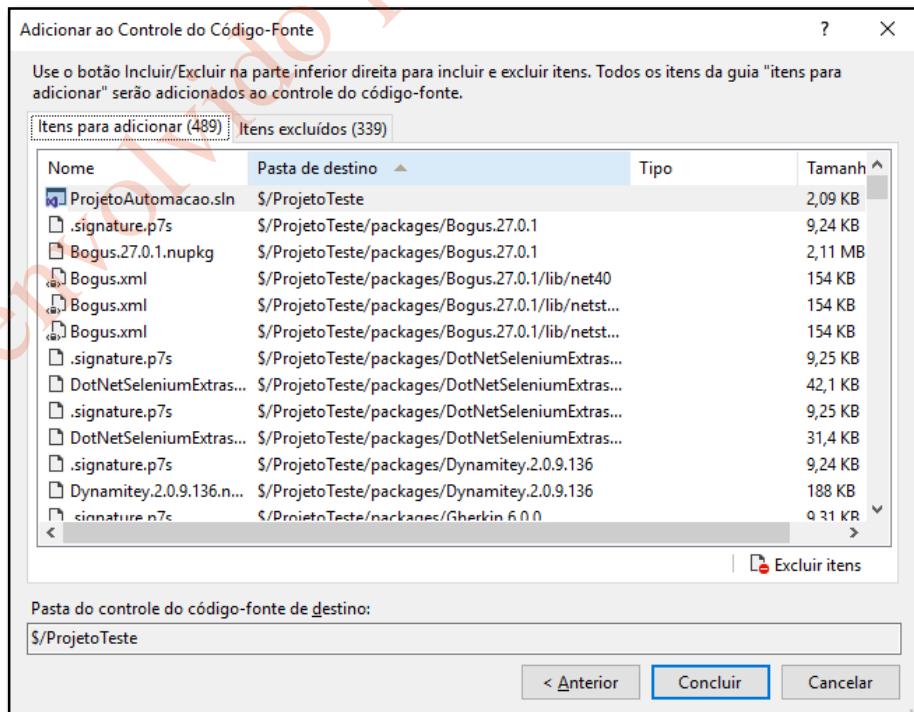


Após o menu ser aberto, selecione todos os itens que pertencem ao seu projeto e clique em **Avançar**:



Aqui serão exibidas as abas **Itens para adicionar**, e **Itens Excluídos**. Repare que aba **Itens Excluídos**, alguns arquivos das pastas **packages**, **bin** e **obj** podem estar sendo listados. O visual Studio entende que estes arquivos são gerados após a compilação (como vimos lá no inicio do projeto) e não devem ser subidos para o repositório. Caso você queira que mais algum arquivo não seja incluído, basta seleciona-los na aba **Itens para adicionar** e clicar em **Excluir Itens**.

Mas como neste caso existem muitos itens que ainda não foram identificado, pode mante-los e clicar em **Concluir**, em seguida irei mostrar uma maneira mais fácil de contornar esta situação.



Source Control Explorer

Local de origem: ProjetoTeste

Pastas

- dev.azure.com/treino2019
 - ProjetoTeste
 - BuildProcessTemplates
 - packages
 - ProjetoAutomacao
 - ProjetoAutomacaoBDD
 - ProjetoBDD
 - TestResults

Caminho Local: C:\Users\rafael.camargo\Source\Workspaces\ProjetoTeste

Nome	Alteração Pend...	Us...	Mais ...	Último Check-in
BuildProcessTemplates	Sim			27/06/2019 16:...
packages	Sim			
ProjetoAutomacao	Sim			
ProjetoAutomacaoBDD	Sim			
ProjetoBDD	Sim			
TestResults	Sim			
ProjetoAutomacao.sln	ra... Sim	adicionar		

Agora os arquivos ficarão com um + ao lado das pastas indicando que são uma nova inclusão ao versionamento.

Clique em **Alterações Pendentes** dentro de **Team Explorer**

Source Control Explorer

Local de origem: ProjetoTeste

Pastas

- dev.azure.com/treino2019
 - ProjetoTeste
 - BuildProcessTemplates
 - packages
 - ProjetoAutomacao
 - ProjetoAutomacaoBDD
 - ProjetoBDD
 - TestResults

Caminho Local: C:\Users\rafael.camargo\Source\Workspaces\ProjetoTeste

Nome	Alteração Pend...	Us...	Mais ...	Último Check-in
BuildProcessTemplates	Sim			27/06/2019 16:...
packages	Sim			
ProjetoAutomacao	Sim			
ProjetoAutomacaoBDD	Sim			
ProjetoBDD	Sim			
TestResults	Sim			
ProjetoAutomacao.sln	ra... Sim	adicionar		

Team Explorer – Início

Início | ProjetoTeste

Azure DevOps

ProjetoTeste
https://dev.azure.com/treino2019/ProjetoTeste

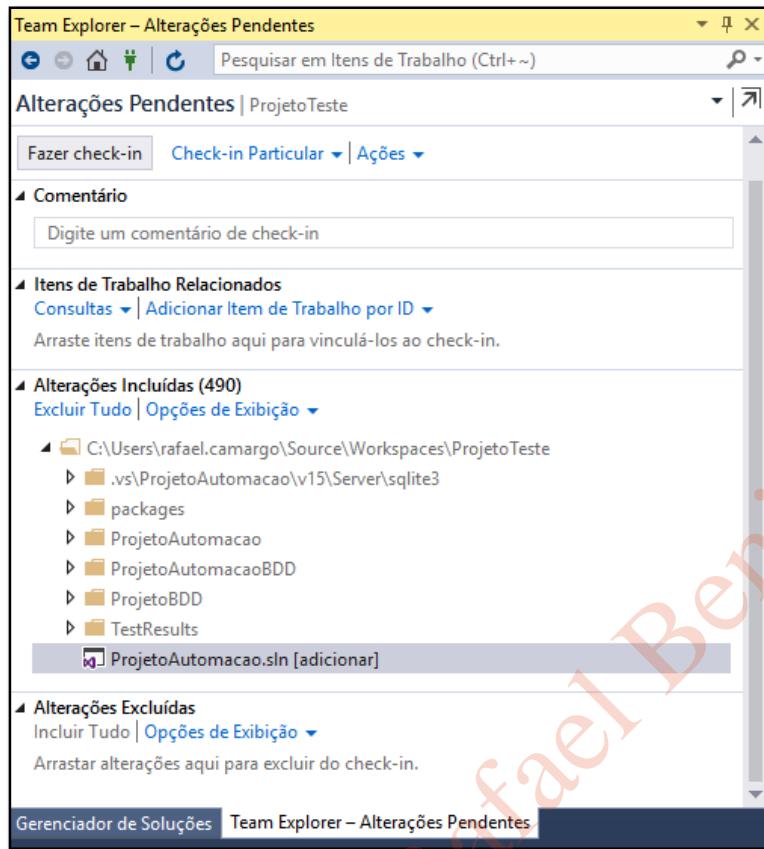
Projeto

Portal da Web | Painel de Tarefas

Meu trabalho ... Alterações Pendentes ...

Itens de Trabalho ... Builds

Esta é a tela que veremos sempre antes de subir alguma alteração no código :



Neste menu temos o seguinte:

- **Comentário** – Mensagem com um resumo das alterações que estão sendo incluídas
- **Itens de Trabalho Relacionados** – Podemos vincular estes commits à itens de trabalho criados no Azure DevOps, por exemplo associar uma alteração à um requisito, ou à um card do boards.
- **Alterações Incluídas** – Aqui são os itens que serão incluídos do checkin. Sempre verifique as alterações antes de subir o código.
- **Alterações Excluídas** – Aqui estarão os itens que serão mantidos localmente do seu computador, mas não entrarão neste checkin.

Quando incluímos as pastas no passo anterior, foram incluídas algumas pastas das quais não devem ser versionadas como:

Packages – que são os pacotes e podem ser baixados após o download do código fonte

.vs – pasta que contém as configurações locais de cada computador, não deve ser incluída

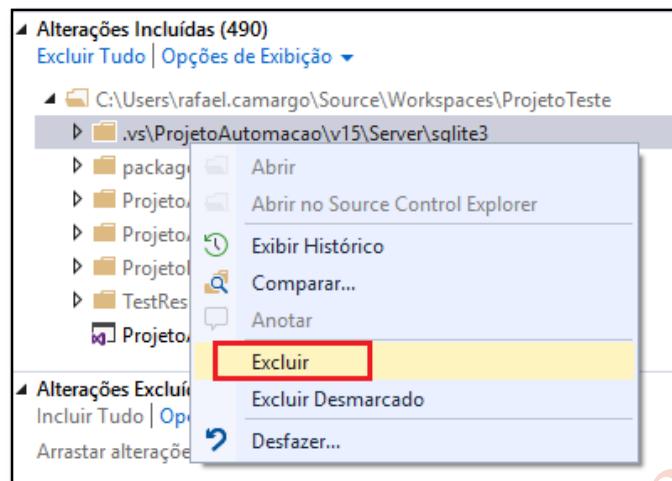
TestResults – são pastas de resultado geradas automaticamente, estes resultados não devem ser incluídos

Bin/Debug e **Bin/Obj** – pastas com o resultado da compilação do projeto, não incluir nenhuma

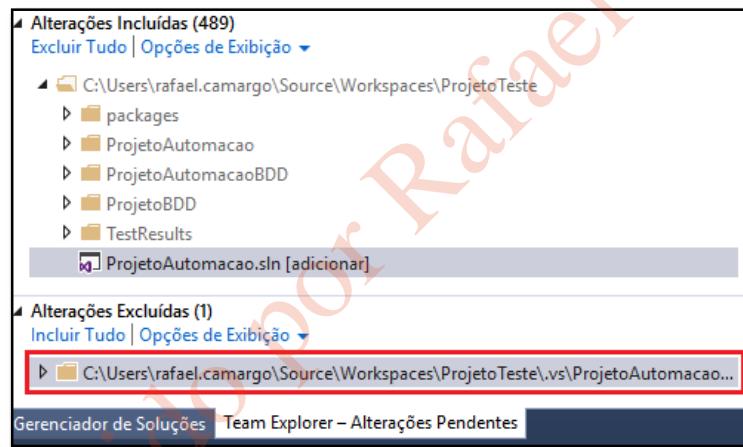
E as pastas que armazenam informações criadas em tempo de execução como a pasta de **Evidencias** que criamos para armazenar os screenshots.

Mantenha apenas os arquivos que são necessários para poder compilar o projeto, estes arquivos que são baixados e gerados após a compilação e execução não devem ser versionados.

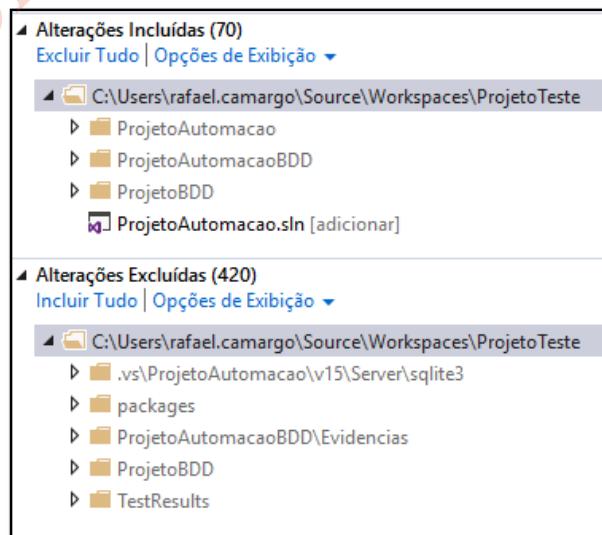
Caso note que algum destes esteja no item Alterações Incluídas faça o seguinte, clique com o botão direito no item que deseja remover do commit e selecione **Excluir**:



Após isto o item é movido para **Alterações Excluídas**.



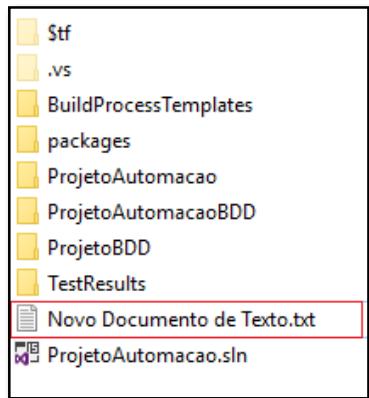
Repita o processo para os demais itens que não devem ser incluídos.



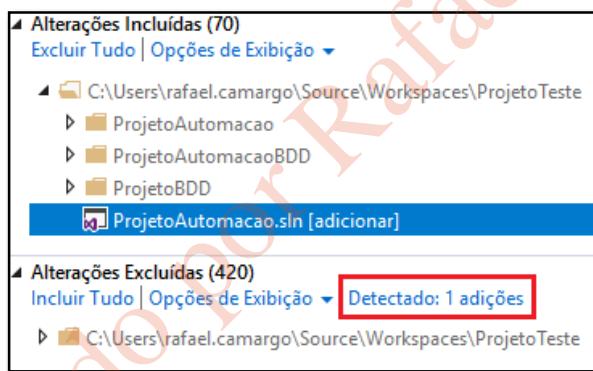
6.5.2 Criando o arquivo tfignore

Para que não seja necessário ficar excluindo estes arquivos toda vez que for subir uma nova alteração no código, vamos incluir um arquivo do tipo **.tfignore** (equivalente ao **.gitignore** em **Git**). Este arquivo mapeia as pastas ou arquivos que não devem ser rastreados dentro de seu projeto.

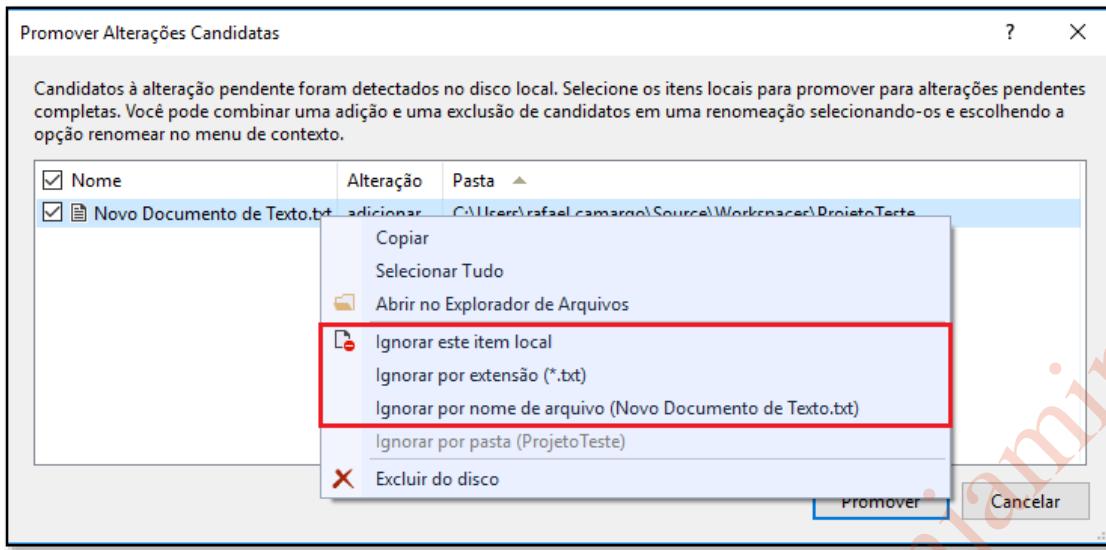
Abra a pasta de seu projeto pelo windows e inclua um arquivo txt em branco na raiz do projeto:



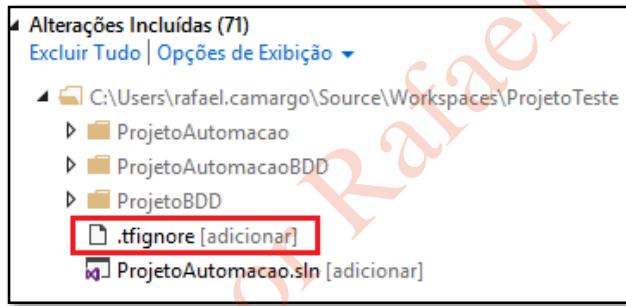
Agora de volta ao visual Studio, será exibido um novo link em **Alterações Excluídas**:



Será exibido um novo menu com as adições encontradas, clique com o botão direito acima do item e selecione umas das três opções para Ignorar o arquivo. Neste momento qualquer uma serve, isto será útil apenas para gerar o arquivo que precisamos. Eu irei selecionar a opção ignorar por extensão.



Depois que clicar o arquivo desaparecerá deste menu, feche este menu e veja em Alterações Incluídas que o arquivo **.tfignore** foi criado:



Dê dois cliques neste arquivo para abri-lo no editor do visual studio

```
C:\Users\rafael.cam...jetoTeste\.tfignore ➔ X Source Control Explorer
22 #
23 # # Exclua todos os arquivos terminados em .txt em Alpha\Beta e todas as suas subpastas.
24 # Alpha\Beta\*.txt
25 #
26 # # Exclua todos os arquivos terminados em .cpp nesta pasta apenas.
27 # \*.cpp
28 #
29 # # Exclua todos os arquivos terminados em .cpp nesta pasta e todas as subpastas.
30 # *.cpp
31 #
32 # # If "Contoso" é uma pasta, então, Contoso e todos os seus filhos são excluídos.
33 # # Se é uma pasta, então, apenas "Contoso" nesta pasta será excluído.
34 # \Contoso
35 #
36 # # Se Help.exe for excluído por um arquivo .tfignore maior ou pela lista global de exclusões da Coleção de Projeto
37 # #, então este padrão reinclui
38 # # nesta pasta apenas.
39 # !\Help.exe
40 #
41 ######
42 #
43 *.txt
44
```

O **#** representa um comentário neste arquivo, então até a linha 41 este arquivo não possui nenhuma ação, este cabeçalho descreve apenas o resumo geral de como ele funciona. Leia esta explicação dentro dele ☺

Abaixo disso, foi gerado um ***.txt**, significando que todos os arquivos com extensão .txt serão ignorados pelo controle de versão em seu projeto, isso foi gerado quando selecionei a opção ‘Ignorar por extensão (.txt)’ no último menu. Como não é nossa intenção ignorar arquivos txt pode apagar esta linha, a intenção foi apenas para gerar o arquivo tfignore mesmo. Agora abaixo da linha 41 inclua o seguinte:

```
#Custom Folders
*/Evidencias/*

## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.suo
*.user
*.userosscache
*.sln.docstates

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
x86/
bld/
[Bb]in/
[Oo]bj/
[Ll]og/

# Visual Studio cache/options directory
.vs/
# Uncomment if you have tasks that create the project's static files in wwwroot
#wwwroot/

# MSTest test Results
[Tt]est[Rr]esult*/
[Bb]uild[Ll]og.*

# NUNIT
*.VisualState.xml
TestResult.xml
```

Após incluir estas linhas salve o arquivo.

