Testing

Automatización de Pruebas

Selenium

Creado en el 2004 - Selenium IDE (inyección de código javascript)

2006 Aparece WebDrive (comunica directamente con el navegador)

Selenium IDE

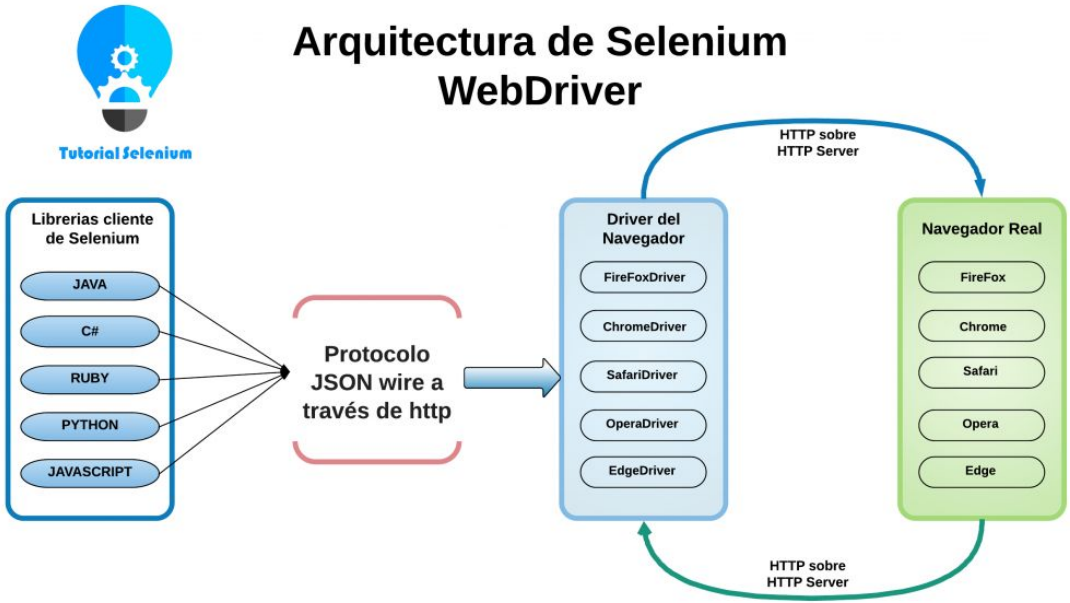
- Extensión de Firefox
- Permite construir y reproducir casos de prueba
- No permite bucles ni flujos de control

Selenium Web Driver

Permite la programación de test en diferentes lenguajes

Agrega el soporte para distintos navegadores

Arquitectura



JSON wire `http://localhost:8080/{"url":"https://www.tutorialselenium.com"}`

Ejemplo test java

```
package com.mystore.tests;
import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;
public class NewTest extends SeleneseTestCase {
        public void setUp() throws Exception {
                setUp("http://www.google.com/", "*firefox");
/                 / Se instancia e inicializa el navegador

                public void testNew() throws Exception {
                selenium.open("/");
                selenium.type("q", "selenium rc");
                selenium.click("btnG");
                selenium.waitForPageToLoad("30000");
                assertTrue(selenium.isTextPresent("Results * for selenium rc"));
                // Estos pasos representan las acciones del usuario
        }
}
```

