Curso 5 - Automação de Testes com Selenium WebDriver e Java

Automação de Testes com Selenium WebDriver e Java

Conheça o Selenium WebDriver, a principal ferramenta de automação de páginas Web. Nesse contexto, explore a linguagem de programação Java e entenda como o Selenium automatiza as ações diretamente em seu browser.

‼️ O SITE http://automationpractice.com/ INFELIZMENTE ESTÁ FORA DO AR, POR FAVOR, USAR QUALQUER OUTRO SITE PARA EXECUTAR OS COMANDOS DO SELENIUM WEBDRIVE. SUGIRO: https://automationexercise.com/‼️

Features

Configuração de Testes Selenium WebDriver + Java

WebDriver

Actions

WebDriverWait

Select

Demostração da Aplicação

Antes de começar, você precisará ter instalado em sua máquina as seguintes ferramentas:

Ferramenta	Versão
Java JDK	8+
Git	2.**
Maven	3.**

** Visando facilitar a demostração da aplicação, recomendo instalar apenas o Eclipse IDE e rodar o projeto através da IDE **

No Terminal/Console:

Entre na pasta raiz do projeto

Execute o comando: mvn test

Automatizando Testes com Selenium WebDriver e Java

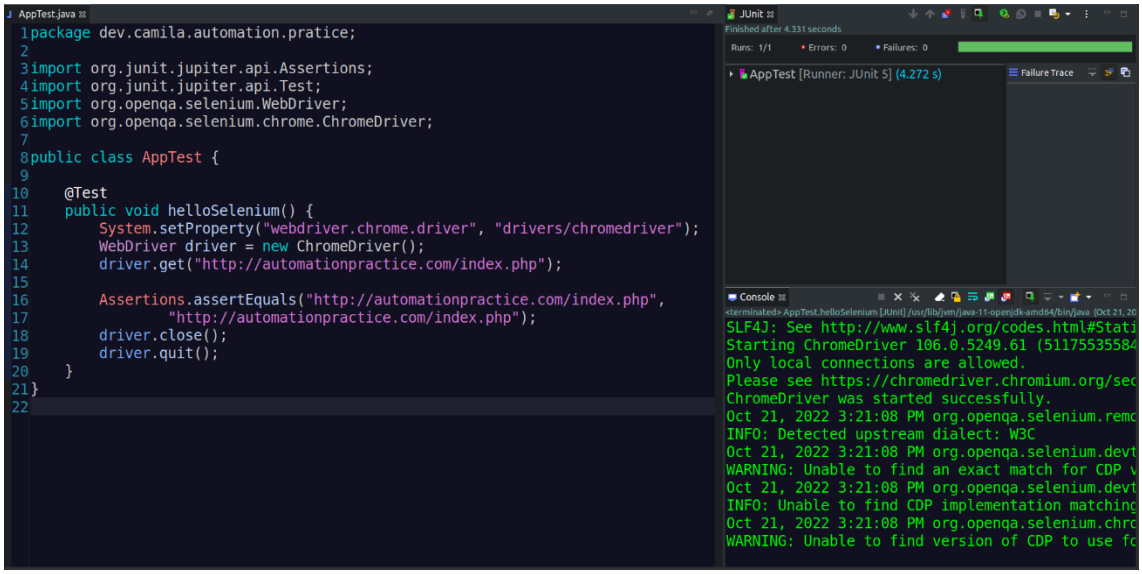
Selenium

- Selenium é um conjunto de ferramentas de código aberto multiplataforma, usado para testar aplicações web pelo browser de forma automatizada.
- Ele executa testes de funcionalidades da aplicação web e testes de compatibilidade entre browser e plataformas diferentes.
- O Selenium suporta diversas linguagens de programação, como por exemplo C#, Java e Python, e vários navegadores web como o Chrome e o Firefox.
- O ecossistema do Selenium é bem completa, tendo: Selenium IDE, Selenium WebDriver e Selenium Grid.
- O Selenium WebDriver usa o próprio driver do navegador para a automação.
- É a forma mais moderna de interação atualmente, pois cada browser possui o seu respectivo driver, permitindo a interação entre o script de teste e o respectivo browser.