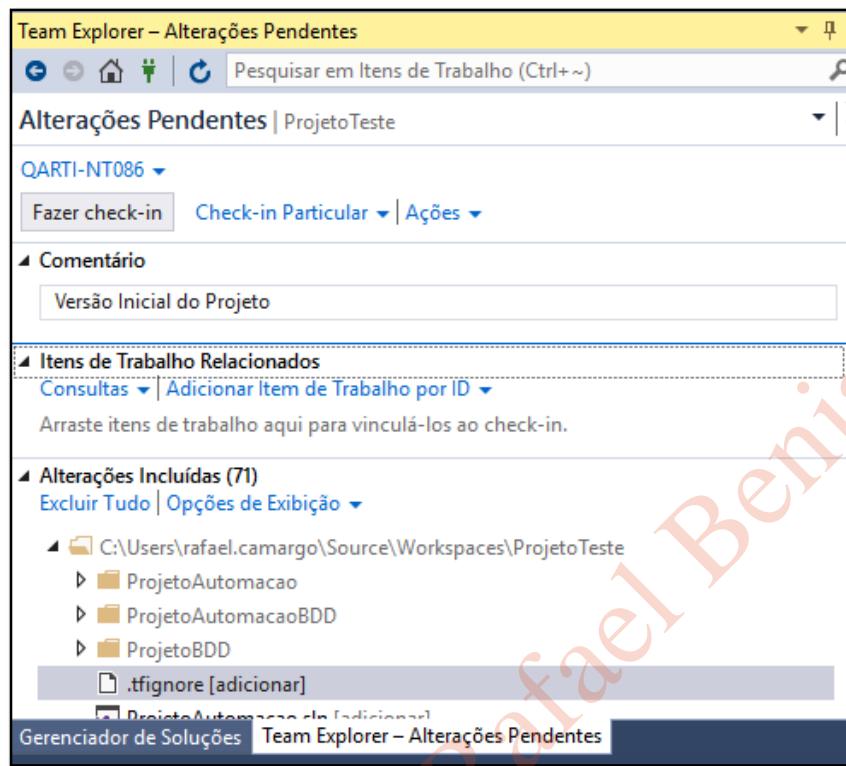
Seguindo as regras especificadas nos comentários, estas linhas irão ignorar os principais arquivos e pastas que serão utilizados nos projetos, este exemplo serve para maioria dos projetos. Também podemos customizar de acordo com nosso projeto, logo no início inclui uma opção para ignorar a pasta Evidencias e todo o conteúdo dentro dela em qualquer projeto.

Referência: <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/add-files-server?view=azure-devops#customize-which-files-are-ignored-by-version-control>

6.5.3 Fazendo o check-in das alterações

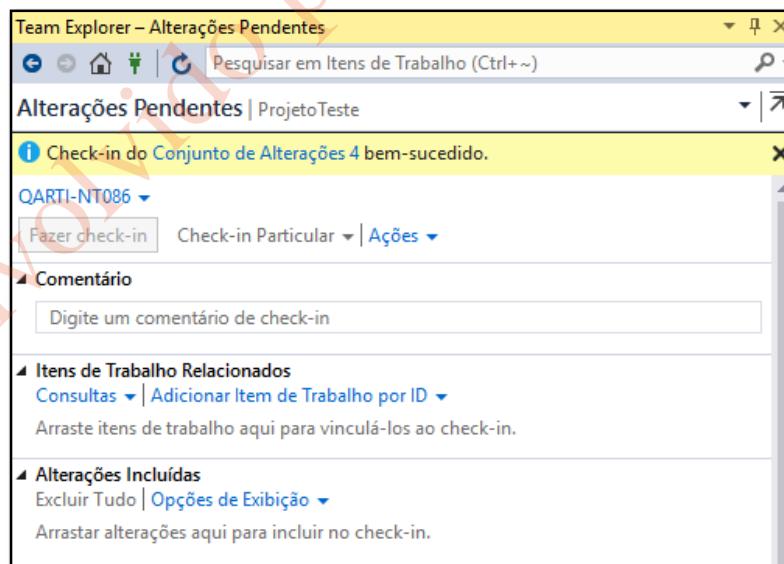
Após revisar o conteúdo que irá subir dentro do menu **Alterações Pendentes**, vamos efetuar o checkin e salvar o código dentro do repositório de projeto.

Feito isso, inclua um comentário descrevendo no que consiste essas alterações, neste caso estamos subindo a primeira versão do projeto no repositório.

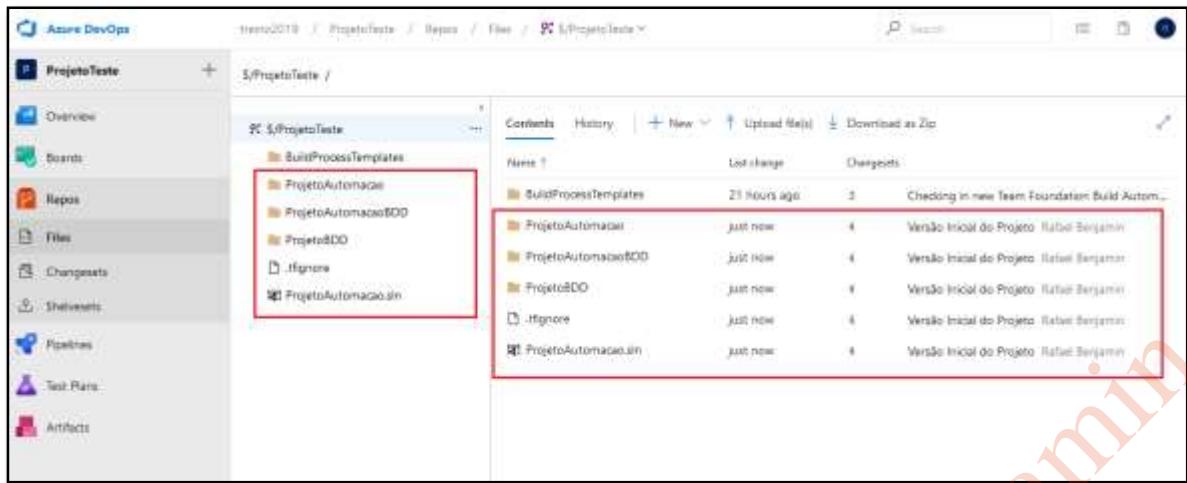


E clique em **Fazer Check-In**.

Então deve ser exibido a mensagem de sucesso.



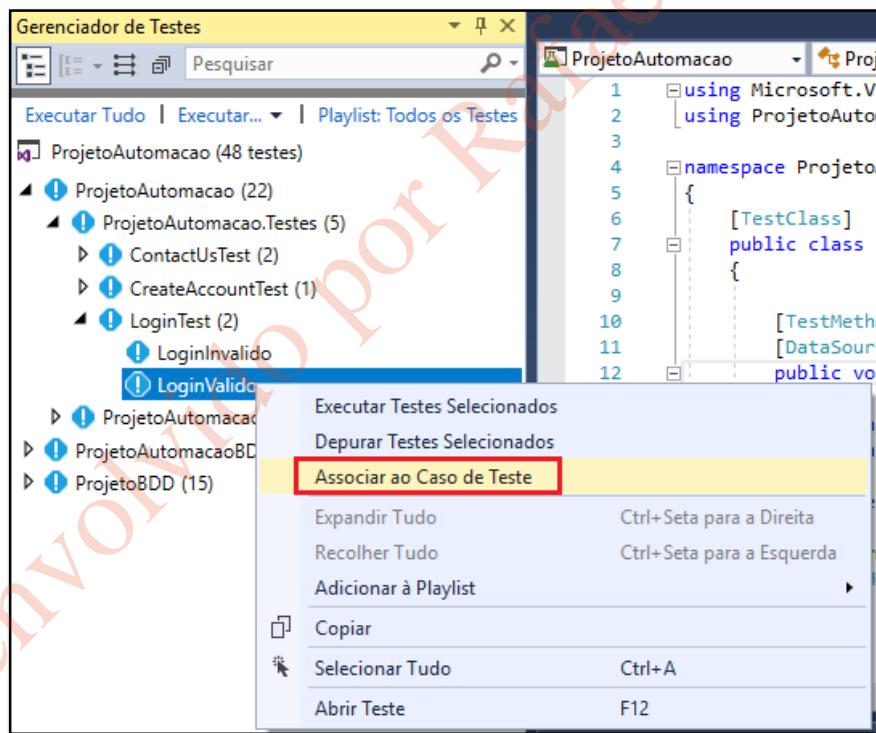
Ao acessar a página de repositório o projeto deve estar disponível:



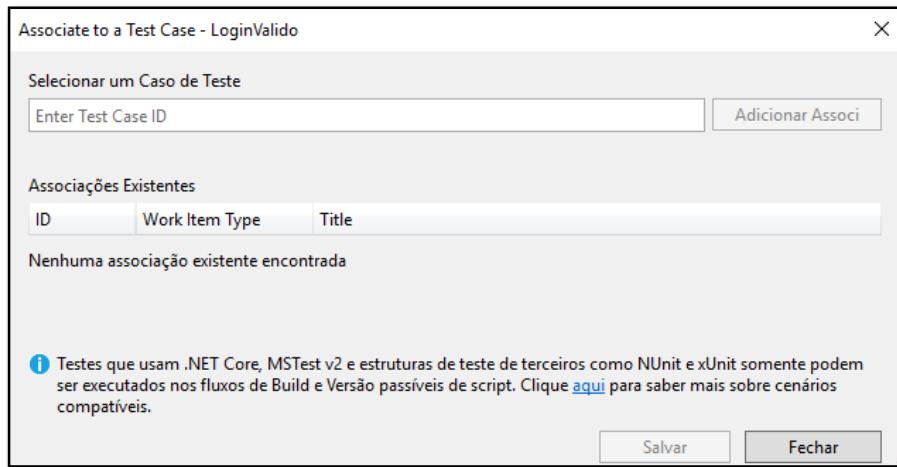
6.6 Associando o script a um caso teste

Já temos os casos de teste criados no Test Plan, e agora temos a automação salva no repositório. Voltando ao visual Studio clique com o botão direito no teste que deseja associar e selecione **Associar ao Caso de Teste**.

Caso seja solicitado um login, insira o email ou usuário do projeto que você está trabalhando.



Pode ser um teste de qualquer projeto, e tanto faz se foi feito usando o cucumber ou não. A associação apenas vincula o método de teste ao script.



Será exibido o menu para associação de casos de teste, e precisaremos do ID do caso de teste. Então acesse o **Test Plan** e veja qual é o **ID** do cenário correspondente a este script.

Test Plans > Regressão Loja

+ □ □ □ □

Regressão Loja

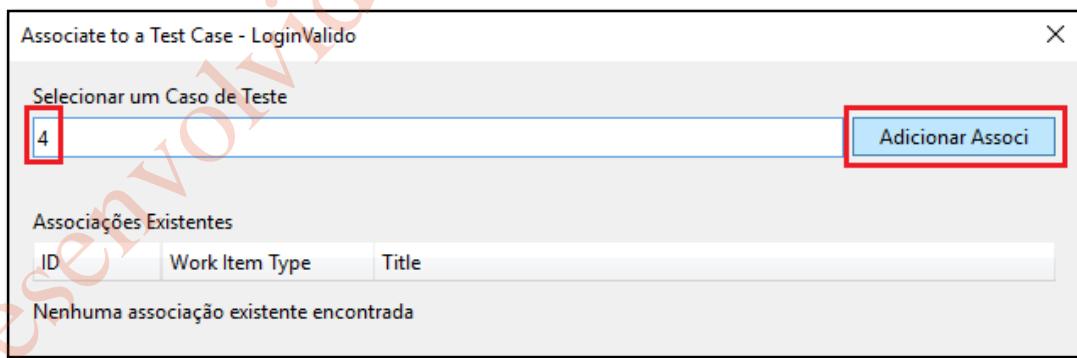
Geral (5)

Test suite: Geral (Suite ID: 3)

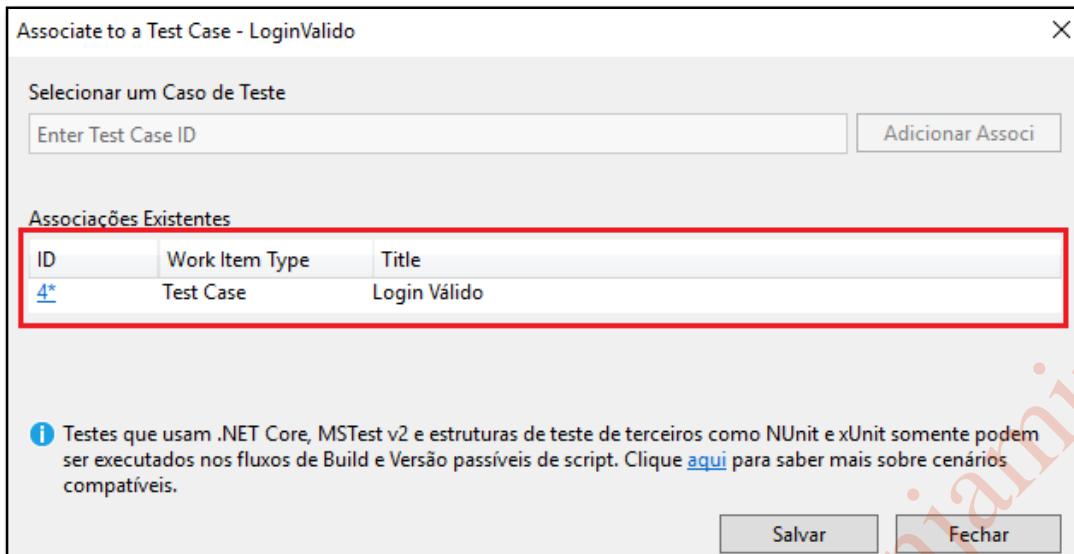
Tests Charts

Outcome	Order	ID ↑	Title
Active	1	4	Login Válido
Active	2	5	Login Inválido
Active	3	6	Fale Conosco Válido

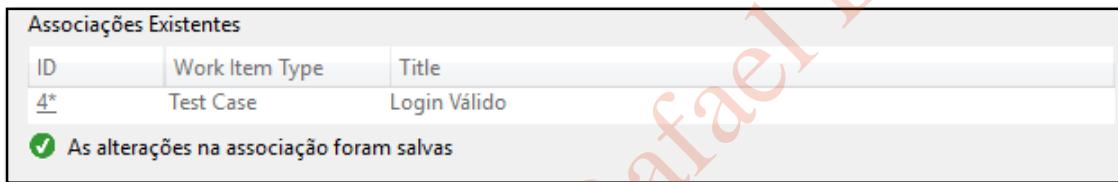
Neste caso o ID é 4, inclua esse ID no menu de associação e clique em **Adicionar Associação**



Após clicar em Adicionar Associação, se o caso de teste for encontrado ele será listado:



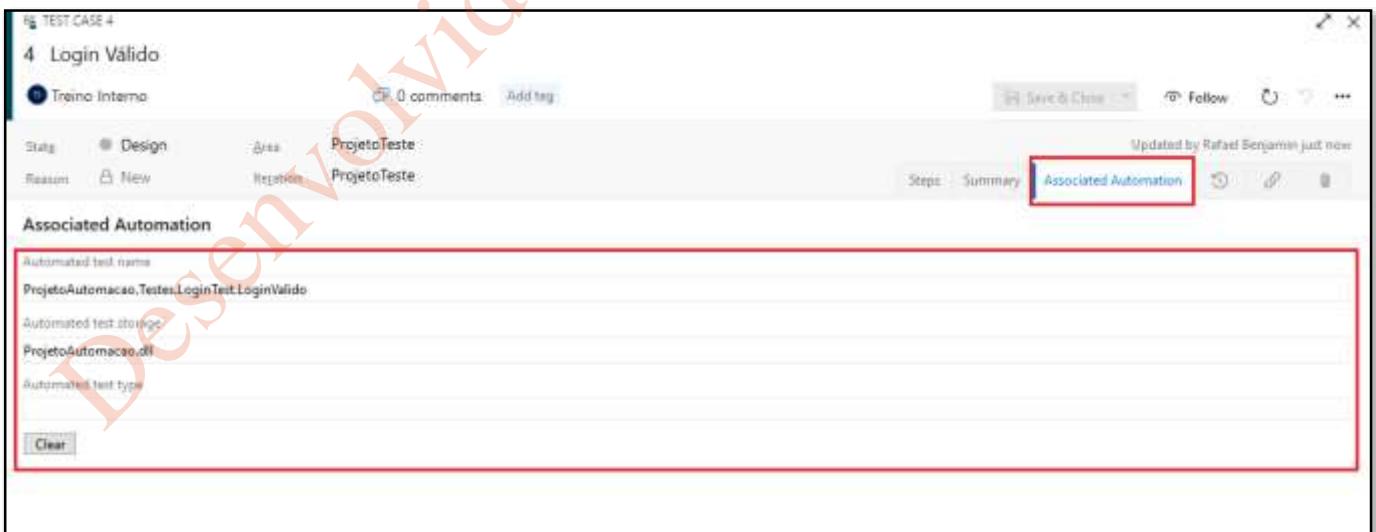
Se estiver ok, clique em Salvar.



