DIGITALHOUSE

Conceptos: Automatización WEB con Selenium

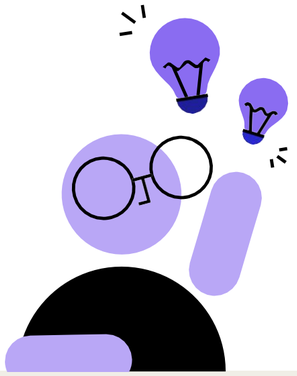
Índice

01 Patrones de diseño para la automatización de pruebas

La automatización de pruebas mediante herramientas como **Selenium** y otras, son esencialmente actividades de desarrollo.

Se necesitan buenos conocimientos de **programación** y **patrones de diseño** para implementar pruebas robustas, estables, eficientes, rápidas y sostenibles.

Patrones de diseño



Modelo Page Object

El modelo **Page Object**, también conocido como POM, es un patrón de diseño que **crea un repositorio de objetos para almacenar todos los elementos de la web**. Es útil para reducir la duplicación de código y mejorar el mantenimiento de los casos de prueba. En el Modelo Page Object, consideren cada página web de una aplicación como un archivo de clase. Cada archivo de clase contendrá sólo los elementos correspondientes de la página web. Con estos elementos, los probadores pueden realizar operaciones en el sitio en prueba.

Screenplay

Es un enfoque centrado en el usuario para escribir pruebas de aceptación automatizadas de alta calidad. Orienta el uso eficaz de las capas de abstracción y fomenta los buenos hábitos de prueba e ingeniería del software. Se compone de cinco elementos, cinco tipos de bloques de construcción que le sirven para diseñar cualquier prueba de aceptación funcional que necesite, por sofisticada o sencilla que sea. Los elementos clave son: actores, habilidades, interacciones, preguntas y tareas.

Marcos de trabajo para la automatización de pruebas

Se trata de un conjunto de directrices o reglas utilizadas para **crear y diseñar casos de prueba**. Un framework compuesto por una combinación de prácticas y herramientas diseñadas para ayudar a los profesionales del control de calidad a realizar pruebas de forma más eficaz. Estas directrices pueden incluir normas de codificación, métodos de manipulación de datos de prueba, repositorios de objetos, procesos de almacenamiento de resultados de pruebas o información sobre cómo acceder a recursos externos.

Hay seis tipos comunes de frameworks de automatización de pruebas, y a la hora de crear un plan de pruebas es importante elegir el adecuado para usted.

