Elias Nogueira

Passos para aprender Automação Web

Passos iniciais para aprender sobre automação de teste para web

Conteúdo

Sobre o autor

Elias Nogueira atualmente é Agile Coach, Instrutor, QA Engineer

Olá! Meu nome é Elias Nogueira e venho trabalhando na área de TI desde 1999.

Atualmente sou Agile Coach e Testador (o termo lá fora que usamos pra isso é QA Enginner) e meu primeiro trabalho foi como programador. E foi ai que, em 2006, conhecí o desenvolvimento ágil e também, dentro da programação, sobre TDD - Test Driven Development.

Foi quando eu decidi trabalhar com Qualidade de Software em todos os pontos, desde o código até a entrega final para o usuário.

Minha experiência

Nos últimos anos minha experiência tem foco em duas três distintas profissionalmente:

- Consultoria
- Treinamentos
- Automação de Teste

Tento alinhar os três no meu dia a dia profissional.

Apesar de eu, de certa forma, vender meu conhecimento através de consultorias e treinamentos eu também crio materiais gratuitos (como este) para que todos tenham como começar seus estudos em teste de software.

Copyright

Aprender Automação Web

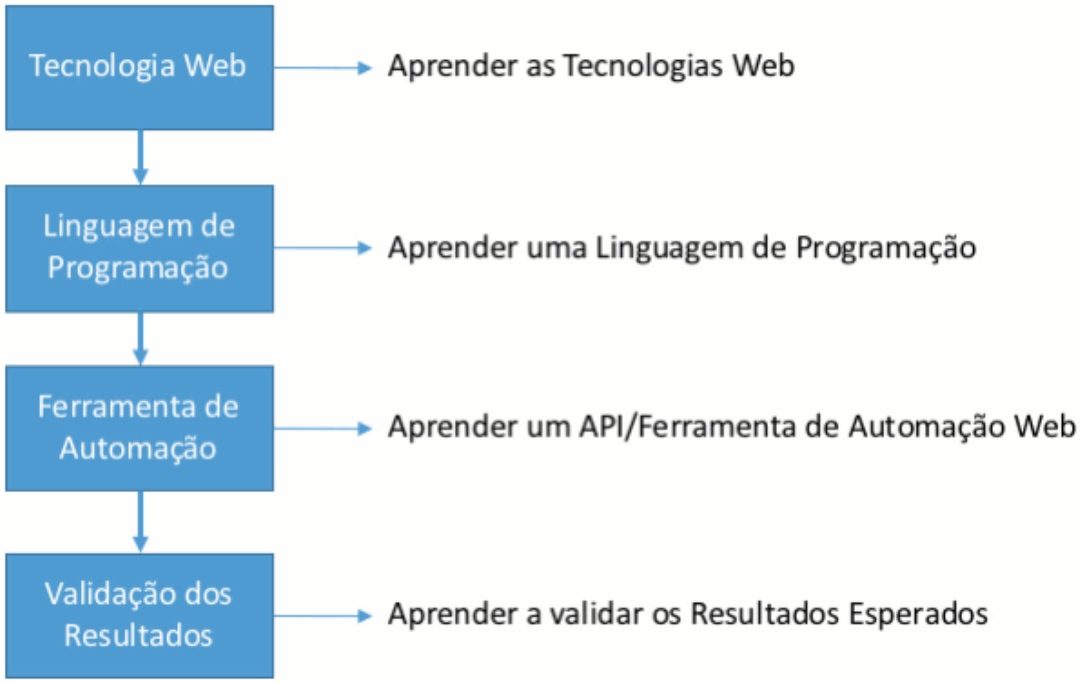
Com o advento das metodologias ágeis faz-se necessário aumentar a eficiência deste processo manual, onde ele é, algumas vezes, substituído pelo processo automatizado. Chamaremos este novo processo de Automação de Teste para Web.

Minha experiência

Eu venho automatizando aplicações web desde 2008 e hoje eu consigo aprender e aplicar muito rapidamente a automação de teste para web com diferentes ferramentas e linguagens.

O que eu vou compartilhar com vocês é a metodologia que eu utilizo para aprender e aplicar a automação de testes web com qualquer ferramenta em qualquer linguagem (que no caso eu conheça).

Passos para a Automação de Teste para Web



Parte 1 - Tecnologia Web

A base para a automação de testes para web é o entendimento de suas tecnologias. Isso irá ajudar você a entender como as páginas web se comportam e também para poder identificar os elementos web, ponto chave para o trabalho com estas ferramentas.