Selenium WebDriver

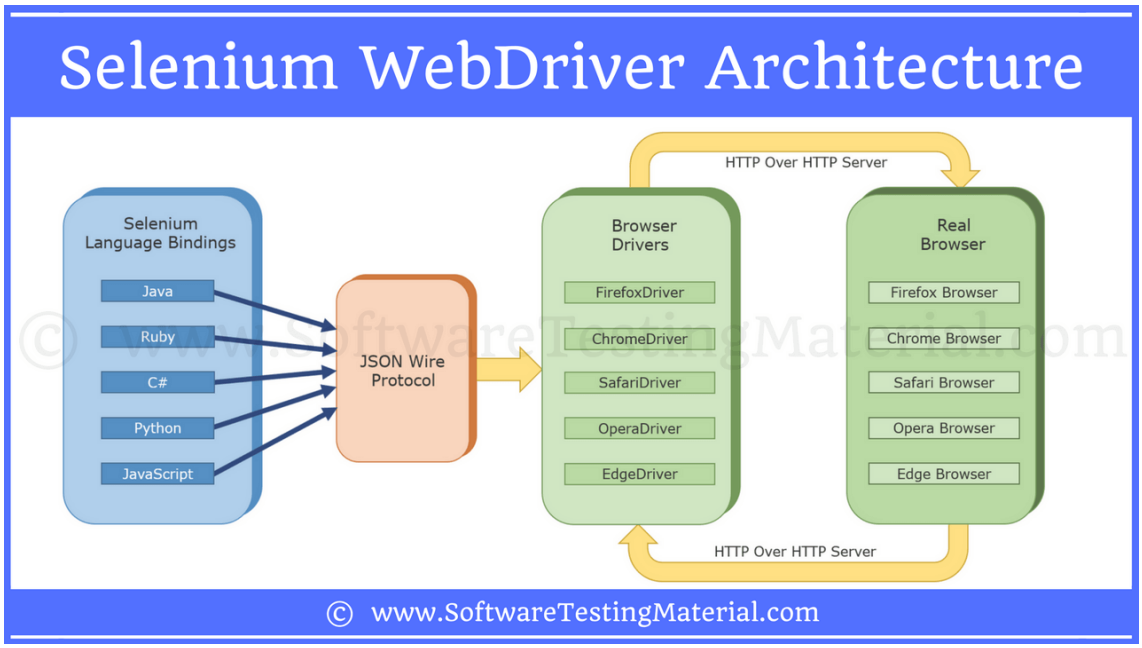
- O ecossistema do Selenium é bem completa, tendo: Selenium IDE, Selenium WebDriver e Selenium Grid.
- O Selenium WebDriver usa o próprio driver do navegador para a automação.
- É a forma mais moderna de interação atualmente, pois cada browser possui o seu respectivo driver, permitindo a interação entre o script de teste e o respectivo browser.

- Para efetuar os testes automatizados precisamos de um framework auxiliar de testes unitários e assim efetuar as operações de lógica de negócio na camada do servidor.
- Existem dois principais frameworks que podemos usar que se comunicam muito bem com o Selenium WebDriver: o JUnit e o TestNG.
- O JUnit é um framework open-source que possibilita a criação das classes de testes e tem como objetivo facilitar a criação de casos de teste, além de permitir escrever testes que retenham seu valor ao longo do tempo.