02

Ubicación de los elementos

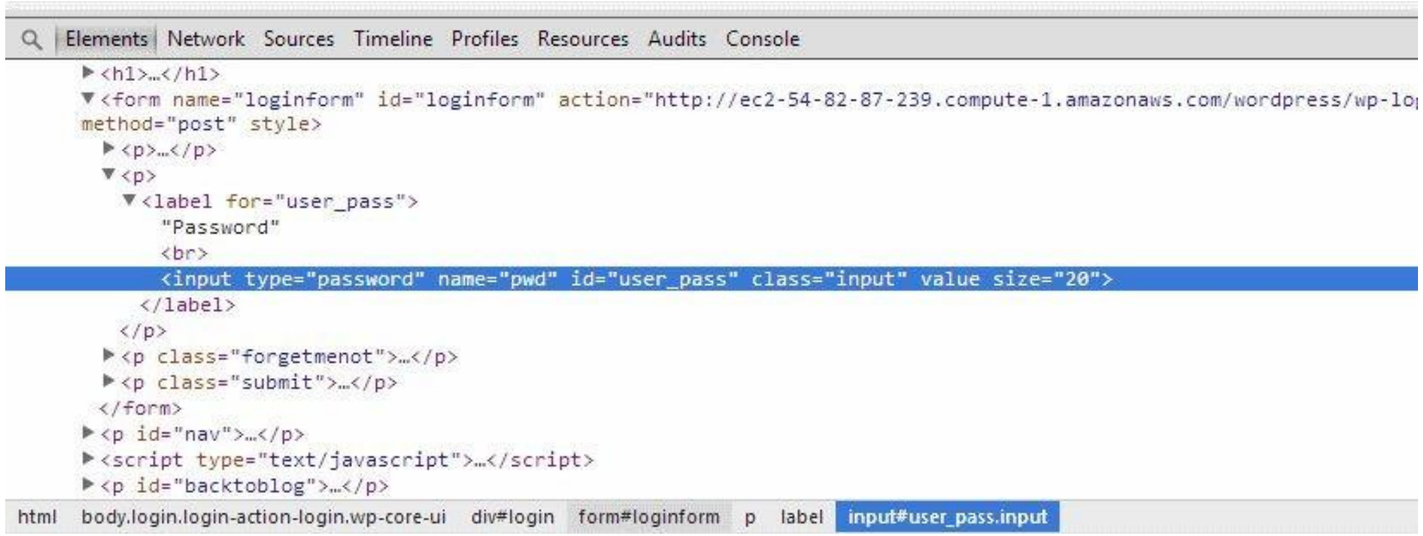
Ubicación de los elementos

Todo lo que se ve en la página es un elemento. Todos los campos, enlaces, imágenes, textos y muchas cosas que no se ven son elementos, aunque puede haber elementos en una página que no provengan de la fuente HTML.

- **Localizador de elementos** - Es un método para localizar un elemento en una página. Hay muchos tipos diferentes de localizadores, decidir cuál usar depende de muchos factores diferentes y no hay una respuesta correcta más que la que funcione para ustedes. En muchos casos, se pueden utilizar varios localizadores diferentes para realizar la misma tarea.

Ubicación de los elementos

- **Encontrar e inspeccionar el elemento -** Necesitamos saber con qué estamos trabajando, así que el siguiente paso es siempre mirar el elemento en la página para ver qué propiedades tiene, y que podemos usar para identificarlo. El navegador (o un plugin) te ayudará a hacerlo.



Ubicación de los elementos

- **Elijan un localizador -** Después de inspeccionar el elemento, es el momento de intentar localizarlo. Entre las diversas estrategias siguen algunos ejemplos como:

  El atributo **id:** el campo de la contraseña tiene un atributo ID (id="user_pass"), que normalmente sería la primera opción. Pero como ya hemos determinado que el ID cambia con cada iteración, sabemos que esto no funcionará en este caso.

  El atributo **name**: este elemento es un elemento de entrada a un formulario y tiene un atributo name (name="pwd"). Un nombre de campo suele ser una buena segunda opción, después del id del elemento. Pruebe un localizador de nombres de campo con el nombre "pwd".

  El atributo **class**: el elemento tiene una clase (atributo) CSS. Si este es el primer (o único) elemento de la página que utiliza esta clase, entonces el elemento puede ser localizado en el localizador de clases CSS con la clase "input"..

Ubicación de los elementos

- **XPath:** Un XPath siempre está disponible para un elemento, de hecho, normalmente hay múltiples XPaths válidos para un elemento. XPath es un recurso potente y flexible. Sabemos que el navegador puede proporcionar un XPath, sin embargo, como este elemento tiene un id, el navegador generará un XPath basado en el id que ya sabemos que no podemos utilizar en este caso. Podemos ver que el formulario padre también tiene un id, si ese id no cambia, un XPath desde ese elemento al campo contraseña debería funcionar.

  //*[@id="loginform"]/p[2]/label/input

  Este XPath sigue utilizando un ID, pero el ID del formulario en lugar del campo de la contraseña. Si el ID del formulario también cambia con cada vista de la página, esto tampoco funcionará y habrá que considerar otro tipo de localización.

Ubicación de los elementos

- **Seletor CSS:** Muchos elementos pueden seleccionarse fácilmente utilizando selectores CSS. En el ejemplo anterior, el elemento tiene una clase CSS de entrada y es probablemente el único campo de entrada en la página con el atributo type=' password', por lo que este selector CSS podría funcionar: [type='password'].input

- **Link Text:** Los links se pueden localizar haciendo coincidir el texto del enlace con el parámetro proporcionado.

- **Text:** Los elementos de uso común que tienen una etiqueta o un texto visible pueden localizarse a menudo mediante este sencillo localizador.

03 Before y After test

Métodos para ejecutar antes y después de cada test

El método **@Before** se ejecuta antes de la ejecución de una prueba.

El método **@After**, en cambio, se ejecuta después de cada prueba. Esto se puede utilizar para configurar repetidamente una prueba, y borrarla después de cada ejecución.