Tecnologia Web

Basicamente existem três tecnologias que fazem uma página web funcionar em um browser: HTML, CSS e JavaScript.

O **HTML** é a linguagem de marcação para a criação de elementos (texto, campos, etc...). Digamos que ele cuida da estrutura da página.
O **CSS** é o que estiliza a página (cores, posicionamento, etc...). Digamos que cuida da forma de apresentação visual da página.
O **JavaScript** é o que dá o dinamismo em uma página HTML. Digamos que ele cuida do comportamento dinâmico da página.

Porque é importante aprender as tecnologias Web?

Você quer automatizar testes para as páginas HTLM de uma aplicação, certo?
Um dos primeiros motivos é você conhecer o que você quer automatizar.

Diversas APIs e/ou ferramentas para automação de testes para web utilizam estas tecnologias para poder localizar e interagir com os elementos na tela. Também nos possibilitam simular eventos de JavaScript, e tudo isso servirá muito para criar e/ou melhorar um teste automatizado já criado.

Você não precisa conhecer 100% destas tecnologias, mas o mínimo que eu recomendo é você saber:

- Criar uma página web (HTML)
- Estiliza-la (CSS)
- Deixa-la dinâmica (Javascript)

Veja o exemplo do link abaixo, onde ele apresenta uma mensagem de erro quando você não informa um número de 1 a 10.

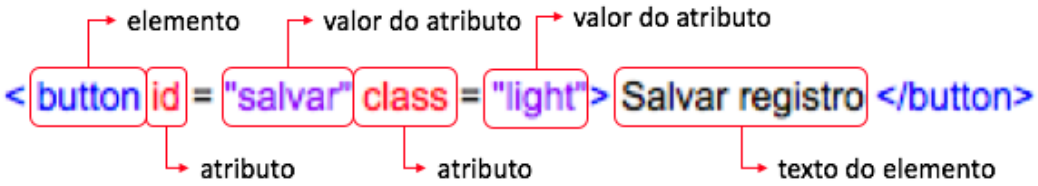
http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_validate

Sobre Elementos Web

Este é um dos maiores focos que você deve dar quando vai automatizar testes para páginas web.
Tudo dentro de uma página web é um elemento, não se esqueça disso e de utilizar esta nomenclatura: *elemento*! Ele é o ponto chave de toda a localização do elementos na tela para com a ferramenta de automação web.

O meu intuito não é te ensinar sobre estas tecnologias aqui (veja abaixo minhas dicas de estudo). Você só não pode esquecer a diferença entre quatro itens importantes: o elemento, o atributo, o valor do atributo e o texto do elemento. Se você souber identifica-los você está no caminho certo.

Abaixo uma pequena ilustração de como identifica-los através de um trecho de código-fonte HTML.



Agora é com você!

Mais pra frente, quando aprender sobre as três tecnologias, se você conseguir fazer algo semelhante ao que foi passado como exemplo, estará apto a passar para o próximo passo! :-)

Mas para conseguir isso eu sei que é necessário muito estudo.
A minha recomendação é a seguinte:

- Aprenda sobre as três tecnologias em conjunto
- Dê um certo foco aos seletores CSS, eles irão te ajudar muito na posterior localização de elementos com uma API/ferramenta de automação de testes para web

Há diversos cursos por na internet, e a grande maioria de graça!!!
Eu recomendo muito estes dois sites, onde você pode aprender da melhor forma: fazendo!

- W3Schools
  - Learn HTML
  - Learn CSS
  - Learn JavaScript

- CodeCademy
  - HTML & CSS
  - JavaScript
  - Make a Website (Criar um Website)

Parte 2 - Linguagem de Programação

Utilizamos uma linguagem de programação para script de teste automatizado. Independente de usar uma ferramenta/API devemos saber o básico da linguagem que escolheremos para que possamos criar o scripts de teste. Depois de aprender o básico já alguns padrões de projeto que podemos entender e aplicar para que o nosso script fique cada vez mais robusto e menos suscetível a falhas e manutenções.

Linguagem de Programação: o básico

Como estamos trabalhando com automação de teste é necessário escrever código com alguma linguagem de programação. Você não precisa ter o mesmo conhecimento de um desenvolvedor na sua empresa (se tiver melhor :P) mas isso não é um fator determinante para criar scripts de teste automatizado.