A mensagem de confirmação é exibida.

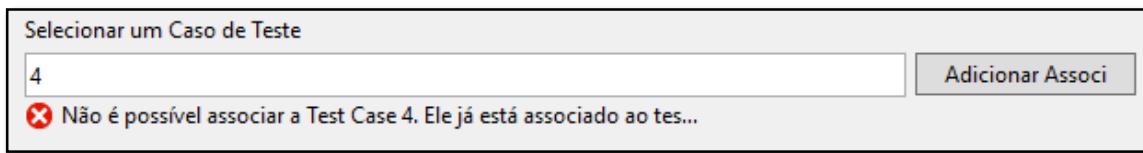
Acesse o **Test Plan** novamente e abra o caso de teste que foi associado. Na guia **Associated Automation** agora é exibido o detalhe, qual método de teste foi associado e em qual DLL este se encontra.

Aqui também é possível limpar esta associação e remover o vinculo.



Um caso de teste pode ter apenas um script associado.

Mas um script de teste, pode ser associado a vários casos de teste.



Para todos os scripts que tenha em seu projeto é necessário repetir estes passos.

Referência: <https://docs.microsoft.com/pt-br/azure/devops/test/associate-automated-test-with-test-case?view=azure-devops>

6.7 Configurando o Agent de Execução

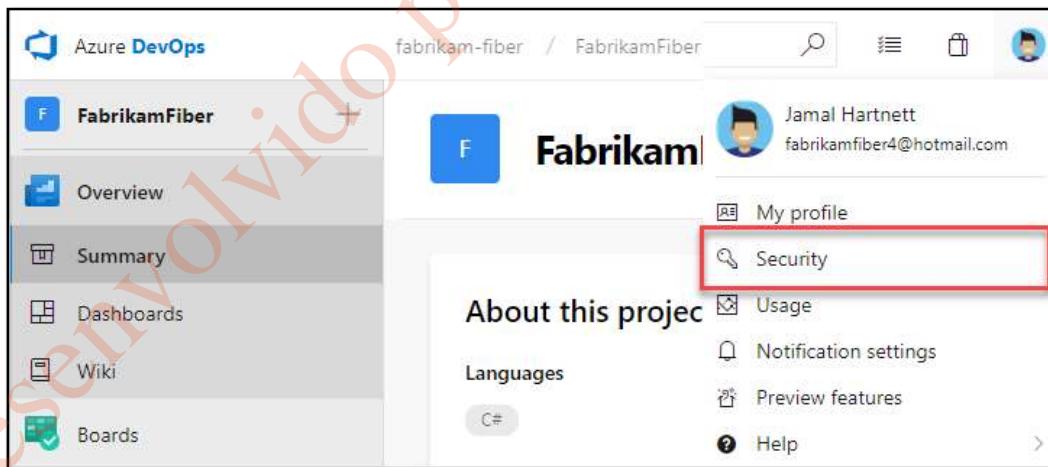
Temos nosso projeto no repositório e os casos de teste já associados, precisamos fazer com que sejam executados em algum lugar. A idéia é disparar a execução dos testes e estes sejam executados em alguma máquina externa, ao final da execução os resultados são registrados no Azure DevOps automaticamente.

A configuração que será feita a seguir geralmente não é necessária por nossa parte, isto é solicitado à área responsável que forneça um agent para realizar o build do projeto e para realizar a execução dos testes automatizados. Mas como precisamos simular esta máquina remota vamos criar um agent em nossa própria máquina.

6.7.1 Gerando um Token de acesso

Primeiro vamos obter um um **Personal Access Token (PAT)** que será utilizado para o agent ter acesso ao projeto.

Acesse a página do projeto, clique no ícone do seu perfil no canto superior direito e clique em **Security**.



Clique em **New Token**

User settings > Personal access tokens

General Notifications Usage Security

Personal access tokens

Alternate credentials SSH public keys

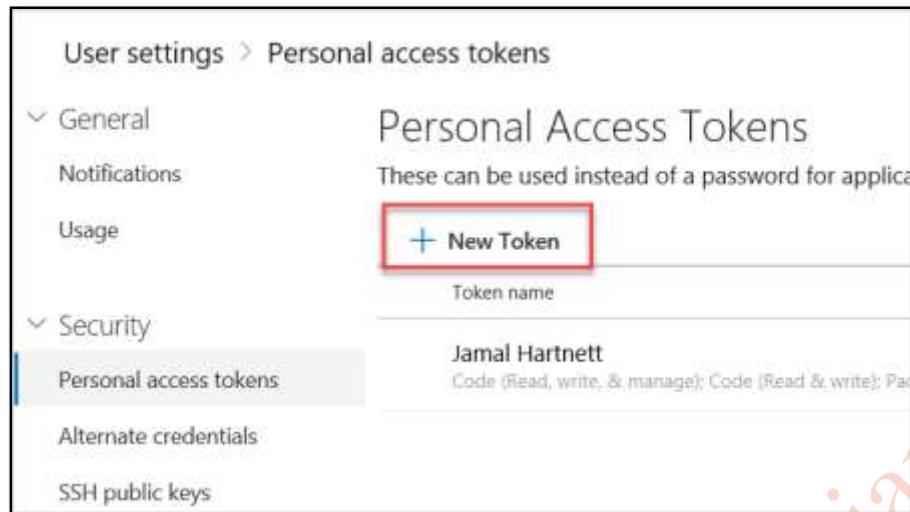
Personal Access Tokens

These can be used instead of a password for applica

+ New Token

Token name: Jamal Hartnett

Code (Read, write, & manage); Code (Read & write); Pac



Nesta tela, é possível definir as permissões e tempo de validade do token. Como é apenas um treinamento mantenha em Full Access.

Create a new personal access token

Name: nt086

Organization: treino2019

Expiration (UTC): 90 days | Thu Sep 26 2019

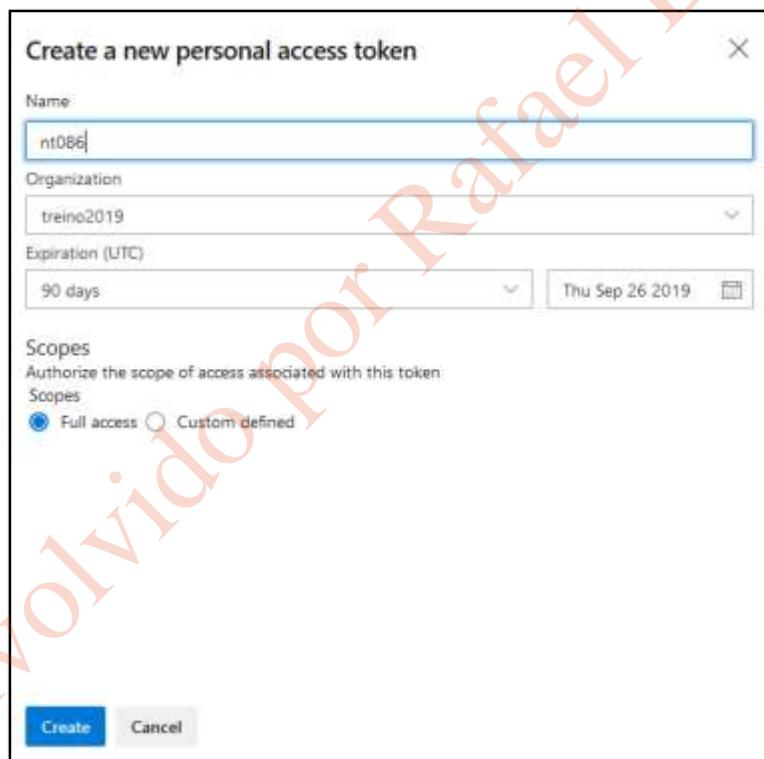
Scopes:

Authorize the scope of access associated with this token:

Scopes:

Full access Custom defined

Create **Cancel**

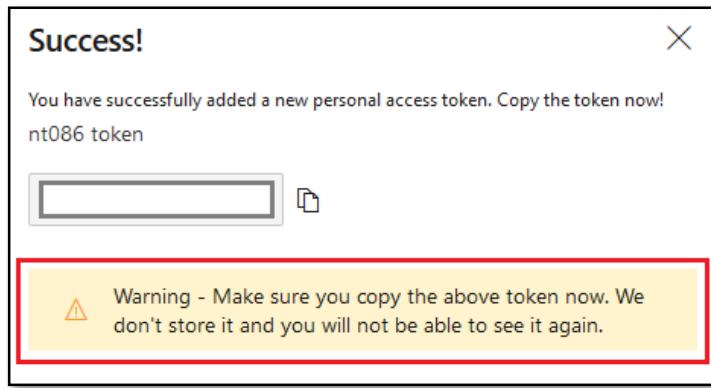


Ao clicar em Create o token será gerado

Atenção: Copie este token e salve-o com você.

Após fechar a tela seguinte não será mais possível obter este token.

Caso necessite terá que gerar outro novo.



6.7.2 Download do agent

O Azure DevOps disponibiliza alguns agents padrão, mas estes servem mais para builds, no nosso caso de execução de testes com interface precisaremos configurar um específico para isto.

Após obter o token, volte a página da sua organização clicando no logo **Azure DevOps** do canto superior esquerdo e clique em **Organization Settings**



Dentro de **Organization Settings**, clique em **Agent Pools** e clique em **Download agent**

Azure DevOps

treino2019 / Organization Settings / Agent pools

New agent pool...

All agent pools

- Default
- Hosted
- Hosted macOS
- Hosted macOS High Sierra
- Hosted Ubuntu 1604
- Hosted VS2017
- Hosted Windows 2019 with ...
- Hosted Windows Container

Agents Roles Details Settings Maintenance history

No agents are registered or you do not have permission to view the agents.

Agents for pool Default

Download agent

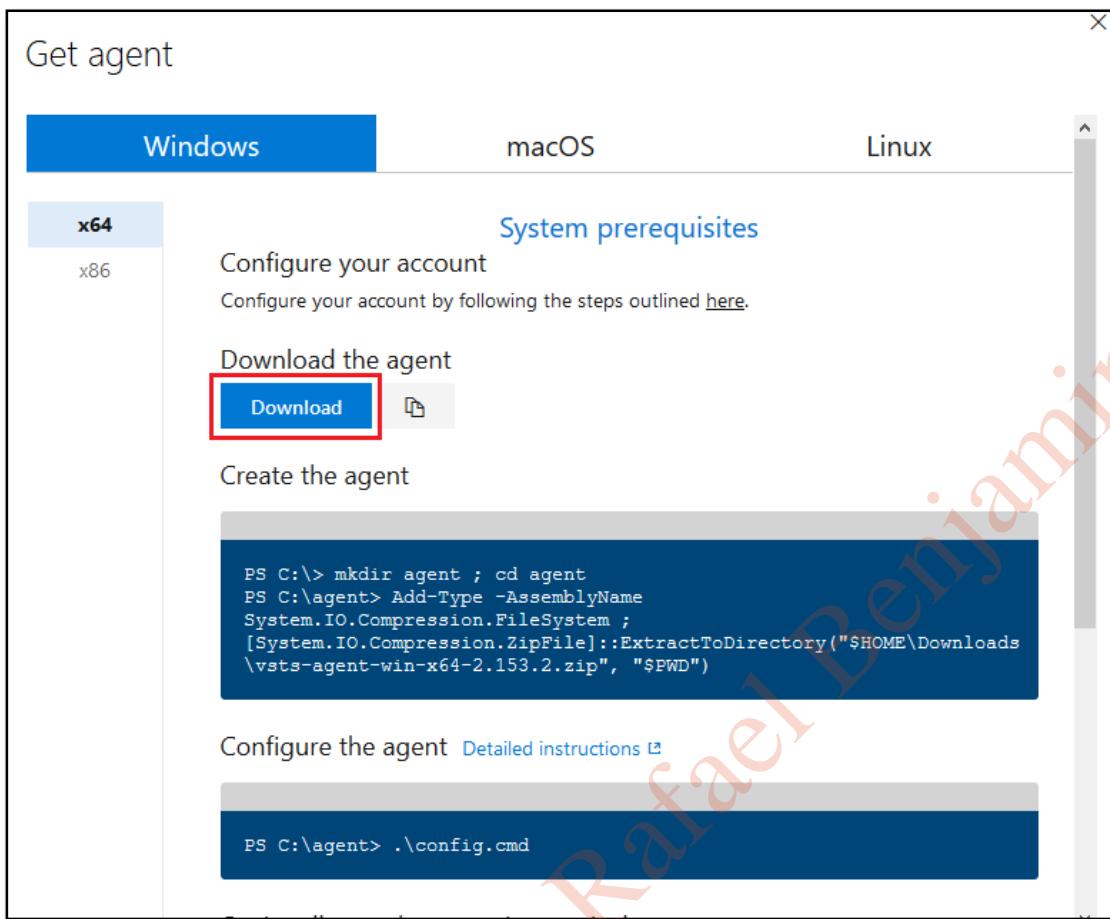
Agents

...
...

Agents
Billing
Auditing
Global notifications
Usage
Extensions
Azure Active Directory
Security
Policies
Permissions
Boards
Process
Pipelines
Agent pools
Deployment pools
Parallel jobs

Será exibida uma tela com instruções e agents para os três sistemas operacionais.

Selecione o correspondente a sua máquina, e clique em **Download**.



Para configurar o agent basta seguir as instruções logo abaixo do botão de download. Neste caso com Windows:

Abra o **Windows Power Shell**. Navegue para pasta aonde deseja criar o agent.

No meu caso irei navegar para a pasta `C:\Dev` da minha máquina, mas pode ser qualquer pasta do seu computador.

```
Windows PowerShell  
Copyright (C) Microsoft Corporation. Todos os direitos reservados.  
PS C:\Users\rafael.camargo> cd C:\dev  
PS C:\dev>
```

O primeiro comando é usado para criar uma pasta chamada agent e navegar para dentro dela:

```
mkdir agent ; cd agent
```

```

PS C:\Users\rafael.camargo> cd C:\dev
PS C:\dev> mkdir agent ; cd agent

        Diretório: C:\dev

Mode          LastWriteTime      Length Name
----          -----          ----- 
d----  28/06/2019     17:31           agent

PS C:\dev\agent> -

```

O segundo irá extrair o arquivo .zip que foi baixado para dentro da pasta atual , pasta agent que acabamos de criar. Mas atenção, é assumido que o arquivo se encontra dentro da sua pasta Downloads, se caso você tenha mudado a pasta padrão de downloads do seu navegador será necessário corrigir na parte em vermelho. E caso você esteja baixando uma versão futura do arquivo também se atente a isto no final.

```
Add-Type -AssemblyName System.IO.Compression.FileSystem ;
[System.IO.Compression.ZipFile]::ExtractToDirectory("$HOME\Downloads\vsts-agent-win-x64-2.153.2.zip", "$PWD")
```

O comando leva um tempo até ser processado, quando finalizar a linha de comando voltará a ficar disponível

```

PS C:\Users\rafael.camargo> cd C:\dev
PS C:\dev> mkdir agent ; cd agent

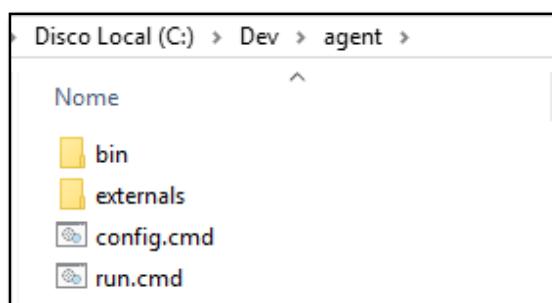
        Diretório: C:\dev

Mode          LastWriteTime      Length Name
----          -----          ----- 
d----  28/06/2019     17:31           agent

PS C:\dev\agent> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.ZipFile]::ExtractToDirectory("$HOME\Downloads\vsts-agent-win-x64-2.153.2.zip", "$PWD")
PS C:\dev\agent>

```

Feito isso os arquivos estarão disponíveis em:



Neste momento talvez você esteja se perguntando, não seria mais fácil simplesmente criar uma pasta agent manualmente e extrair o conteúdo para dentro pasta diretamente por uma ferramenta (winrar por exemplo) ?

Sim ☺

Mas quero que comecem a ver esse tipo de situação para se acostumar com estes tipos de comando.

Seguindo.

Agora vamos configurar este agent. Ainda pelo Windows Power Shell (agora realmente precisa ser via linha de comando) e dentro desta pasta com o conteúdo descompactado digite:

```
.\config.cmd
```

Será solicitado a url de seu projeto, copie a **url de seu projeto** neste formato: <https://dev.azure.com/myorganization>

Para o tipo de autenticação, digite apenas **Enter** (para usar tipo **PAT**)

Em seguida cole o token que foi criado e aperte Enter.