Teste Simples com Selenium WebDriver + Chrome + Java + JUnit 5

Arquitetura Selenium WebDriver



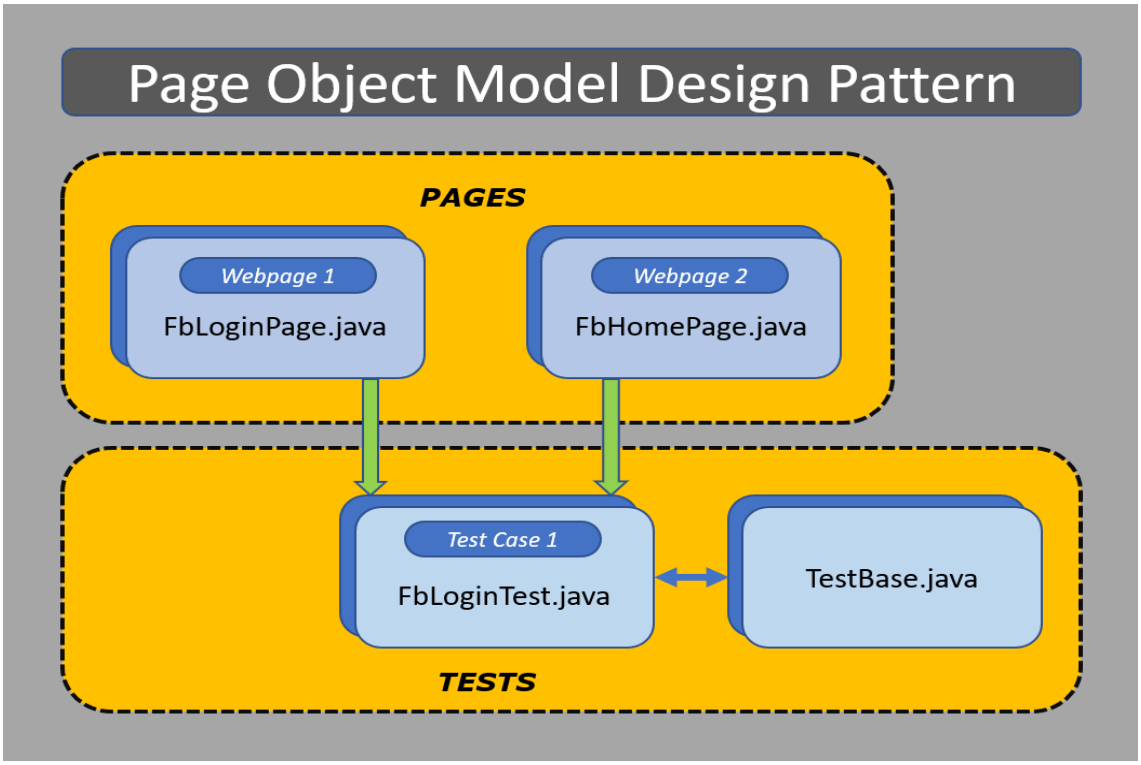
Arquitetura Selenium WebDriver

- A Selenium Client Library consiste em linguagens como Java, Ruby, Python, C# e etc. Após os casos de teste acionados, o código do Selenium será convertido para o formato Json.
- O Json gerado é disponibilizado para os drivers do navegador por meio do protocolo http.
- Cada navegador tem um driver de navegador específico. Assim que o driver do navegador recebe instruções, ele as executa no navegador. Em seguida, a resposta é dada de volta na forma de resposta HTTP.

Page Objects

É um padrão de design que se tornou popular na automação de teste para aprimorar a manutenção de teste e reduzir a duplicação de código. Um Page Object é uma classe orientada a objetos que serve como interface para uma página do seu AUT. Os testes usam os métodos dessa classe de objeto de página sempre que precisam interagir com a interface do usuário dessa página.

Page Object Model (ou POM como também é chamado) nos permite criar um repositório de objetos com elementos contidos numa página Web. Sob este modelo, para cada página, deve haver uma classe correspondente. Esta classe obtém e classifica os WebElements da página e também pode conter métodos que executam operações nesses WebElements. Ou seja, abstrair uma página HTML (ou parte) em uma API específica da aplicação, permitindo você manipular os elementos de página sem se aprofundar no HTML.



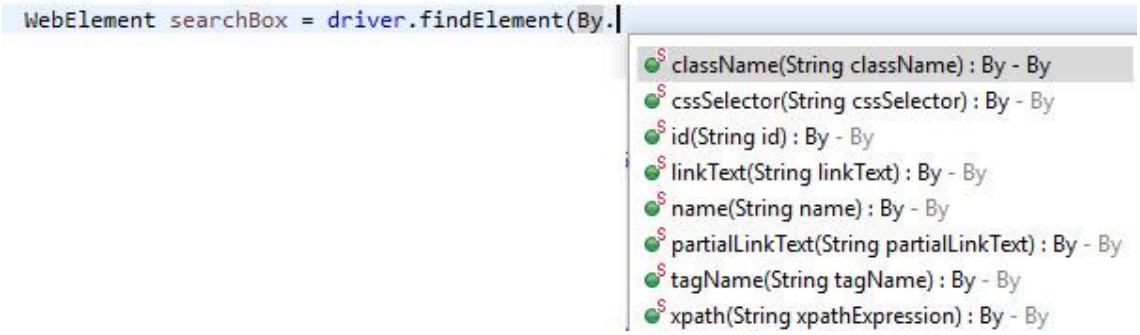
Alguns Locators Selenium WebDriver

Um localizador é uma maneira de identificar elementos em uma página. É o argumento passado para os métodos do elemento Finding.

Locator	Descrição	Desvantagens

By.id("id")	Este é o primeiro localizador que devemos tentar usar porque na maioria das vezes eles identificam o elemento de forma única.	Às vezes, o id é gerado automaticamente e é difícil, se não impossível, prever isso. Às vezes, os elementos não possuem um id completamente.
By.name("name")	Este localizador é o segundo que devemos tentar usar caso não tenhamos um id. Os nomes dos elementos geralmente são exclusivos e nos permitem localizar um elemento facilmente.	Às vezes, os nomes dos elementos podem não ser exclusivos. Alguns elementos podem não ter o atributo name.
By.tagName("tag name")	Este localizador procura o nome da tag do elemento dentro do DOM (Document Object Model).	Pode haver vários elementos com o mesmo nome de tag (por exemplo,)