Mas você precisa conhecer o básico da linguagem de programação. Vou procurar explicar abaixo todos os pontos básicos que devemos aprender sobre uma linguagem, preferencialmente uma linguagem orientada a objetos.

Orientação a Objetos

> *Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. Fonte Wikipedia*

Explicação bonita né? Mas não encontrei melhor explicação, por isso a cópia do Wikipedia.

O que eu precisa aprender sobre programação básica

De preferência, dentro da orientação a objetos, é necessário começar do básico:

- **Classes:** vai representar um conjunto de objetos para serem utilizados. Um exemplo é que o nosso script de teste será uma classe, e você pode criar classes para apoiar em alguma ação dos scripts de teste

- **Tipos de Dados:** Sabemos que existem tipos diferentes de dados, não é mesmo? Os mais comuns para a utilização da automação de teste são *String* (texto), *int* ou *integre* (números inteiros) e *boolean* (booleano)

- **Atributos:** São as características do objeto que vão compor a estrutura de dados. Exemplo: Uma classe Pessoa poderia ter os atributos como nome, CPF, endereço, etc... No caso dos scripts de teste criamos atributos referentes a inicialização do browser (pensando em web) e atributos de suporte.

- **Métodos com retorno e sem retorno:** Muitos criam seus scripts de teste no método de teste (quando usam algum framework de teste unitário como suporte), mas é necessário também para uma bola legibilidade ou mesmo como suporte ao teste criar métodos (funções) que irão fazer algo específico. Eles podem não ter retorno (somente executar algo) ou retornar uma informação (um tipo de dado)

- **Loops:** existem diversos tipos de loops, que são códigos que executam N vezes. Os loops mais comuns são *while* (enquanto), *for* (para) e *d0...while* (faça... enquanto).

- **Condicionais:** são fatores de tomada de decisão onde podemos desviar o código para um determinado trecho ou ação. Os mais utilizados são *if-else* (se-senão), *switch-case* (desvie-caso). Tome cuidado no uso de condicionais no seu teste. Se o caso de teste tem uma derivação (mais de um teste), tente criar um novo teste ao invés de utilizar um *if* no seu código.

Exemplo a aplicação da programação básica

O código abaixo mostra um exemplo de todos os tópicos básicos referente a programação. Pode ser que você não entenda no início, mas ao longo dos seus estudos você será capaz de escrevem um script assim. É isso que muitos entrevistadores técnicos vão te perguntar ou querer que você faça algo semelhante.

Este script foi escrito em Java, é é uma automação de teste para a página de cursos da Qualister.

```
1  public class LinksAutomacao {
2
3     // Atributo que guarda a url que sera acessada
4     private final static String URL = "http://www.qualister.com.br/cursos";
5
6     @Test
7     public void test() {
8        WebDriver driver = new FirefoxDriver();
9        driver.get(URL);
10
11       List<WebElement> links = obtemLinks(driver);
12       listaCursosAutomacao(links);
13       listaCursosPerformance(links);
14
15       driver.quit();
16    }
17
18    // Metodo que retorna uma lista de elementos web que sao links na pagina
19    public List<WebElement> obtemLinks(WebDriver driver) {
20       return driver.findElements(By.cssSelector("a"));
21    }
22
23    /*
24     * Metodo que lista os cursos de automacao
25     * Ha a utilizacao de uma condicional (if) para verificar qual links tem o texto Automacao
26     */
27    public void listaCursosAutomacao(List<WebElement> links) {
28       for (WebElement link : links) {
29          if (link.getText().contains("Automação")) {
30             System.out.println(link.getText());
31          }
32       }
33    }
34
35    /*
36    * Metodo que lista os cursos de automacao
37    * Ha a utilizacao de uma condicional (if) para verificar qual links tem o texto Performance
38    */
39    public void listaCursosPerformance(List<WebElement> links) {
40       for (WebElement link : links) {
41          if (link.getText().contains("Performance")) {
42            System.out.println(link.getText());
43          }
44       }
45    }
46 }
```

O mesmo código acima pode ser visualizado através do link abaixo:

https://gist.github.com/eliasnogueira/d0907daca6d1...