Agora nas demais configurações:

Enter agent pool(press enter for default) – Digite enter

Enter agent name(press enter for nome-da-maquina) – Digite enter (Aqui pode digitar algum nome para ele se quiser)

Enter work folder(press enter for _work) – Digite enter

Enter run agent as service? (Y/N) (press enter for N) > Digite enter (Não configure como serviço, precisa ser interativo por conta dos testes de UI)

Enter configure autologin and run agent on startup (Y/N) (press enter for N) – Digite enter

```
PS C:\dev\agent> .\config.cmd
>> Connect:
Enter server URL > https://dev.azure.com/treino2019/
Enter authentication type (press enter for PAT) >
Enter personal access token > *****
Connecting to server ...

>> Register Agent:
Enter agent pool (press enter for default) >
Enter agent name (press enter for QARTI-NT086) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2019-06-28 20:43:45Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) > N
Enter configure autologon and run agent on startup? (Y/N) (press enter for N) > N
PS C:\dev\agent> ■
```

E finalmente para ativar o agent digite:

```
.\run.cmd
```

```
PS C:\dev\agent> .\run.cmd
Scanning for tool capabilities.
Connecting to the server.
2019-06-28 20:55:15Z: Listening for Jobs
```

Nas próximas vezes que queria ligar este agent, basta digitar este último comando para liga-lo. Para encerra-lo digite **CTRL + C** no Power Shell, e digite S para encerrar seu processo.

```
PS C:\dev\agent> .\run.cmd
Scanning for tool capabilities.
Connecting to the server.
2019-06-28 20:55:15Z: Listening for Jobs
Exiting...
Deseja finalizar o arquivo em lotes (S/N)? s
PS C:\dev\agent> ■
```

Ao acessar a página de Agent pools novamente, podemos ver nosso agent, seu status atual e se está com algum job em andamento. Por padrão nosso agent está no pool **Default**.

The screenshot shows the 'Agent pools' page in the Azure DevOps interface. On the left, there's a sidebar with 'New agent pool...', a search bar, and a list of 'All agent pools' including 'Default', 'Hosted', 'Hosted macOS', 'Hosted macOS High Sierra', 'Hosted Ubuntu 1604', 'Hosted VS2017', 'Hosted Windows 2019 with ...', and 'Hosted Windows Container'. The 'Default' pool is selected. The main area is titled 'Agents for pool Default' with a 'Download agent' button. It has tabs for 'Agents', 'Roles', 'Details', 'Settings', and 'Maintenance history'. Under 'Agents', there's a table with columns: Enabled, Name, State, Current status, Requests, and Capabilities. One agent, 'QARTI-NT086', is listed as Enabled, Name, Online, Idle. The 'Requests' column shows 'No jobs have been run for this agent.'

Referências:

Download e configuração do agent: <https://docs.microsoft.com/pt-br/azure/devops/pipelines/agents/v2-windows?view=azure-devops>

Agent Interativo VS Agent de Serviço: <https://docs.microsoft.com/pt-br/azure/devops/pipelines/agents?view=azure-devops#interactive-or-service>

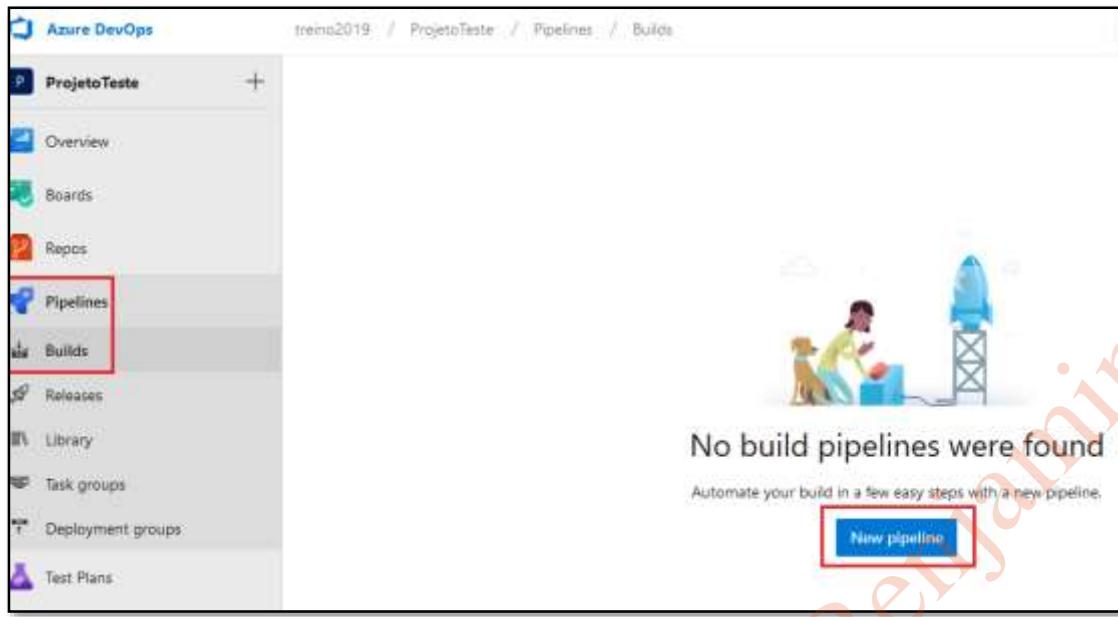
6.8 Configurando o Build

Podemos compilar nosso projeto através do próprio Azure DevOps utilizando o agent criado.

Vamos criar uma nova pipeline para realizar a compilação do projeto.

Acesse o projeto novamente, clique em **Pipelines**, e automaticamente direcionados para o submenu **Builds**

Dentro de **Builds** clique em **New Pipeline**



No primeiro passo vamos apontar de onde vem o código que queremos compilar. Selecione a última opção **Team Foundation Version Control**

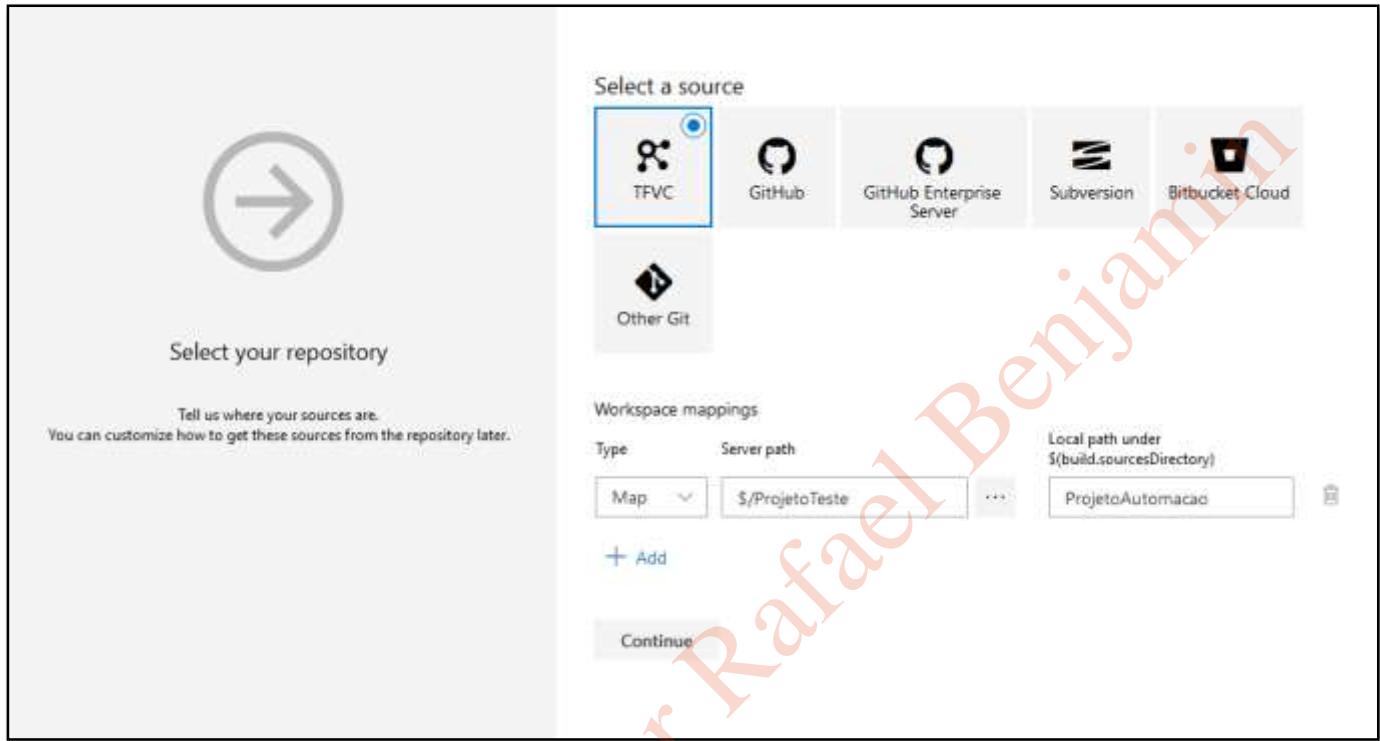
A screenshot of the 'Where is your code?' selection screen in Azure DevOps. The top navigation bar has tabs 'Connect', 'Select', 'Configure', and 'Review', with 'Select' being the active tab. Below the tabs, the text 'New pipeline' is displayed. The main heading 'Where is your code?' is bolded. A list of code sources follows:

- Azure Repos Git YAML
Free private Git repositories, pull requests, and code search
- Bitbucket Cloud YAML
Hosted by Atlassian
- GitHub YAML
Home to the world's largest community of developers
- GitHub Enterprise Server YAML
The self-hosted version of GitHub Enterprise
- Other Git
Any generic Git repository
- Subversion
Centralized version control by Apache
- Team Foundation Version Control** YAML
Centralized version control repositories

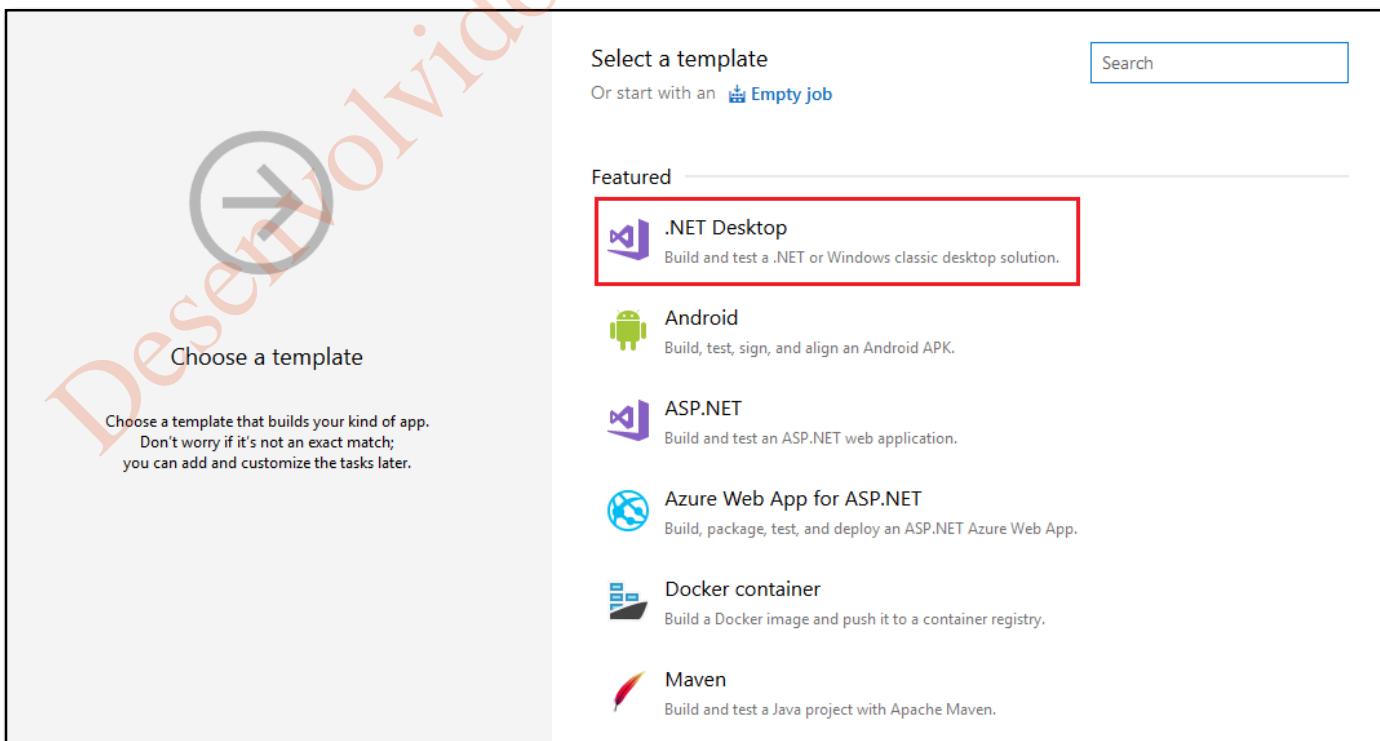
The 'Team Foundation Version Control' option is highlighted with a red box.

No próximo passo, temos que mapear em qual pasta do repositório nossa solução se encontra. Como ficou na raiz podemos manter da forma como está em **Server Path**. Em **Local Path**, é a pasta que será criada no agent para o que foi mapeado de Server Path, deixei como **ProjetoAutomacao**.

Clique em **Continue**.



Agora na seleção de template, serão listados vários tipos de builds que são suportados pelo Azure DevOps, selecione a opção **.Net Desktop** que trará um template próximo ao que iremos precisar.



Agora o menu exibirá as opções de acordo com o que for selecionado no menu da esquerda.

No menu a esquerda clique na primeira opção **Pipeline**.

The screenshot shows the Azure DevOps Pipeline editor interface. At the top, there's a navigation bar with links for Tasks, Variables, Triggers, Options, Retention, History, Save & queue, Discard, Summary, Queue, and a three-dot menu. Below the navigation bar, the left sidebar has a 'Pipeline' section highlighted with a red box. The main area displays a pipeline configuration with the following steps:

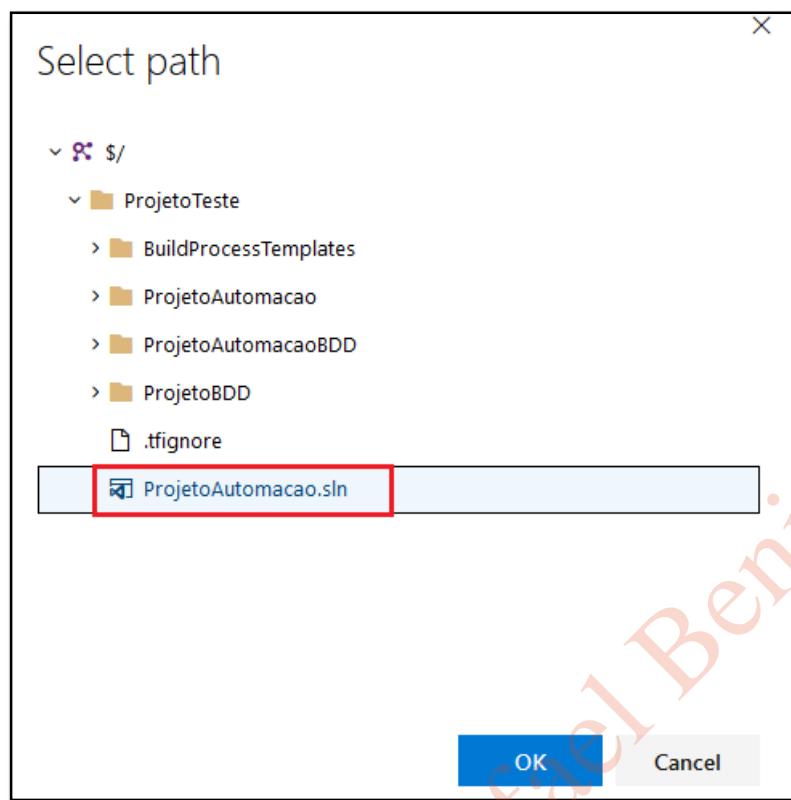
- Get sources: ProjetoTeste, S/ProjetoTeste
- Agent job 1: Run on agent
- + (Add step)
- Use NuGet 4.4.1: NuGet tool installer
- NuGet restore: NuGet
- Build solution ***.sln: Visual Studio build
- VsTest - testAssemblies: Visual Studio Test
- Publish symbols path: Index sources and publish symbols

On the right side, there's a detailed configuration panel for the 'Pipeline' step, which is currently selected. The panel includes fields for Name (ProjetoTeste - Build), Agent pool (Default), Parameters (Solution: ***.sln), and a 'Unlink all' button.

Em **Name**, sugiro que por hora deixe como **nome do projeto – Build**.

Em Agent pool, altere para o **Default**, que é onde se encontra nosso agent de execução.

Em **Parameters**, podemos configurar valores que serão usados nas demais tarefas deste pipeline, por padrão a **Solution** é exibida. Clique em ... e selecione o arquivo solution do seu repositório:



Para o menu **Get sources** não será necessário alterar nada, já foi configurado no segundo passo.

E Para o menu **Run on Agent** também não é necessário alterar nada.

The screenshot shows the Azure DevOps Pipeline editor interface. On the left, a list of tasks is visible, including 'Get sources', 'Agent job 1', 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution \$/ProjetoTeste/ProjetoAutomac...', 'VsTest - testAssemblies', and 'Publish symbols path'. The 'Agent job 1' task is currently selected and highlighted with a red box. On the right, the configuration for this agent job is shown. It includes a 'Display name' field set to 'Agent job 1', an 'Agent selection' dropdown set to '<inherit from pipeline>', and a 'Demands' table with three entries: 'msbuild' (Condition: exists), 'visualstudio' (Condition: exists), and 'vstest' (Condition: exists).

Na primeira task **NuGet tool Instaler**, também não é necessário alterar nada. Este passo só ira adicionar o instalador do nuget na máquina.