By.xpath("xpathExpression")	Xpath que significa XML Path Language, é uma linguagem que permite percorrer e processar os elementos do DOM, por isso é muito útil encontrar um WebElement.	O WebDriver usa os recursos XPath nativos de um navegador sempre que possível. Nos navegadores que não possuem suporte nativo a XPath, o Selenium forneceu sua própria implementação. Isso pode levar a um comportamento inesperado, a menos que essas diferenças sejam tratadas nos vários mecanismos XPath. É necessário conhecimento de XPath.
By.cssSelector("css Selector")	Este localizador é em si uma estratégia de localização que usa a linguagem CSS (Cascade Style Sheet).	Nem todos os navegadores lidam com CSS da mesma maneira, então pode funcionar em alguns e não em outros. Necessário conhecimento em seletor css.



Exemplo da chamada de um Locator com Selenium WebDriver e Java

Interação com WebElement

Existem apenas 5 comandos básicos que podem ser executados em um elemento. Segue 3 comandos mais utilizados:

Comando	Aplicação	Descrição
Click	Aplica-se a qualquer elemento	O comando de clique do elemento é executado no centro do elemento. Se o centro do elemento estiver obscurecido por algum motivo, o Selenium retornará um erro de interceptação de clique no elemento.
Send Keys	Aplica-se apenas a campos de texto e elementos editáveis de conteúdo.	O comando element send keys digita as chaves fornecidas em um elemento editável. Normalmente, isso significa que um elemento é um elemento de entrada de um formulário com um tipo de texto ou um elemento com um atributo editável por conteúdo. Se não for editável, um erro de estado de elemento inválido será retornado.
Clear	Aplica-se apenas a campos de texto e elementos	O comando element clear redefine o conteúdo de um elemento. Isso requer que um elemento seja

	editáveis de conteúdo.	editável e reajustável. Normalmente, isso significa que um elemento é um elemento de entrada de um formulário com um tipo de texto ou um elemento com um atributo editável por conteúdo. Se essas condições não forem atendidas, um erro de estado de elemento inválido será retornado.

WebDriverWait (Explict Wait)

- Eles permitem que seu código interrompa a execução do programa ou congele o encadeamento, até que a condição que você passou seja resolvida.
- A condição é chamada com uma certa frequência até que o tempo limite da espera seja decorrido.
- Isso significa que enquanto a condição retornar um valor falso, ela continuará tentando e esperando.
- Como as esperas explícitas permitem que você aguarde a ocorrência de uma condição, elas são adequadas para sincronizar o estado entre o navegador e seu DOM e seu script WebDriver.
- Para remediar nosso conjunto de instruções com erros de antes, podemos empregar uma espera para que a chamada findElement aguarde até que o elemento adicionado dinamicamente do script seja adicionado ao DOM.
- ExpectedConditions nos fornece diversos métodos para verificarmos se existe algo no HTML, se está visível ou clicável, por exemplo.

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(20));

WebElement blouseaddToCart = driver.findElement(By.xpath("//*[@id=\"homefeatured\"]/li[2]/div/div[2]/div[2]/a[1]/span"));

blouseaddToCart.click();

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//*[@id=\"layer_cart\"]/div[1]/div[1]/h2")));

Exemplo de uso da classe WebDriverWait

Selenium Actions

- Selenium Actions permitem que você execute ações de interface do usuário em seu teste. Clicar, clicar duas vezes, passar o mouse ou outras ações complexas do mouse podem ser roteirizadas com uma ação.
- As interações avançadas do usuário, como segurar uma tecla enquanto clica em algo ou arrastar e soltar um item, são suportadas com as Selenium Actions.
- Essas ações são executadas pela API Advanced User Interactions, que consiste na classe Selenium Action para realizar essas interações.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofMinutes(10));

WebElement registerBtn = driver.findElement(By.id("submitAccount"));
registerBtn.click();

WebElement alertMessage = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/div[2]/div/div[3]/div/div")));

Actions action = new Actions(driver);
action.moveToElement(alertMessage).perform();
```

Exemplo de uso da classe Actions

Select

- O objeto Select agora lhe dará uma série de comandos que permitem que você interaja com um elemento select.
- Observe que esta classe só funciona para elementos HTML select e option.
- É possível projetar menus suspensos com sobreposições de JavaScript usando div ou li, e essa classe não funcionará para eles.

```
Select days = new Select(driver.findElement(By.id("days")));
days.selectByValue("13");

Select months = new Select(driver.findElement(By.id("months")));
months.selectByValue("1");

Select years = new Select(driver.findElement(By.id("years")));
years.selectByValue("1994");
```