Vou fazer o de-para de cada item:

- **Classe**: todo o arquivo, iniciando na *linha 1*
- **Tipo de dados e atributo:** *linha 4*, onde o tipo de dado é *String* (texto), criando o atributo chamado *URL*
- **Método com retorno:** *linha 19 a 21*, onde o método retorna todos os links existentes na página
- **Método sem retorno:** *linhas 27 a 33* e *linhas 39 a 45*, que são métodos que imprimem o texto dos links com uma determinada condição
- **Loops:** *linhas 28 a 32* e *linhas 40 a 44* são loops que vão iterar (percorrer) cada link da página
- **Condicionais:** *linhas 29 a 31* e *linhas 41 a 43* são condicionais verificando se o link tem determinado texto

O que posso aprender de forma mais avançada

Se você já sabe o básico de alguma linguagem de programação e quer tornar o seu teste ainda mais robusto, ou pretende depois de aprender o básico melhorar seu script ainda mais dê uma olhada nos três tópicos abaixo.

Page Objects

Page Objects é um padrão de projeto criado com foco em uma página web que concretiza uma página como uma classe e seus campos como atributos. Cada ação no atributo vira um método para utilização no script de teste. Há uma separação bem definida entre a classe criada (page objects) e seu teste. O seu teste irá utilizar os métodos do page object. Se alguma manutenção for necessária você não precisa alterar N testes, basta alterar diretamente no page objects.

Esse é um assunto muito importante para a automação de teste web e mobile, então, assim que possível tente estudar um pouco mais sobre este assunto.

Singleton

Singleton é um padrão de projeto que garante que uma classe tenha apenas uma instância e tenha um único ponto de acesso, que é global. Um exemplo concreto é a criação do browser. Seguidamente você precisará iniciar o browser web a todo momento, certo? Implementando o padrão Singleton para o browser você consegue além de iniciar apenas um browser para o seu teste (se assim você quiser) você pode inserir diversos browser diferentes para facilitar a execução e manutenção.

Recomendações de Curso e Estudo

A Loiane Groner tem um EXCELENTE curso gratuito de Java (a maioria dos testadores que eu conheço criam scripts de teste em Java) e eu recomendo fortemente, mesmo que você conheça um pouco sobre Java, fazer o curso ou aprender tópicos específicos.

http://loiane.training/curso/java-basico/
Também recomendo o site CodeCademy para aprender sobre outras linguagens também utilizadas para automação de teste, como Ruby ou Python. Eu fiz estes dois cursos e recomendo muito!

https://www.codecademy.com/pt-BR/learn/ruby

https://www.codecademy.com/pt-BR/learn/python

Infelizmente existem poucos materiais em Português referente a parte Page Objects e aplicação do Singleton com Selenium, por exemplo, no caso de automação web. Minha recomendação neste sentido é pesquisar no google mesmo e, se você tiver dúvidas, pode deixar um comentário aqui.

Parte 3 - Ferramentas de Automação

Há diversas APIs/ferramentas de automação de testes para web, cada uma com sua particularidade, funcionalidades e linguagens de programação diferentes.

Este post tem o intuito de te dar uma das dicas mais preciosas para a automação de teste para web: o que aprender em qualquer ferramenta para estar apto a utiliza-la.

Modelo de Aprendizado

Este é o meu modelo de aprendizado quando estou aprendendo uma nova API/ferramenta de automação de teste, não importando a linguagem de programação.

Se você focar nestes quatro itens você terá uma facilidade incrível na utilização da ferramenta, bem como uma maior velocidade no momento da criação do seu script de teste.

Navegação

A navegação é a forma com que a ferramenta irá interagir com o browser web. Isso inclui:

- Qual browsers web é possível utilizar
- Como abrir o browser
- Como abrir uma página
- Como efetuar uma atualização de página (refresh)
- Como simular o voltar (back)
- Como simular o avançar (forward)

Todos estes pontos são básicos para que possamos trabalhar com qualquer browser. O que podemos ter de avançado na navegação é como trabalhar com cookies.

A recomendação aqui é: você precisa aprender a fazer todos estes pontos. Se você não conseguir nem abrir o browser o script de automação nem irá executar.

Uma dica é procurar o browser recomendado pela ferramenta. Um exemplo prático na utilização do Selenium WebDriver é que o browser "padrão" e suportado totalmente é o Firefox. O GoogleChrome e outros browser só funcionam com algumas linhas de código a mais, o que pode dificultar o aprendizado. Foque sempre no mais fácil! ;-)

Interrogação

A interrogação é a capacidade de encontrar um elemento em uma página web. Qualquer ferramenta bem implementada para a automação de páginas web faz com que você possa localizar um elemento pelas seguintes formas:

Formas de localização:

- atributo ID
- atributo name
- por um nome de elemento
- por css selector

Algumas ferramentas também dão a possibilidade de localizar elementos através do XPATH.