Por lo tanto, las pruebas son independientes y el código de configuración no se copia en el método de prueba y ambos deben ser public void.

Ejemplo

```java
import static org.junit.Assert.assertEquals;

import java.util.ArrayList;
import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class DemoTest{
  private List<Integer> list;

  @Before
  public void setUp(){
    list - new ArrayList<>();
    list.add(3);
    list.add(1);
    list.add(4);
    list.add(1);
    list.add(5);
    list.add(8);
  }

  @After
  public void tearDown(){
    list.clear();
  }

  @Test
  public void shouldBeOkAfterTestData(){
    list.remove(0); // Remove first element of list
    assertEquals(5,  list.size()); // Size is down to five
  }

  @Test
  public void shouldBeIndependentOfOtherTests(){
    assertEquals(6, list.size());
  }
}
```

Tareas comunes antes de la ejecución (utilicen @BeforeTest):

- Abran el navegador.
- Establecer una página web/sitio web/URL específica para iniciar.
- Limpieza de la caché del navegador.

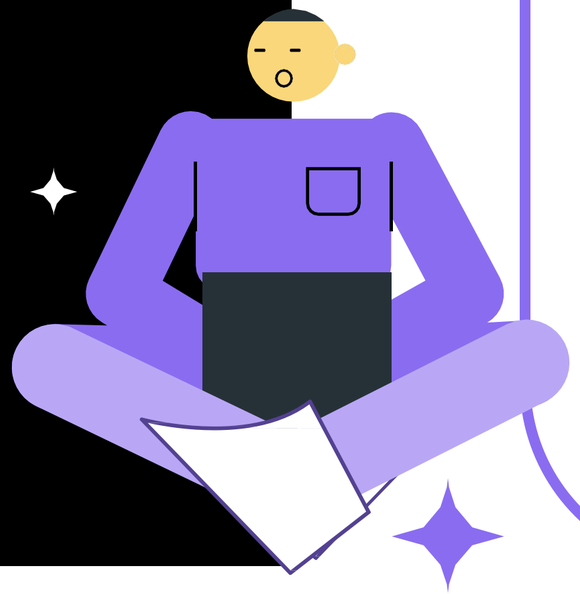
Tareas comunes después de la ejecución (utilice @AfterTest):

- Envíen los resultados de las pruebas a un servicio de registro o supervisión.
- Cierre el navegador.
- Creen un informe después de ejecutar la prueba.
- Recojan capturas de pantalla.
- Las variables se pueden borrar o restablecer.

04 Uso de anotaciones

Las anotaciones se utilizan para **controlar el siguiente método a ejecutar** en el script de prueba. Se colocan antes de cada método en el código de prueba. Si algún método no está prefijado con anotaciones, será ignorado y no se ejecutará como parte del código de prueba.

Para definirlos, basta con anotar los métodos con **'@Test'.**



Anotaciones

A continuación se muestra la lista de anotaciones que lo soportan en Selenium:

- **@BeforeMethod:** Se ejecutará antes de cada método anotado @test.
- **@AfterMethod:** Se ejecutará después de cada método anotado en @test.
- **@BeforeClass**: Se ejecutará antes de la primera ejecución del método @Test. Se ejecutará sólo una vez en todo el caso de prueba.
- **@AfterClass**: Se ejecutará después de que se hayan ejecutado todos los métodos de prueba de la clase actual.
- **@BeforeTest:** Se ejecutará antes del primer método @Test anotado. Se puede ejecutar varias veces antes del caso de prueba.

Anotaciones

A continuación se muestra la lista de anotaciones que lo soportan en Selenium:

- **@AfterTest:** Un método con esta anotación se ejecutará cuando todos los métodos anotados a @Test hayan terminado de ejecutar esas clases dentro de la etiqueta
  <test> en el archivo TestNG.xml.
- **@BeforeSuite**: Se ejecutará sólo una vez, antes de que se ejecuten todas las pruebas de la suite.
- **@AfterSuite**: Un método con esta anotación se ejecutará una vez que se hayan completado todas las pruebas de la suite.
- **@BeforeGroups**: Este método se ejecutará antes de la primera prueba de ese grupo específico.
- **@AfterGroups:** Este método se ejecutará después de que todos los métodos de prueba de ese grupo hayan terminado su ejecución.

DIGITALHOUSE

¡Muchas gracias!