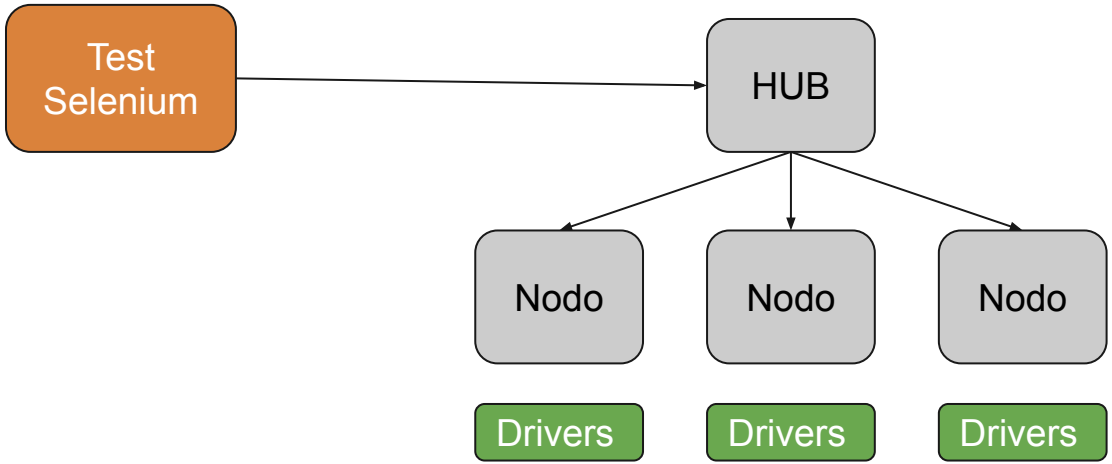
Selenium GRID

Permite controlar la ejecución de test

Puede interactuar con máquinas remotas para probar distintos navegadores y sistemas operativos

Selenium GRID

Selenium GRID permite ejecutar test en diferentes máquinas y diferentes browsers en paralelo.



Ejemplo de código

```
Webdrive driver = new FirefoxDriver();
driver.get(“http://www.google.com”)
WebElement searchBox = driver.findElement(By.name(“q”));
searchBox.sendKeys(“selenium”);
searchBox.submit();
System.out.println(“Title: “ + driver.getTitle());
```

Selectores CSS o Xpath

- CSS: Son los más utilizados

  Ejemplo:

  <input type="text" id="fistname" name="first_name" class="myForm">

  css = element_name[<nombre_atributo>='<value>']

  css=input[name='first_name']

  css="input#firstname" o solo css="#firstname" (por id)

  css="div[class='ajax_enabled'] [style='display:block']" (seleccionar por varios atributos)

  css="div#child img" (seleccionar un hijo)

  css="ul#fruit li:nth-of-type(2)" (seleccionar un elemento dentro de un listado)

  css="ul#fruit li:last-child" (seleccionar el último del listado)

  css="div[id^='123']" (el elemento con id que comienza con)

  Input:disabled (los input deshabilitados)

Page Object Model

El page object model es un patrón de diseño de objetos, donde las páginas web se representan como clases y los diversos elementos de la página se definen como variables en la clase.

Page Object

Implementar las interacciones como métodos:

clickLoginButton();

setCredentials(user_name,user_password);

Instanciar la página para utilizarla

Página LoginPage = PageFactory.intElements (driver, LoginPage.class)

Buscar los elementos

@FindBy(name="username")

private WebElement user_name;

Page object

```
@FindBy(id="username")
private WebElement user_name;

@FindBy(name="passsword")
private WebElement user_password;

@FindBy(className="h3")
 private WebElement label;

@FindBy(css=”#content”)
private WebElement text;
```

Implicit wait

Reemplazar los sleep que me hacen perder tiempo que puede ser innecesario.

Puedo esperar hasta que se cargue la página, en cuanto carga continúa. Sino espera hasta 10 segundos

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Si no carga a los 10 segundo me da un timeout

Explicit wait

Espera hasta que se cumpla una condición n y, si no sucede falla.

```
WebDriverWait wait = new WebDriverWait(driver,10);
WebElement campoUsuario = wait.until(ExpectedConditions.presenceOfElementLocated(inputUsuario));
```

Selenium IDE

npm install -g chromedriver

npm install -g selenium-side-runner

https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd

https://app.lambdatest.com/console/screenshot

selenium-side-runner -w 4 "UM2020.side" --server "https://fabian.contigliani@um.edu.ar:y6zO2qbrEeAMqbRBaaymRRfAfi5UvV6qMOXUwScBhT6JtEiQ7H@hub.lambdatest.com/wd/hub" -c "browserName='chrome' version='72.0' platform='Windows 10'"

¿Qué probar?

De una aplicación Web

Pruebas funcionales

- Revisar los enlaces y links
  - Probar los enlaces salientes de todas las páginas
  - Probar los enlaces internos
  - Probar los enlaces de salto dentro de la misma página
  - Probar enlaces de envío de email.

Pruebas funcionales

- Probar formularios
  - Revisar las validaciones de cada campo.
  - Probar los valores por defecto de cada campo
  - Probar el ingreso de datos incorrectos en los formularios