Minha dica é simples: aprenda a localizar um elemento com a ferramenta selecionada em todas as formas citadas. Na grande maioria dos casos você irá utilizar o *atributo ID* ou *atributo name*, mas quando estes atributos não existem ou são dinâmicos você precisa utilizar alguma outra forma.

Manipulação

A manipulação é a capacidade de interação com um elemento web na página.
Vou separar em duas partes:

Parte básica

Consiste em efetuar as seguintes interações (ações):

- clique
- digitação de texto em um campo
- limpar o texto de uma caixa de texto
- pegar o texto de algum elemento

Parte avançada

Consiste em efetuar as seguintes interações (ações):

- arrastar e soltar
- mover
- clicar e segurar
- soltar (depois que você "clicou e segurou")
- chamar alguma menu de contexto (simular o botão direito do mouse)
- trabalhar com alguma tecla especial e/ou de atalho (F1, ALT, CTRL, etc..)

Minha recomendação é que você primeiro aprenda a interagir com

Sincronização

A sincronização é a capacidade da API/ferramenta de automação aguardar por uma ação. Podemos chama-la carinhosamente de espera!
Geralmente as ferramenta possuem duas abordagens de sincronização com diferentes nomes que vou apresentar abaixo mas com a mesma ação final:

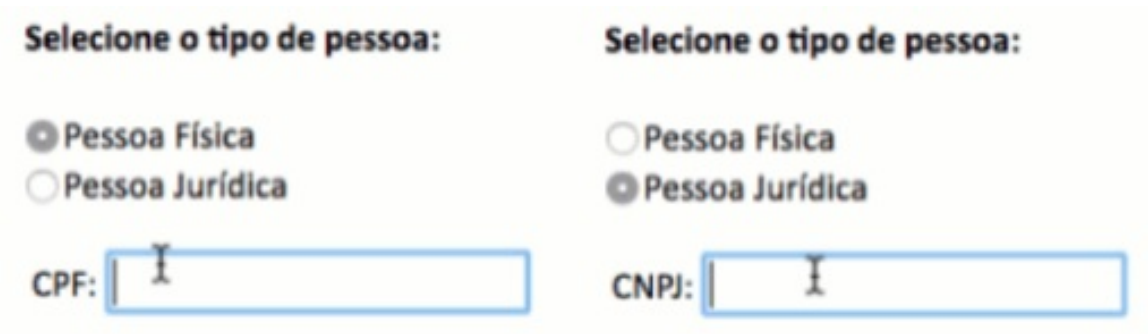
- Esperar por "qualquer coisa"
- Esperar quando você disser que tem que esperar

Já abordei um assunto semelhante a este referente a forma de sincronização no Selenium WebDriver. Se você quiser ver um exemplo disso acesse este post.

Vamos a dois exemplos práticos:

Exemplo 1: Você tem de ser capaz de clicar em um botão e esperar até que a mensagem "loading..." desapareça.

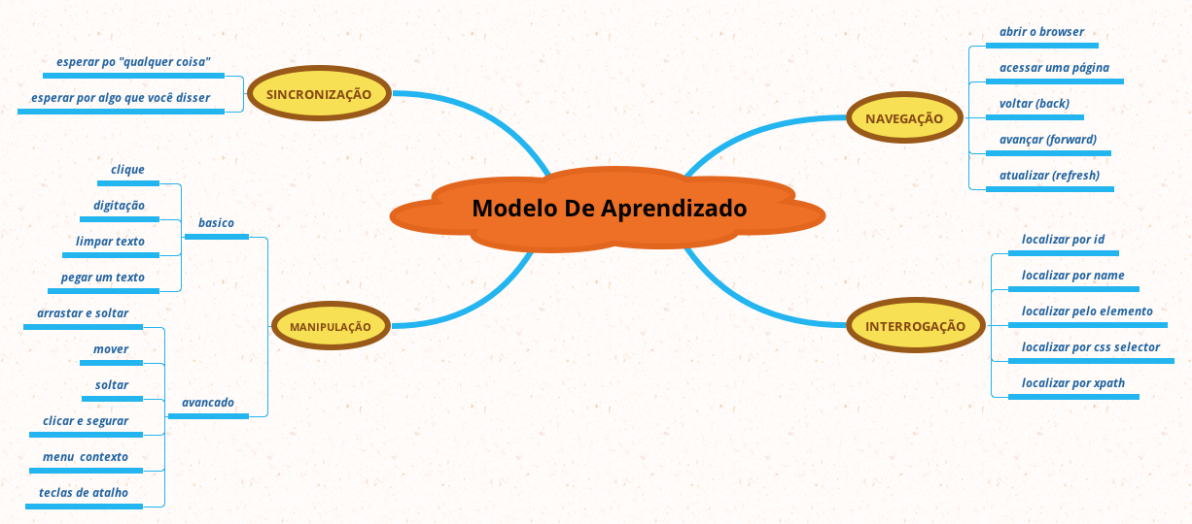
Exemplo 2: você pode selecionar, em um cadastro, o tipo de pessoa que pode ser Pessoa Física ou Jurídica.
Quando você clicar em Pessoa Física o campo CPF é apresentado (ele não estava visível anteriormente).
Quando você clicar em Pessoa Jurídica o campo CNPJ é apresentado (ele não estava visível anteriormente).