O próximo, **NuGet restore** é o mesmo comando que usamos quando baixamos um projeto, ele irá realizar o download de todos os pacotes que estão apontados na solução.

O passo **Visual Studio Build**, é o responsável por compilar o projeto.

No item **Visual Studio Version**, certifique-se de selecionar a versão na qual o projeto foi criado, neste caso a 2017.

E no item **MsBuild Arguments**, podem ser opções para a compilação. Inclua o seguinte comando:

```
/p:OutDir="$(build.artifactstagingdirectory)"
```

Este comando define que após a compilação, os arquivos gerados serão incluídos em um diretório padrão representado pela variável pré-definida `build.artifactstagingdirectory`.

Referência:

Linha de comando MsBuild: <https://docs.microsoft.com/pt-br/visualstudio/msbuild/msbuild-command-line-reference?view=vs-2019>

Variáveis Pré definidas do Azure DevOps: <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops&tabs=yaml>

The screenshot shows a build pipeline named 'Pipeline'. It includes a 'Get sources' task and an 'Agent job 1' which runs on an agent. Inside 'Agent job 1', there are several tasks: 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution \$/ProjetoTeste/ProjetoAutomacao.sln' (which is checked and has a blue checkmark icon), 'VsTest - testAssemblies' (disabled, indicated by a greyed-out icon), and 'Publish Artifact: drop'. To the right of the pipeline, the 'Visual Studio build' task is expanded. It has fields for 'Task version' (set to 1.*), 'Display name' (Build solution \$/ProjetoTeste/ProjetoAutomacao.sln), 'Solution' (\$/ProjetoTeste/ProjetoAutomacao.sln), 'Visual Studio Version' (set to Visual Studio 2017), and 'MSBuild Arguments' (/p:OutDir="\$(build.artifactstagingdirectory)". A large red box highlights the 'Visual Studio Version' and 'MSBuild Arguments' fields.

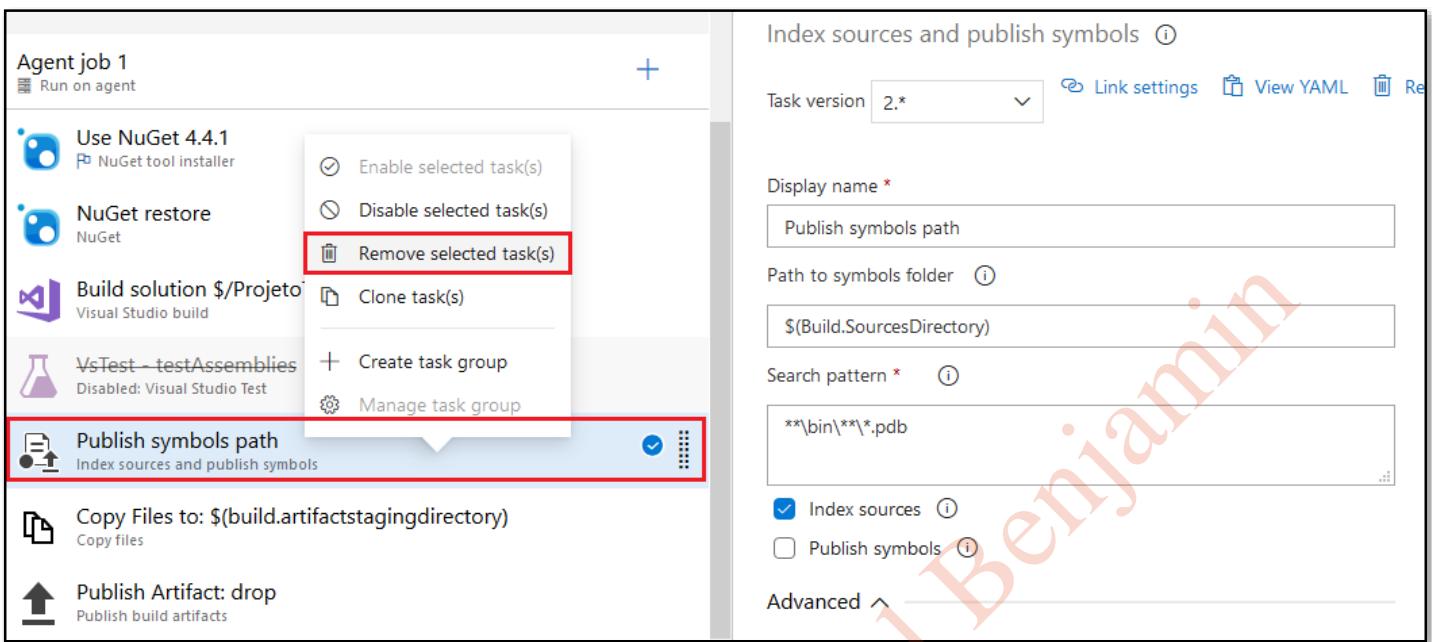
O próximo passo que é **Visual Studio Test**, é o que executa os testes contidos na solução. Mas por hora mantenha-o desabilitado, vamos ver como configurado na próxima seção em mais detalhes.

Para desabilitá-lo clique com o botão direito nele e selecione **Disable selected task(s)**.

The screenshot shows the same build pipeline as before, but with the 'VsTest - testAssemblies' task selected. A context menu is open over this task, with the 'Disable selected task(s)' option highlighted by a red box. The pipeline also shows other tasks: 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution \$/ProjetoTeste/ProjetoAutomacao.sln', 'VsTest - testAssemblies' (which is disabled and has a greyed-out icon), 'Publish symbols path', 'Copy Files to: \$(build.artifactstagingdirectory)', and 'Publish Artifact: drop'. To the right of the pipeline, the 'Visual Studio Test' task is expanded, showing its configuration options like 'Task version' (2.*), 'Display name' (VsTest - testAssemblies), 'Test selection' (with 'Select tests using' set to 'Test assemblies'), 'Test files' (with a pattern of **\\$(BuildConfiguration)*test*.dll !**\obj**), and 'Search folder'.

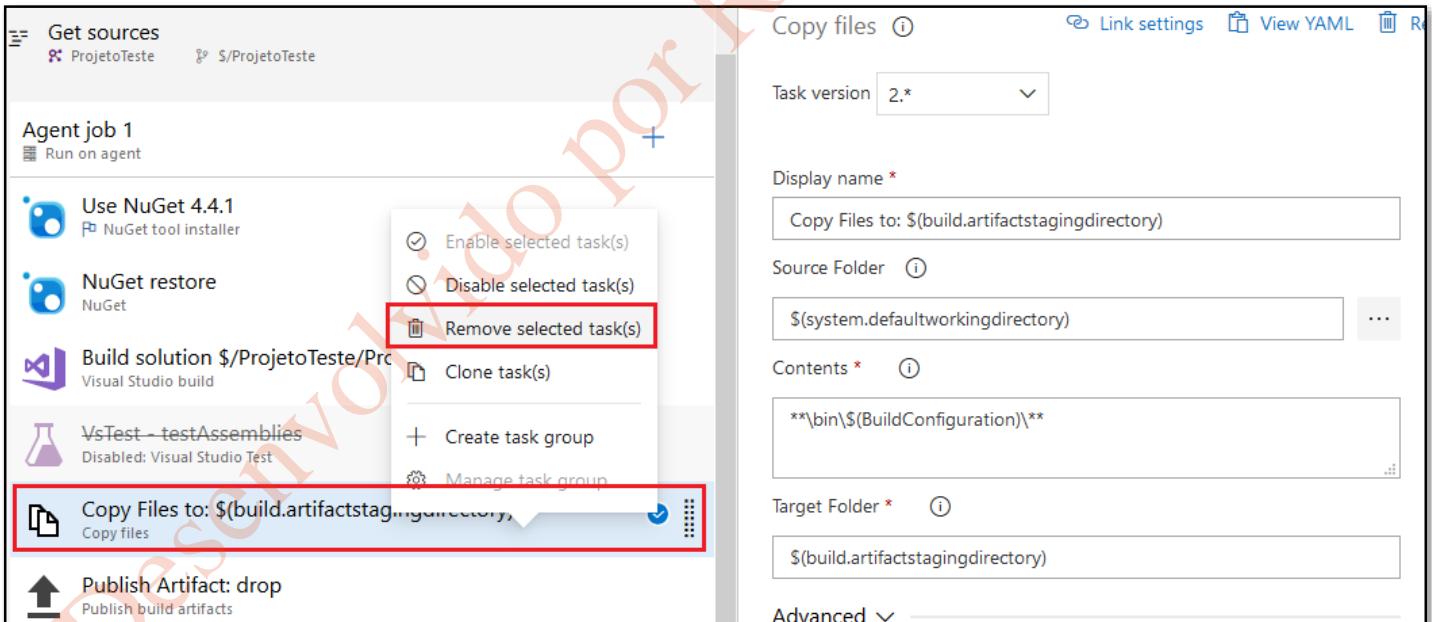
A task **Index sources and publish symbols** não será utilizada por nós.

Para excluí-la clique com o botão direito nele e selecione **Remove selected task(s)**.



A task **Copy files** também não será utilizada por nós.

Para excluí-la clique com o botão direito nele e selecione **Remove selected task(s)**.



E finalmente a task **Publish build artifacts**, que irá publicar os arquivos especificados dentro do servidor para ser utilizado na fase de **Release**. Por padrão o **Path to Publish**, que é quais arquivos devem ser publicado já vem com a variável *build.artifactstagingdirectory*, a mesma a qual estamos mandando a saída de nossa compilação.

Não é necessário alterar mais nada aqui.

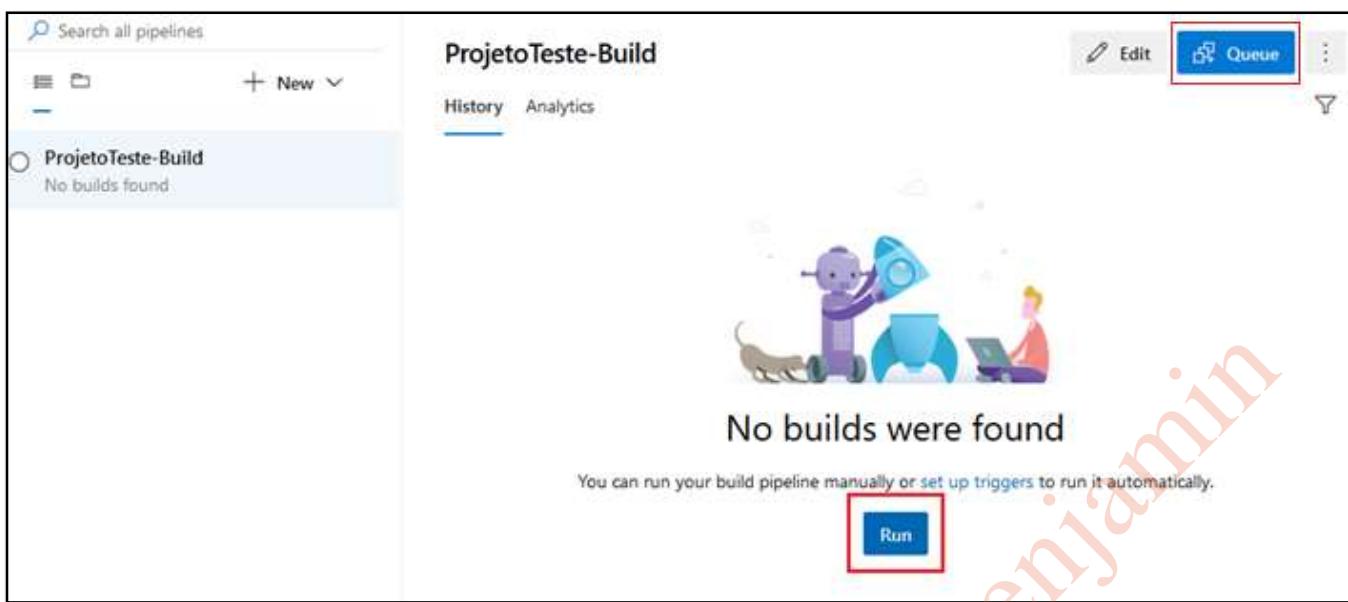
The screenshot shows the Azure Pipelines interface for configuring a pipeline. On the left, a list of tasks is visible: 'Get sources', 'Agent job 1' (which contains 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution \$/ProjetoTeste/ProjetoAutomaca...', and 'VsTest - testAssemblies'), and 'Publish Artifact: drop'. The 'Publish Artifact: drop' task is highlighted with a red box. On the right, the configuration details for this task are shown:

- Display name ***: Publish Artifact: drop
- Path to publish ***: \$(build.artifactstagingdirectory)
- Artifact name ***: drop
- Artifact publish location ***: Azure Pipelines
- Control Options**:
 - Enabled
 - Continue on error
- Timeout ***: 0

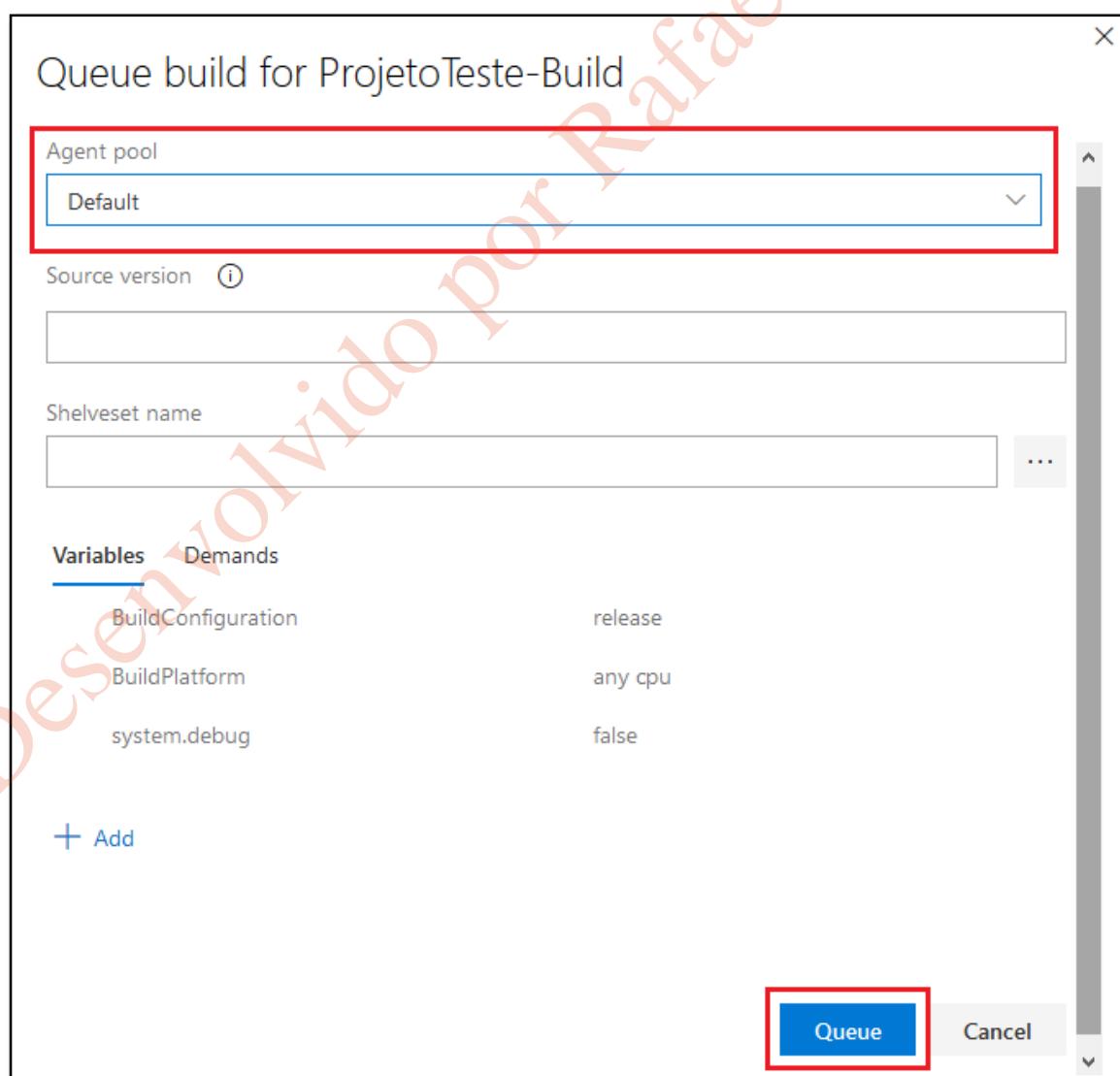
Concluída a configuração, Clique em **Save**

The screenshot shows the 'Save & queue' dropdown menu in the Azure Pipelines interface. The 'Save' option is highlighted with a red box. Other options in the menu include 'Save & queue' (disabled), 'Save as draft', and 'Link settings'.

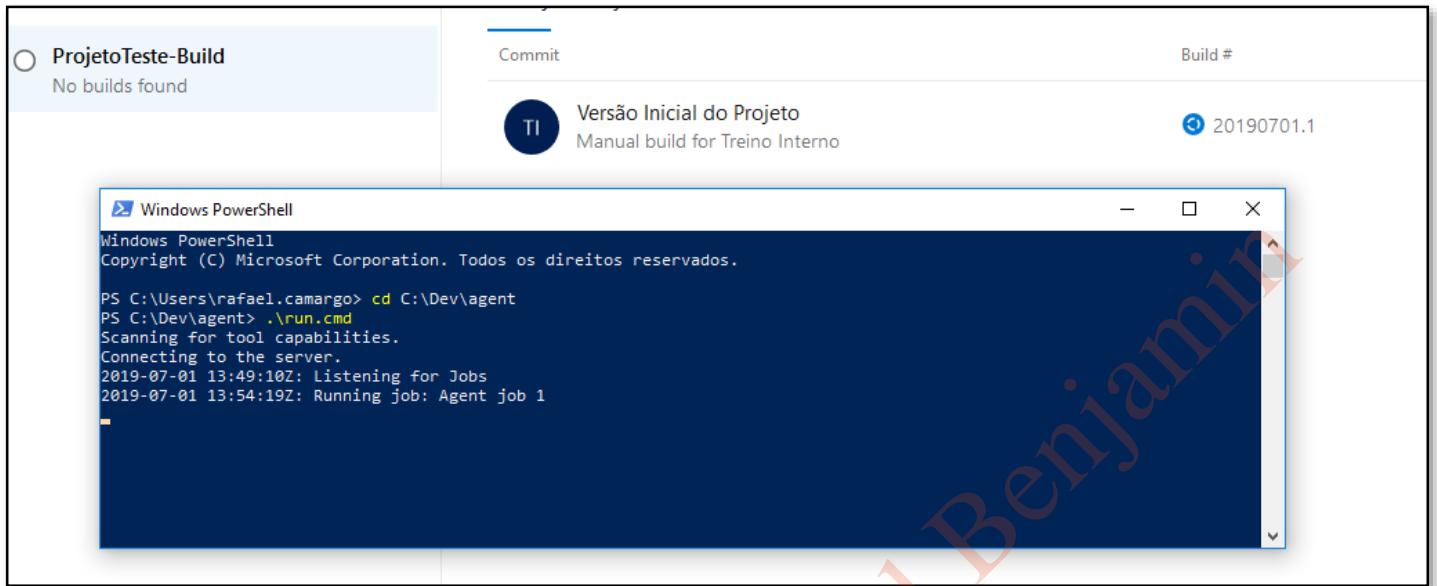
Retorne ao menu de **Builds**, e veja que agora a build é exibida mas ainda não teve nenhuma execução. Clique em **Queue**, para incluir uma build desta na fila.



Após isso, será exibido um menu com opções para execução, certifique-se de que o **Agent pool**, está apontado para o **Default**, que é onde se encontra nosso agent de execução. Feito isso clique em **Queue**.



O job com a build será então disparado para o agent, que no momento é a sua máquina. Como é apenas uma compilação não ficará nada visível além da mensagem de processamento pelo PowerShell do agent.



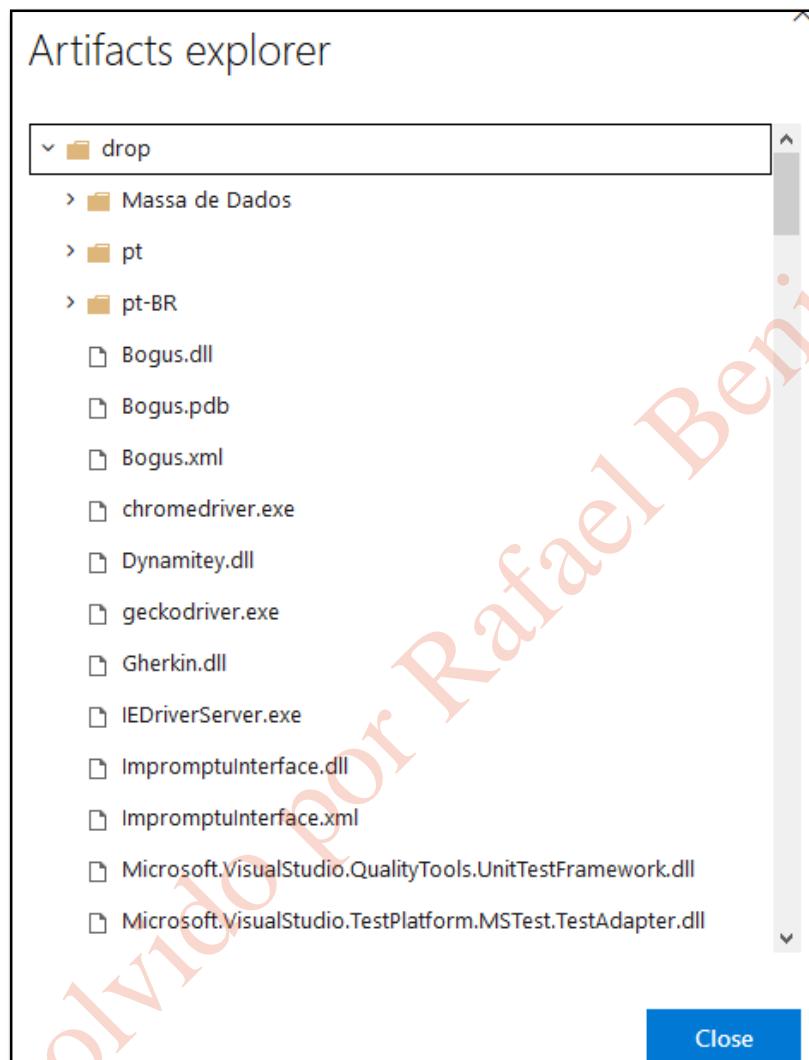
Após a conclusão, atualize a página do build e veja o seu resultado. Será gerada uma opção **Artifacts**, onde estará contido o resultado da compilação.

A screenshot of the build results page for build #20190701.4. At the top, there's a green checkmark icon and the build number. To the right are buttons for "Release" and "Artifacts", with "Artifacts" being highlighted by a red box. Below this, there are tabs for "Logs", "Summary", and "Tests", with "Logs" selected. The main area shows the "Agent job 1" details, including the start time (01/07/2019 11:34:07), pool ("Default"), and agent ("QARTI-NT086"). A list of build steps is shown, each with a green checkmark and the status "succeeded": "Prepare job", "Initialize job", "Checkout", "Use NuGet 4.4.1", "NuGet restore", "Build solution \$/ProjetoTeste/ProjetoAutomacao.sln", "Publish Artifact: drop", "Post-job: Checkout", and "Finalize Job". A large watermark reading "Reservado por Rafael Benjamim" is diagonally across the image.

Aqui na pasta **drop**, nome vem definido por padrão no item **Artifact Name** da task **Publish build artifacts** quando estavamos configurando o build.

E o conteúdo é que foi definido no step **Visual Studio Build**, quando incluímos comando `/p:OutDir="$(build.artifactstagingdirectory)"`.

Repare que não é incluído nada da solution ou do projeto que usamos durante o desenvolvimento, são apenas os arquivos compilados que geralmente estão dentro da pasta **/bin/debug**.



Fechando esta tela, podemos clicar em alguma das taks executadas e ver seu log. Útil em caso de falhas e erros.

```

1  ##[section]Starting: Checkout
2  =====
3  Task      : Get sources
4  Description : Get sources from a repository. Supports Git, TfsVC, and SVN repositories.
5  Version    : 1.0.0
6  Author     : Microsoft
7  Help       : [More Information](https://go.microsoft.com/fwlink/?LinkId=798199)
8  =====
9  Prepending Path environment variable with directory containing 'tf.exe'.
10 Setting environment variable TFVC_BUILDAGENT_POLICYPATH
11 Querying workspace information.
12 ##[command]tf vc workspace /new /location:local /permission:Public ws_1_8 /collection:https://dev.azure.com/treino2019/_/loginType:OAuth /loginType:OAuth /login:.,** /noprompt
13 ##[command]tf vc workfold /unmap /workspace:ws_1_8 $/ /collection:https://dev.azure.com/treino2019/_/loginType:OAuth /login:.,** /noprompt
14 ##[command]tf vc workfold /map /workspace:ws_1_8 $/ProjetoTeste C:\Dev\agent\_work\1\s\ProjetoAutomacao /collection:https://dev.azure.com/treino2019/_/loginType:OAuth /login:.,** /noprompt
15 ##[command]tf vc get /version:4 /recursive /overwrite C:\Dev\agent\_work\1\s /loginType:OAuth /login:.,** /noprompt
16 C:\Dev\agent\_work\1\s:
17 Obtendo ProjetoAutomacao
18
19 C:\Dev\agent\_work\1\s\ProjetoAutomacao:
20 Obtendo .tfignore
21 Obtendo BuildProcessTemplates
22 Obtendo ProjetoAutomacao
23 Obtendo ProjetoAutomacao.sln
24 Obtendo ProjetoAutomacaoBDD
25 Obtendo ProjetoBDD
26
27 C:\Dev\agent\_work\1\s\ProjetoAutomacao\BuildProcessTemplates:
28 Obtendo AzureContinuousDeployment.11.xaml
29 Obtendo DefaultTemplate.11.1.xaml
30 Obtendo UpgradeTemplate.xaml
31
32 C:\Dev\agent\_work\1\s\ProjetoAutomacao\ProjetoAutomacao:
33 Obtendo BaseTest.cs

```

6.8.1 Incluindo testes na fase de Build

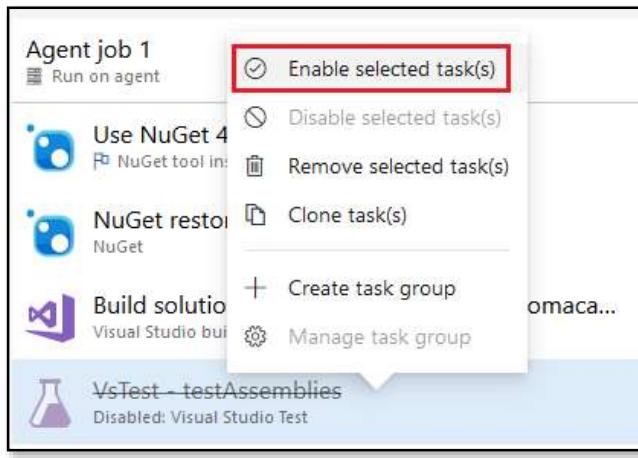
Temos a compilação de nosso projeto sendo realizada com sucesso, vamos incluir a task de testes agora nesta fase de build.

Nota: O ideal nesta fase de build é executar os **testes unitários** que são muito **mais rápidos**, mas vamos ver como executar os nossos testes funcionais aqui. Ainda veremos como executar nossos testes funcionais através dos casos de teste criados no Test Plan.

Volte ao menu de Build, e selecione a opção **Edit**



Ative a task de testes que havíamos desabilitado anteriormente, clique com o botão direito e selecione **Enable selected task(s)**.



Agora temos que definir uma forma de execução de testes pela opção **Select tests using**.

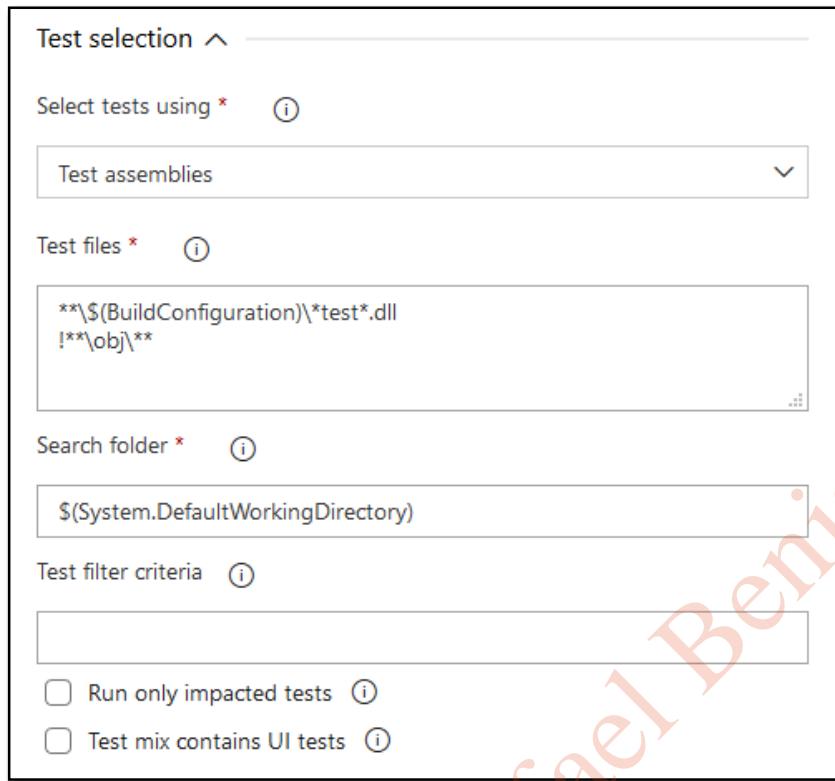
Dentro dela temos três opções:

- **Test assembly:** Use this option to specify one or more test assemblies that contain your tests. You can optionally specify a filter criteria to select only specific tests.
- **Test plan:** Use this option to run tests from your test plan that have an automated test method associated with it.
- **Test run:** Use this option when you are setting up an environment to run tests from the Test hub. This option should not be used when running tests in a continuous integration / continuous deployment (CI/CD) pipeline.

Os testes podem ser executados em três maneiras diferentes. Na opção Select Tests using, é possível selecionar as seguintes opções:

- **Test assembly:** Use esta opção para especificar um ou mais test assemblies que contém seus testes. Você pode opcionalmente especificar um filtro para selecionar apenas testes específicos. Indicado para execução dos testes unitários. - ***Não iremos usar esta forma, porém esta é a indicada quando se utiliza testes unitários***
- **Test plan:** Use esta opção para executar testes de um test plan que contém casos de teste automatizado associados. - ***Iremos utilizar esta forma aqui pelo build***
- **Test run:** Use esta opção quando estiver configurando um ambiente de testes para execução pelo Test Hub. Esta opção não deve ser utilizada quando executando uma pipeline de (CI/CD) integração contínua / deploy contínuo. ***Não iremos utilizar agora, mas sera explicada mais detalhadamente nas próximas seções.***

Usando a opção **Test Assemblies**, não iremos utilizar mas irei deixar uma breve explicação sobre sua configuração:

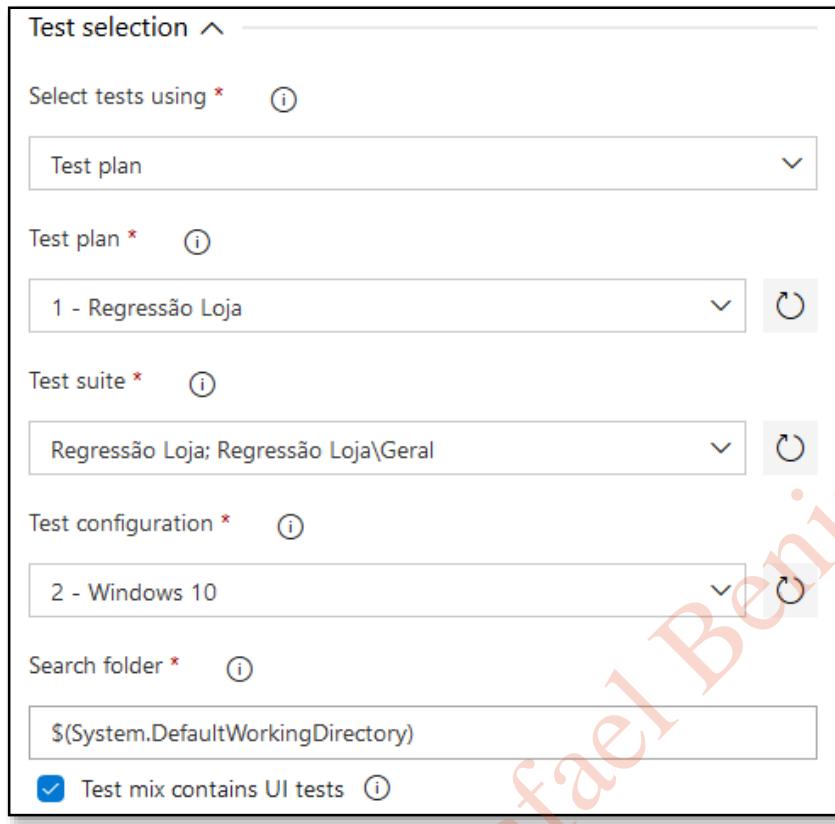


Test assemblies: indica quais arquivos devem ser buscados.

Search folder: indica em qual pasta do projeto esta busca deve ser realizada.

Test filter criteria: Pode aplicar filtros para selecionar apenas testes específicos. Caso não seja preenchida executa todos os testes encontrados nos arquivos dll especificados na busca. Ex: Name=LoginValido

6.8.1.1 Usando a opção Test Plan:



Test plan: indica qual test plan será selecionado para execução dos testes. Apenas executa os testes que possuam associação à testes automatizados, e estes estejam dentro do projeto de testes em que será realizado o build.

Test suite: Selecionar a suite de testes vinculada ao test plan que possua os casos de testes automatizados correspondentes ao projeto que está sendo compilado.