Notem que estes dois exemplos são comuns no nosso dia a dia.
O que você precisa fazer é entender como é a forma de sincronização (espera) da ferramenta que você adotar!

Dica

Se você quiser sempre lembrar deste modelo de aprendizado baixa a imagem abaixo.
Ela poderá te ajudar a sempre lembrar o que é necessário aprender em uma ferramenta de automação para web.



A imagem está disponível para download através do link abaixo:

mind_map_modelo_aprendizado.png

Parte 4 - Validação dos Resultados

Em qualquer aplicação de automação de teste para web é necessário validar os resultados esperados, sejam eles um texto apresentado em uma mensagem, a apresentação de um campo, o desaparecimento de uma imagem, etc...

Se não inserirmos uma (ou mais) validação(ões) de resultado(s) nosso script de automação será meramente um script de navegação e nada mais. Um dos pontos principais da automação de teste, que é a validação automática dos resultados será perdida.

O que são Validações de Resultados?

Em um teste manual, por qualquer que seja a forma que testamos (com base informal [ad-hoc test], com base formal [casos de teste] ou com base mediana de formalidade [teste exploratório]) é extremamente necessário a validação de resultados. Ela nada mais é que a comparação entre o resultado esperado versus o resultado obtido.

Com um exemplo simples, podemos explicar os dois termos. Digamos que temos um caso de teste bem simples escrito assim:

Dado que eu esteja autenticado e na página de Cadastro de Fornecedores
Quando eu preencher todos os campos obrigatórios e clicar no botão Cadastrar
Entao a mensagem "Fornecedor cadastrado com sucesso!" é apresentada
E eu visualizo a lista de todos os fornecedores cadastrados
E os dados informados como sendo o primeiro fornecedor da lista

Você já notou que existem três resultados esperados, certo?

1. A mensagem
2. Visualizar a lista de fornecedores cadastrados
3. Os dados do fornecedor cadastrado apresentado na primeira linha

Agora você executa o teste e a tela que você vê é a tela abaixo:

O *resultado esperado* é "Fornecedor cadastrado com sucesso!"
O *resultado obtido* foi "Registro duplicado!"

Em resumo o *resultado esperado* é aquele que esperamos previamente para a validação com sucesso. O *resultado obtido* é aquele apresentado e que devemos comparar para garantir que o sucesso da execução, neste caso o mesmo texto do do "Fornecedor cadastrado com sucesso!"

Aqui foi apresentado um erro na validação, pois as mensagens são diferentes.

asserções (assertions)

Para validar os resultados de forma automática existem ferramentas de teste, geralmente unitários, que provem a funcionalidade de asserções.

> *Nota: sempre utilizamos o suporte de uma ferramenta de teste unitário na escrita de qualquer script de automação de teste, mas não quer dizer que estamos desenvolvendo teste unitário. Apenas usamos as funcionalidades já existentes nestas ferramentas para facilitar diversos pontos, entre eles a validação dos resultados.*

Existem, basicamente, três tipos de asserções:

- igualdade: garante que um resultado obtido é igual ao resultado esperado
- verdadeiro/falso: garante que o resultado esperado é igual a verdadeiro ou falso
- nulo: garante que o resultado esperado é nulo

Neste três tipos existe sempre a palavra "não", que quer negar o que ele vai garantir.
Por exemplo, um "garantirNaoNulo" seria garantir que um resultado não é nulo.

Existem vários frameworks de teste unitário para diferentes linguagens. Todos eles suportam as asserções.
Para saber a lista de frameworks de teste unitário existentes para cada linguagem de programação, que chamamos de xUnit, acesse:
https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Exemplo de asserção em Java

Para a linguagem Java existem dois frameworks muito conhecidos:

- JUnit | http://junit.org/junit4/
- TestNG | http://testng.org

Os exemplos de asserções abaixo foram escritos usando JUnit.
Na dúvida de qual você pode usar eu recomendo o JUnit por ter mais materiais informativos, exemplos e fácil aplicação.

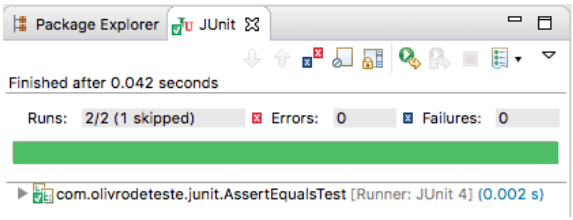
asserção para garantia de valores iguais

Essa asserção no JUnit é chamada de *assertEquals*. Veja o exemplo abaixo.

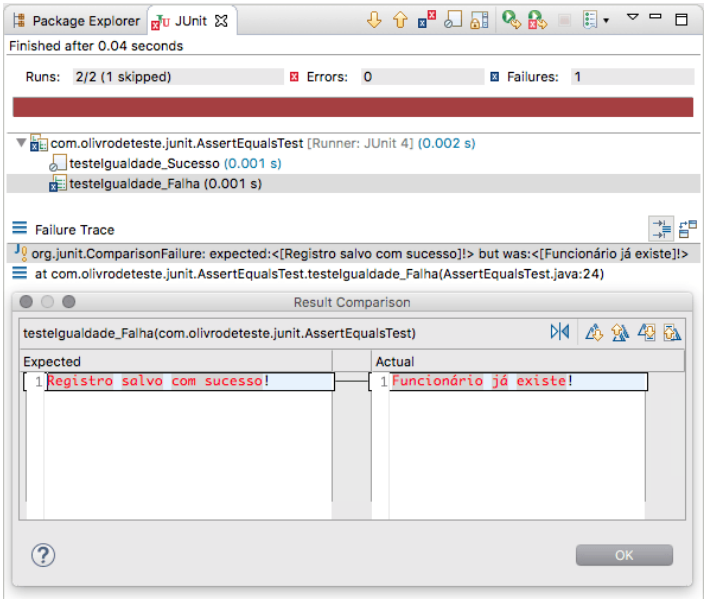
```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class AssertEqualsTest {
5
6  @Test
7  public void testeIgualdade_Sucesso() {
8  String resultadoEsperado = "Registro salvo com sucesso!";
9  String resultadoObtido = "Registro salvo com sucesso!";
10
11       assertEquals(resultadoEsperado, resultadoObtido);
12    }
13
14 @Test
15 public void testeIgualdade_Falha() {
16 String resultadoEsperado = "Registro salvo com sucesso!";
17 String resultadoObtido = "Funcionário já existe!";
18
19       assertEquals(resultadoEsperado, resultadoObtido);
20    }
21 }
```

Na **linha 8 e 9** temos o *resultadoEsperado* e *resultadoObtido* como sendo o mesmo texto (simulando o que seria uma mensagem sendo apresentada).
Na **linha 11** temos a função *assertEquals*, onde o primeiro parâmetro é referente ao resultado que esperados e o segundo parâmetro é o resultado obtido.

A execução deste teste tem o retorno como sucesso.



Na **linha 16 e 17** temos o *resultadoEsperado* diferente do *resultadoObtido*.
Na **linha 19** temos a validação (*assertEquals*) que mostrará um erro ao executar este teste, pois o resultado esperado é diferente do resultado obtido.



Na imagem é possível ver a primeira linha (com o fundo cinza) a apresentação do erro. No detalhe abaixo dessa linha há uma janela mostrando o que era esperado (Expected) e o que é o obtido (Actual). Como os dois são diferentes, o erro foi apresentado.

E agora?

Gostou desde eBook?

Fique ligado no meu blog para acessar mais conteúdos de forma gratuita sobre diversos tópicos e tecnologias referente a teste de software.

eliasnogueira.com