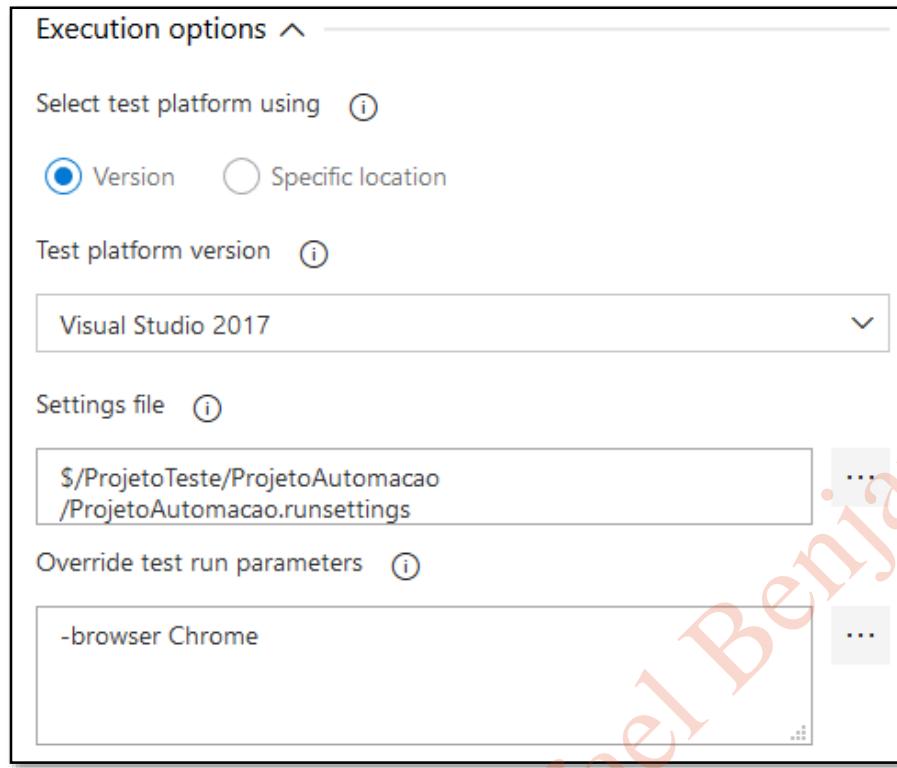
Test Configuration: Caso exista uma configuração criada para os casos de teste dentro do plan, esta pode ser selecionada. Esta não afeta os testes automatizados, serve apenas para seleção dos testes.

Search folder: indica em qual pasta do projeto sera realizada a busca pelas dlls.

Test mix contains UI tests: Marcar para indicar que testes interativos serão executados.

6.8.1.2 Especificando um arquivo de configurações de execução .runsettings

Caso o projeto possua um arquivo de configurações de execução (*.runsettings), este pode ser especificado na seção Execution Options.



Test platform version: Especifica a versão do visual studio a ser utilizada. No caso a 2017.

Settings file: Indica qual o arquivo de configurações a ser utilizado. Selecionar o arquivo do projeto que sera executado.

Override test run parameters: Pode sobrescrever os parametros definidos dentro do arquivo de configuração. Os valores a serem sobreescritos dependem do projeto a ser executado. Os valores a serem sobreescritos também podem ser armazenados dentro de variáveis do build.

6.8.1.3 Variaveis do build

Através da guia Variables é possível configurar as variáveis a serem utilizadas. No caso apenas iremos sobreescriver os valores que estão dentro do arquivo de configuração utilizado. Caso estes valores não sejam sobreescritos, utilizam os valores configurados no arquivo do tipo .runsettings do projeto.

Por exemplo, o arquivo possui a configuração browser que define em qual browser a execução sera realizada.

Lembrando que neste caso estas configurações devem ser alteradas e salvas antes de cada execução.

The screenshot shows the 'Variables' tab in the Azure DevOps interface for a build pipeline named 'ProjetoTeste-Build'. The 'Pipeline variables' section lists several variables:

Name ↑	Value
BuildConfiguration	release
BuildPlatform	any cpu
system.collectionId	110eafc5-6980-42c8-827e-145a6ae4f293
system.debug	false
system.definitionId	1
system.teamProject	ProjetoTeste
varBrowser	Chrome

A red box highlights the row for 'varBrowser'. A blue '+' icon and the word 'Add' are located at the bottom left of the table.

Utilizando o valor da variável **browser** para sobrescrever o parâmetro browser:



Referência: <https://devblogs.microsoft.com/devops/supplying-run-time-parameters-to-tests/>

6.8.1.4 Execução da build com Testes

Após configurar a task **Visual Studio Test** dentro da pipeline de build, vamos executá-la novamente.

Clique em **Queue** para agendar uma nova build e aguarde a finalização.

Após a conclusão, clique na guia **Tests**.

Uma visão geral dos testes executados será exibido, como a suíte de testes que selecionei possui 5 testes, podemos ver os resultados aqui:

#20190702.3: Atualizando massa de dados

Manually run today at 12:13 by Treino Interno ProjetoTeste \$/ProjetoTeste 6

Logs Summary Tests

Summary

1 Run(s) Completed (0 Passed, 1 Failed) [5 unique failing tests in the last 14 days](#)

Total tests: 5 | **Passed:** 4 | **Failed:** 1 | **New:** 0 | **Existing:** 1

Pass percentage: 80% | **Run duration:** 1m 29s (647ms)

Bug Link Test run Column Options

Filter by test or run name: Test file Owner Outcome: Aborted (+1)

Test	Duration	Failing since	Failing build
TestRun_ProjetoTeste-Build_20190702.3 (1/5)	0:01:13.778	0:00:13.793	32m ago 20190702.1
>Login Válido			

Expandido o caso de teste com falha, é listado a mensagem de erro e a stack trace na guia Debug:

Logs Summary Tests

TestRun_ProjetoTeste-Build_20190702.3 > LoginValido (Data Row 0)

Failed 1h ago on QARTI-NT086 Duration 0:00:13.150
Failing build 20190702.1 Owner not available

Debug Work items Attachments History Bug Link

Test method ProjetoAutomacao.Testes.LoginTest.LoginValido threw exception:
OpenQA.Selenium.NoSuchElementException: Could not find element by: By.CssSelector: .account

Stack trace

```

    em SeleniumExtras.PageObjects.DefaultElementLocator.LocateElement(IEnumerable`1 bys)
    em SeleniumExtras.PageObjects.WebElementProxy.get_Element()
    em SeleniumExtras.PageObjects.WebElementProxy.Invoke(IMessage msg)
    em System.Runtime.Remoting.Proxies.RealProxy.PrivateInvoke(MessageData& msgData, Int32 type)
    em OpenQA.Selenium.IWebElement.get_Text()
    em ProjetoAutomacao.Testes.LoginTest.LoginValido() na C:\Dev\agent\_work\1\s\ProjetoAutomacao\ProjetoAutomacao\Testes>LoginTest
  
```

E clicando em Attachments é possível ver os anexos que incluímos, neste caso apenas a última evidência:

#20190702.3: Atualizando massa de dados

Manually run today at 12:13 by Treino Interno ProjetoTeste \$/ProjetoTeste 6

Logs Summary Tests

TestRun_ProjetoTeste-Build_20190702.3 > LoginValido (Data Row 0)

✗ Failed 2h ago on QARTI-NT086 Duration 0:00:13.150
Failing build 20190702.1 Owner not available

Debug Work items **Attachments** History Bug Link

>LoginValido_Imagen_Final_02_07_...
59K Added 1h ago

Mesma coisa para um cenário com status Passed.

#20190702.3: Atualizando massa de dados

Manually run today at 12:13 by Treino Interno ProjetoTeste \$/ProjetoTeste 6

Logs Summary Tests

TestRun_ProjetoTeste-Build_20190702.3 > Fale Conosco Válido

✓ Passed 2h ago on QARTI-NT086 Duration 0:00:13.780
Owner not available

Debug Work items **Attachments** History Bug Link

FaleConoscoValido_Imagen_Final_...
58K Added 1h ago

6.9 Criando uma release

Para que os testes sejam executados a partir do test plan, precisamos criar uma release que seja associada a este test plan.

Antes de prosseguir, caso a sua build possua a task de testes, desabilite-a. Os testes serão executados de outra forma.

The screenshot shows the Azure DevOps Pipeline editor for a project named 'ProjetoTeste-Build'. The pipeline consists of several tasks:

- Get sources (ProjetoTeste, \$/ProjetoTeste)
- Agent job 1 (Run on agent):
 - Use NuGet 4.4.1 (NuGet tool installer)
 - NuGet restore (NuGet)
 - Build solution \$/ProjetoTeste/ProjetoAutomaca... (Visual Studio build)
- VsTest - testPlan (Disabled: Visual Studio Test)
- Publish Artifact: drop (Publish build artifacts)

The 'VsTest - testPlan' task is highlighted with a red box. On the right, the configuration for this task is shown:

- Display name: VsTest - testPlan
- Test selection:
 - Select tests using: Test plan
 - Test plan: 1 - Regressão Loja
 - Test suite: Regressão Loja; Regressão Loja\Geral
 - Test configuration: 2 - Windows 10

Acesse o submenu **Releases** e clique em **New Pipeline**

The screenshot shows the Azure DevOps Releases page for the 'ProjetoTeste' project. The left sidebar menu is highlighted with a red box, showing the following options:

- Overview
- Boards
- Repos
- Pipelines
- Builds
- Releases** (highlighted with a red box)
- Library
- Task groups
- Deployment groups
- Test Plans
- Artifacts

The main area displays a message: "No release pipelines found". Below it, there is a call-to-action: "Automate your release process in a few easy steps with a new pipeline" and a blue "New pipeline" button, which is also highlighted with a red box.

Selecione o template **Run Automated tests from Test Manager**

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

Stages | + Add

Stage 1 Select a template

IIS website and SQL database partially online upgrade

IIS website deployment

Machine Learning Model

Run automated tests from Test Manager

Empty job

Start with an empty job and add your own steps.

Em **Artifacts**, clique em **Add**

No menu exibido teremos que selecionar a fonte com os arquivos para executar os testes, vamos manter a seleção da **Build**. No item **Source(build pipeline)** selecione o seu projeto de testes. Clique em **Add**

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

Stages | + Add

Stage 1 1 job, 1 task

Source type

Build

Azure Repos... GitHub TFVC

5 more artifact types

Project * ProjetoTeste

Source (build pipeline) * ProjetoTeste-Build

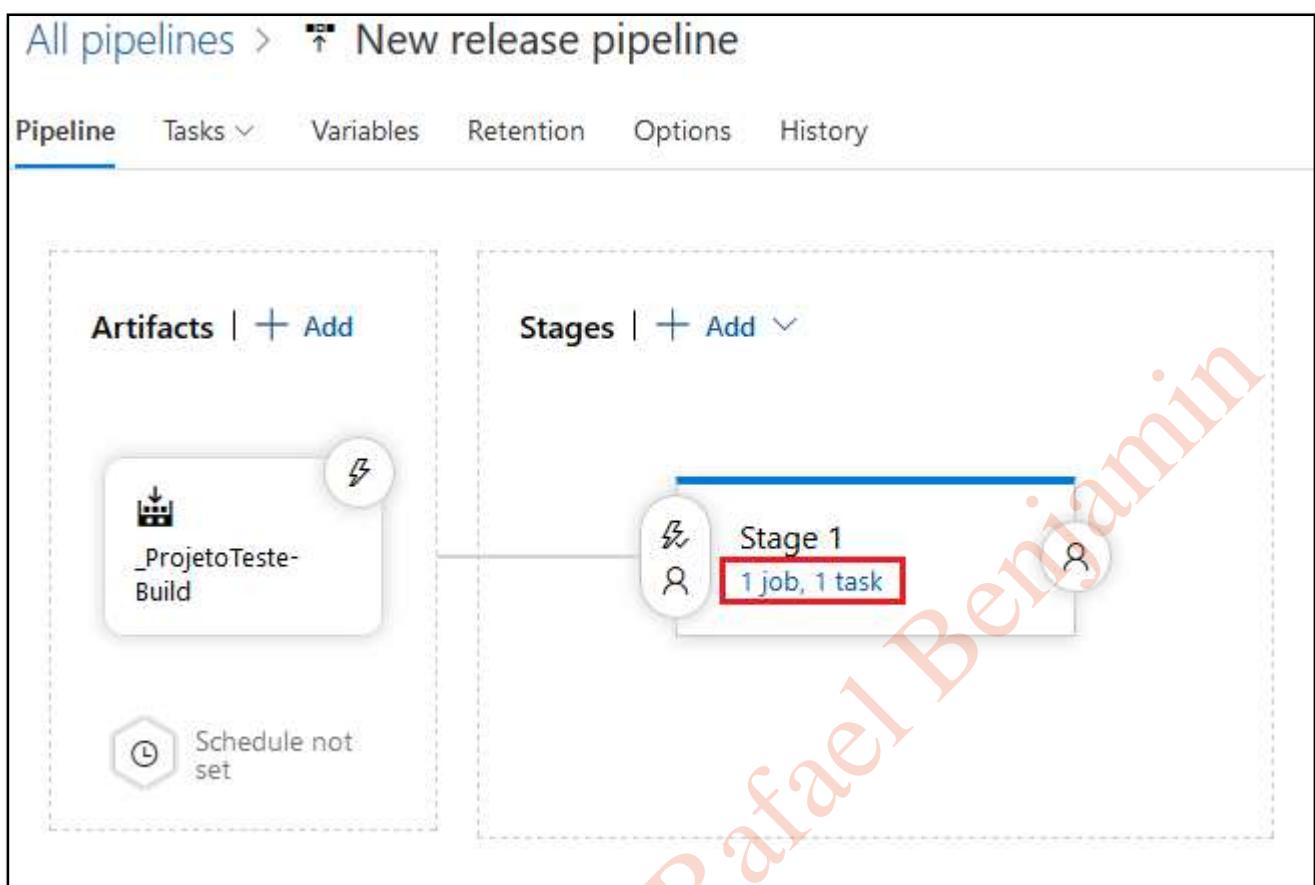
Default version * Latest

Source alias * _ProjetoTeste-Build

(i) No version is available for ProjetoTeste-Build or the latest version has no artifacts to publish. Please check the source pipeline.

Add

Depois de configurado o **Artifact**, clique dentro do link em **1job, 1task** dentro de Stages



Aqui será realizada uma configuração semelhante a que foi feita na task da build.

Clique em **Run on agent** à esquerda, e certifique-se de selecionar o **Agent pool** com o nosso agent criado para a execução dos testes automatizados.

Pipeline Tasks Variables Retention Options History

Stage 1 Deployment process

Run on agent

Test run for Test plans Visual Studio Test

Agent job

Display name *

Run on agent

Agent selection

Agent pool | Pool information | Manage

Default

Demands

Name	Condition	Value
vstest	exists	

+ Add

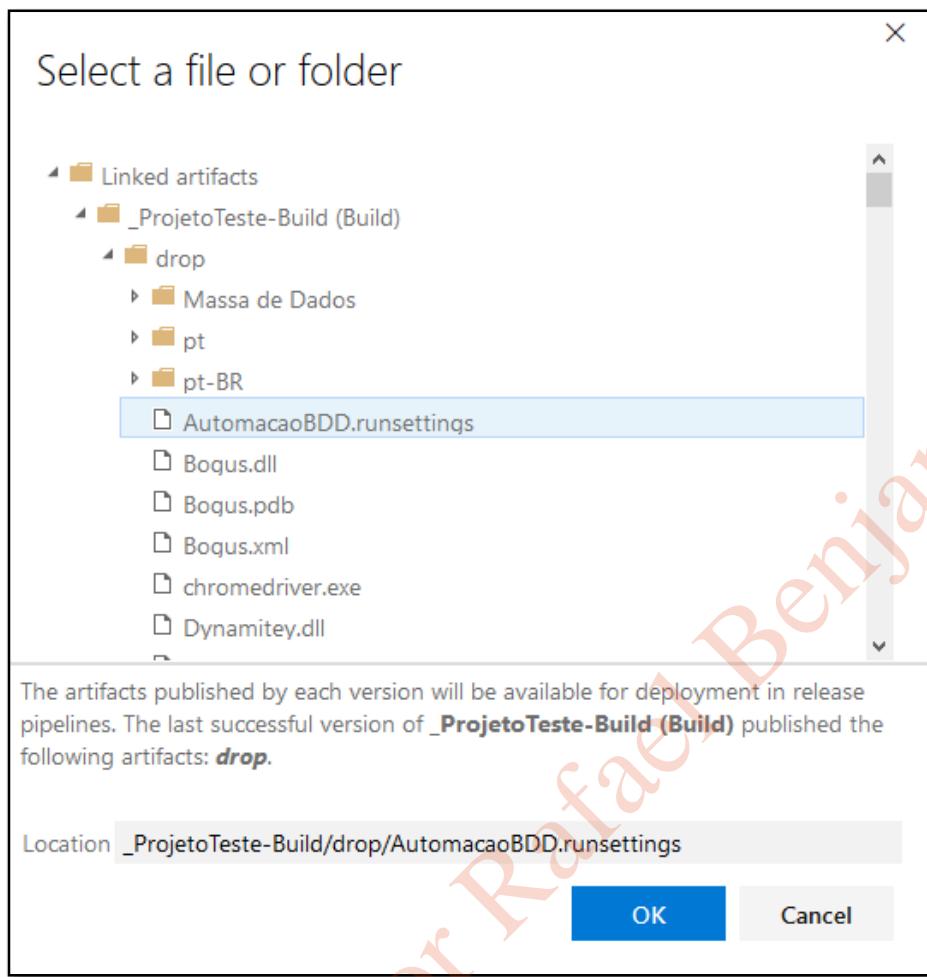
Clique na task **Visual Studio Test**, e agora no item **Select tests using**, escolha a opção **Test Run**

E marque o check **Test mix contains UI tests**

The screenshot shows the 'Test run for Test plans' task configuration. The 'Select tests using' dropdown is set to 'Test run'. The 'Test mix contains UI tests' checkbox is checked and highlighted with a red box.

Selecione o arquivo .runsettings no item **Settings file** do projeto correspondente e crie as variáveis se necessário:

The screenshot shows the 'Test run for Test plans' task configuration. The 'Settings file' dropdown is set to '\$(System.DefaultWorkingDirectory)/_ProjetoTeste-Build/drop /AutomacaoBDD.runsettings'. The 'Override test run parameters' field contains '-browser Chrome'.



Feito as configurações clique em **Save** na release.

All pipelines > **Testes de Regressao Loja**

Save Create release ...

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

_ProjetoTeste-Build

Schedule not set

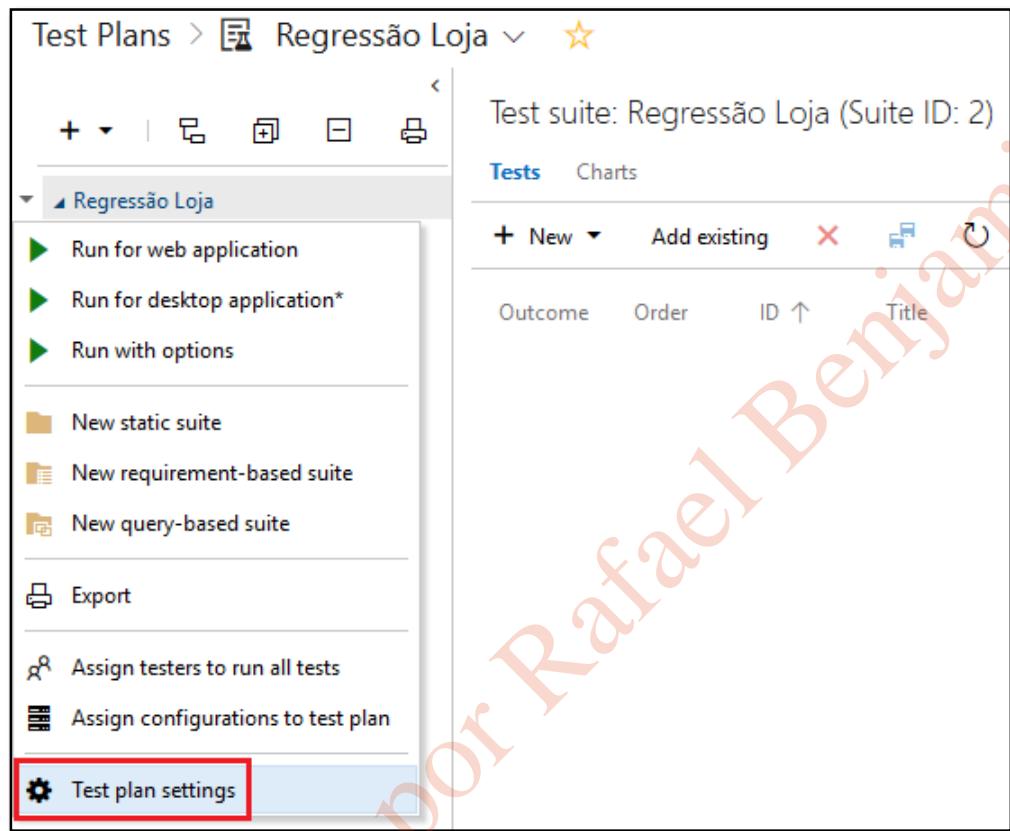
Stages | + Add

Stage 1
1 job, 1 task

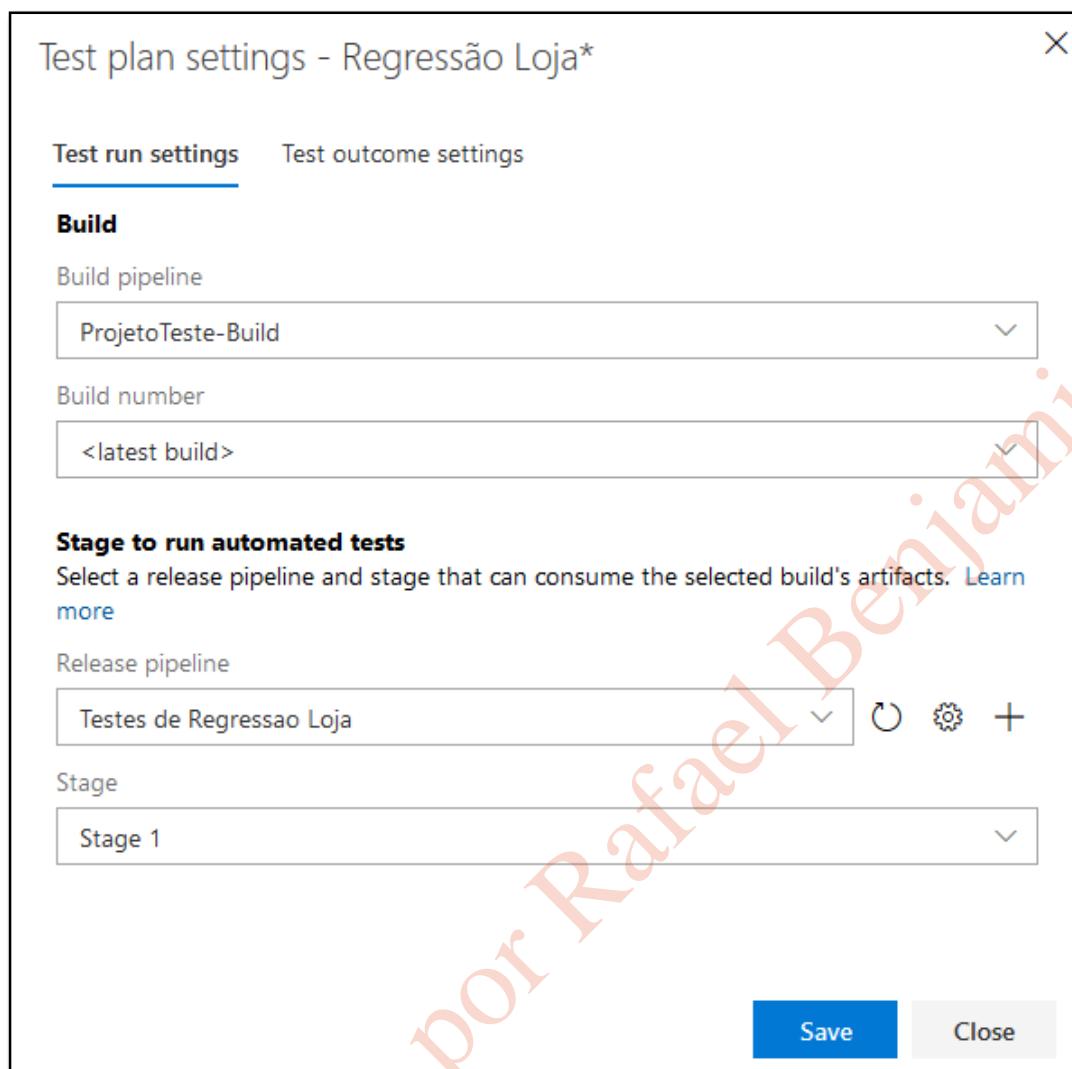
6.10 Configurando a release para execução via Test Plan

Agora com a release pronta para execução dos testes, vamos atualizar nosso test plan para ler os testes automatizados desta release.

Volte ao Test Plan desejado contendo os testes do projeto e clique em **Test Plan Settings**

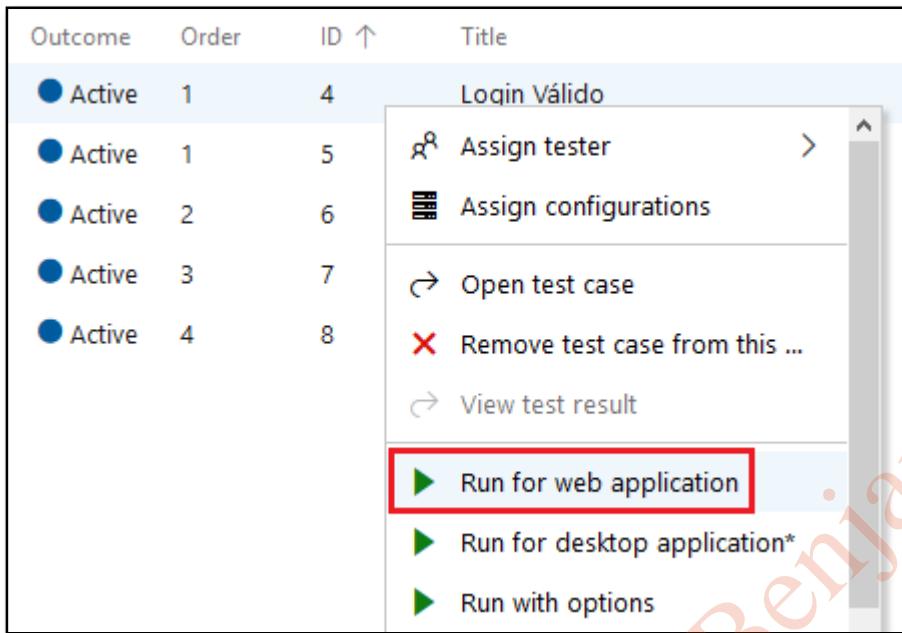


Aqui iremos selecionar de qual **Build** e qual **Release**, estão vindo nossos testes.



Feito isso, clique em **Save**.

Agora com tudo configurado, selecione qual teste deseja executar, clique com botão direito e selecione **Run for web application**



Se tudo correr bem, os três checks ficarão verdes. Agora você pode aguardar os testes serem executados, ou clicar em **View test run** para poder acessar a página onde ficarão os detalhes desta execução.



Ao clicar em **View test run**, a página abaixo será exibida. Caso queria ver os logs em tempo real da execução, clique em **View Logs**:

Run 4 - TestRun_Testes de Regressao Loja_Release-2

Run summary Test results Filter

⟳ | ⌂ Update comment |

Summary

● In progress , Running for 000 milliseconds

Run type	Automated
Owner	Treino Interno
Tested build	20190701.7
Release	Release-2 (View logs)
Release Stage	vsts://ReleaseManagement/Environment/2
Build platform	not available
Build flavor	not available
Test settings	903d851d-27ce-42a5-b8ad-79280d7c03cf
MTM lab environment	not available

Comments

No comments

Error message

No error message

Attachments ()

Outcome

1

None

Outcome by priority

Você será direcionado para a página da release onde os testes estão sendo executados:

↑ Testes de Regressao Loja > Release-2 > Stage 1 ✓ Succeeded

← Pipeline Tasks Variables Logs Tests Deploy Cancel Refresh Download all logs Edit ...

Deployment process Succeeded

Run on agent

Pool: Default · Agent: QARTI-NT086 Started: 01/07/2019 12:38:31 · 50s

Run on agent Succeeded · 2 warnings

Step	Description	Status	Time
1	Initialize job	succeeded	<1s
2	Download artifact - _ProjetoTeste-Build - drop	succeeded	4s
3	Test run for Test plans	succeeded 2 warnings	44s
4	Finalize Job	succeeded	<1s

Após o término da execução a página da Test Run é atualizada:

Run 4 - TestRun_Testes de Regressao Loja_Release-2

Run summary Test results Filter

| Update comment |

Summary

Completed just now, Ran for 30 seconds

Run type	Automated
Owner	Treino Interno
Tested build	20190701.7
Release	Release-2 (View logs)
Release Stage	vstfs:///ReleaseManagement/Environment/2
Build platform	not available
Build flavor	not available
Test settings	Default
MTM lab environment	not available

Comments

No comments

Error message

No error message

Attachments (1)

Name ↑	Size	Created Date
rafael.camargo_OARTI-NT08...	... 4K	just now

Outcome

Passed

Outcome by priority

Cliando na guia **Test results**, é possível ver todos os testes executados nesta run.

Run 4 - TestRun_Testes de Regressao Loja_Release-2 1 results (1 selected)

Test results Run summary Filter

| Create bug | Update analysis

Outcome	Test Case Title	Priority	Duration	Owner
Passed	Login Válido	2	0:00:15.470	Treino Interno

E ao dar duplo clique no teste é possível entrar em seu detalhe e visualizar seus anexos e detalhes de erros.

Run 4 - TestRun Testes de Regressao Loja Release-2 / Login Válido 1 of 1 tests

[Summary](#) [History](#)

[⟳](#) | [Bug](#) | [Update analysis](#) | [↓](#) [↑](#) [☰](#)

Summary		Analysis		
✔ Passed on QARTI-NT086		Owner Treino Interno Failure type None Resolution None Comment not available		
Run by	Treino Interno	Size	Created Date	Comment
Tested build	20190701.7	59K	just now	
Test Plan	1			
Priority	2			
Test suite	Geral			
Test Case	Login Válido			
Configuration	Windows 10			

Attachments (1)

	Name	Size	Created Date	Comment
<input checked="" type="checkbox"/>	LoginValido_Imagen_Final_01_07_2019 12_39_12.jpeg			
<input checked="" type="checkbox"/>	LoginValido_Imagen_Final_01_07_2019 12_...	59K	just now	...

Bugs (0)

Requirements (0)

Após as execuções o resultado também é refletido dentro da Test Suite.

Test suite: Geral (Suite ID: 3) NO iteration date

[Tests](#) [Charts](#)

	Outcome	Order	ID	Title	Configur...	Tester
✔	Passed	1	4	Login Válido	...	Windows ... Treino Int...
●	Active	2	5	Login Inválido	Windows ... Treino Int...	
●	Active	3	6	Fale Conosco Válido	Windows ... Treino Int...	
●	Active	4	7	Fale Conosco Inválido	Windows ... Treino Int...	
●	Active	5	8	Cadastro	Windows ... Treino Int...	

Exemplo de execução com mais de um cenário:

Test Plans > Regressão Loja

+ -

Regressão Loja

- Geral (5)**

Test suite: Geral (Suite ID: 3)

Tests Charts

+ New Add existing

Outcome	Order	ID ↑	Title
Active	1	4	Login Válido
Active	2	5	Login Inválido
Active	3	6	Fale Conosco Válido
Active	4	7	Fale Conosco Inválido
Active	5	8	Cadastro

Open test case Remove test case from this ... View test result

Run for web application Run for desktop application* Run with options Resume test

Edit selected test cases... Edit selected test cases in g...

No iteration dates

Tester All Configuration All View List

Order tests Column

Configurat... Tester
Windows ... Treino Int...
Windows ... Treino Int...
Windows ... Treino Int...
Windows ... Treino Int...
Windows ... Treino Int...

Run 14 - OnDemandTestRun 02/07/2019 16:26:55

Run summary Test results Filter

Update comment |

Summary

Completed just now, Ran for 01 minutes 04 seconds

Run type	Automated
Owner	Treino Interno
Tested build	<u>20190701.7</u>
Release	<u>Release-3 (View logs)</u>
Release Stage	<u>vstfs:///ReleaseManagement/Environment/3</u>
Build platform	not available
Build flavor	not available
Test settings	Default
MTM lab environment	not available

Comments

No comments

Error message

No error message

Attachments (1)

Name ↑	Size	Created
rafael.camara... OARTI-NT08...	6K	just now

Passed

Outcome by priority

Referência: <https://docs.microsoft.com/en-us/azure/devops/test/run-automated-tests-from-test-hub?view=azure-devops>

Anexo I

Links de artigos sobre automação de testes.

- Boas Práticas para Sucesso na Automação de Testes - <https://www.linkedin.com/pulse/10-boas-pr%C3%A1ticas-para-sucesso-na-automa%C3%A7%C3%A3o-reinaldo-mateus-rossetti/>
- Dicas para iniciantes: <https://blog.cedrotech.com/o-desafio-da-automacao-de-testes-dicas-para-iniciantes/>
- Melhores Práticas na Automação de Testes - <https://www.devmedia.com.br/automacao-de-testes/10249>
- Automação de testes – 7 passos para o sucesso - <http://nerds-on.com/2013/03/19/automacao-de-testes-7-passos-para-o-sucesso/>
- Testes Automatizados - Por onde eu começo ? -<https://www.knowledge21.com.br/blog/testes-automatizados-por-onde-eu-comeco/>
- Pirâmide de testes – <https://medium.com/venturus/pir%C3%A2mide-de-testes-uma-boa-estrat%C3%A9gia-para-automa%C3%A7%C3%A3o-de-testes-na-pr%C3%A1tica-1d87e64c3a44>
- Desafios da automação - <https://abstracta.us/blog/test-automation/the-4-most-common-test-automation-challenges-and-how-to-overcome-them/>
- Desafios da automação - <https://www.monitoratec.com.br/blog/5-desafios-na-automacao-de-teste/>
- How To Design Page Objects In Fluent Style <http://www.testautomationguru.com/selenium-webdriver-how-to-design-page-objects-in-fluent-style/>
- Automation Abstractions: Page Objects and Beyond - Conference Talk <https://www.youtube.com/watch?v=YExv0DIm0bc&feature=youtu.be>
- O que vem antes da automação de testes? - https://www.youtube.com/watch?v=_5IAGdC8VOo
- Comparação de Atributos NUnit X MsTest - <http://www.anarsolutions.com/automated-unit-testing-tools-comparison/>
- TDD não é teste unitário - <https://deviniciativa.wordpress.com/2019/06/19/tdd-nao-e-teste-unitario/>
- **Meetup - Como descrever cenários de teste utilizando Gherkin de forma correta -** https://www.youtube.com/watch?v=SAmwMD1_xJg
- **BDD não é Automação de Testes** <http://www.eliasnogueira.com/bdd-nao-e-automacao-de-testes/>
(Video) - <https://www.infoq.com.br/presentations/bdd-nao-e-automacao-de-teste>
- **Introducing Example Mapping** - <https://cucumber.io/blog/example-mapping-introduction/>
(Video) - https://www.youtube.com/watch?v=VwvrGfWmG_U

Livros sobre BDD:

- The Cucumber Book: Behaviour-Driven Development for Testers and Developers <https://www.amazon.com/Cucumber-Book-Behaviour-Driven-Development-Programmers/dp/1934356808>
- BDD in Action: Behavior-driven development for the whole software lifecycle <https://www.amazon.com/BDD-Action-Behavior-driven-development-lifecycle/dp/161729165X>
- Specification by Example: How Successful Teams Deliver the Right Software <https://www.amazon.com.br/Specification-Example-Successful-Deliver-Software/dp/1617290084>