Automatización de pruebas funcionales.
Ejercicio con Selenium

Trabajo sesión seminario 21 de marzo ACG
Curso 2022-2023

Ya hemos visto durante una sesión de teoría qué son las pruebas funcionales. Este tipo de pruebas verifican que el producto software bajo test cumple con lo especificado y hace lo que debe y cómo lo debe hacer. Para la planificación de pruebas funcionales, tendremos que recurrir a los documentos de especificación de requisitos o funcionalidad acordada con los clientes.

También hemos visto la importancia de automatizar las pruebas de software. Cuando hablamos de automatización de pruebas funcionales, nos referimos a la construcción de un conjunto de scripts reutilizables que prueban distintas funciones del software. Estos scripts aumentan la capacidad de testear software en lo que respecta a pruebas de regresión antes y después de la publicación de una nueva versión del software. Aunque la automatización de pruebas de software no garantiza la calidad del software, sí que ayuda al equipo de pruebas a mejorar sus resultados. La automatización de pruebas permite reducir los costes de cualquier organización que necesite probar sucesivas versiones de un mismo producto.

Existen en el mercado diferentes herramientas para la automatización de pruebas funcionales. Una de las más utilizadas es Selenium[1]. Selenium es un entorno para la automatización de pruebas de software de código abierto. Se utiliza para automatizar pruebas en aplicaciones web. Con Selenium, los testers pueden crear scripts de prueba que imitan las acciones que un usuario realizaría en un navegador web, como hacer clic en botones, introducir datos y navegar por páginas. Estos scripts se pueden ejecutar automáticamente para verificar el comportamiento de la aplicación y detectar errores.

Entre los conceptos básicos de Selenium, destacamos los siguientes:

- WebDriver: Webdriver es una API que se encarga de enviar comandos al navegador web para realizar acciones, como hacer clic en un botón, ingresar texto en un campo de entrada, navegar a una nueva página, etc.
- Locator: Un *locator* es una forma de identificar elementos en una página web, como botones, campos de entrada y enlaces. Selenium utiliza distintos selectores para identificar estos elementos, como ID, nombre, clase, selector de CSS, selector de XPath, etc.
- Assertion: Un assertion es una declaración que verifica si un resultado esperado es igual al resultado real. Las assertions se utilizan en los test scripts de Selenium para verificar que la aplicación se comporta como se espera.

A continuación se muestran varios ejemplos con código java que muestran el uso de los conceptos básicos de Selenium para el desarrollo de scripts de pruebas.

[1] https://www.selenium.dev/

- Configurar el entorno de prueba y crear una instancia de WebDriver. Esta instancia permitirá interactuar con el navegador.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

// Configurar la propiedad del sistema para indicar dónde se encuentra el controlador de Chrome
System.setProperty("webdriver.chrome.driver","/ruta/al/controlador/chromedriver");

// Crear una instancia de ChromeDriver
WebDriver driver = new ChromeDriver();
```

- Encontrar distintos elementos de una página web.

`findElement(By.id("xxx"))` → Localiza el elemento con *id* especificado.
`findElement(By.name("xxx"))` → Localiza el elemento con *name* especificado.
`findElement(By.cssSelector("xxx"))` → Localiza el elemento con el selector css especificado.

```
WebElement textBox = driver.findElement(By.name("my-text"));
WebElement submitButton = driver.findElement(By.cssSelector("button"));
WebElement elementUser = driver.findElement(By.id("ap_email"));
```

- Realizar acciones sobre elementos de una página web.

```
textBox.sendKeys("Selenium");
submitButton.click();
```

- Otras funciones básicas de Selenium.

```
// Maximizar la ventana del navegador
driver.manage().window().maximize();

// Navegar a una página
driver.get("https://www.example.com");

// Obtener título de la página
String title = driver.getTitle();

// Consultar información de un elemento
WebElement message = driver.findElement(By.id("message"));
String value = message.getText();

// Cerrar sesión
driver.quit();
```

- Comprobar resultados.

```
String value = message.getText();
assertEquals(“Received!”, value);
```

En el siguiente ejemplo, se muestra cómo se inicia sesión en una página web de ejemplo.

```
//Pulsamos el botón para iniciar sesión en la web
driver.findElement(By.id("nav-link-yourAccount")).click();


//Buscamos el elemento para introducir el usuario y lo introducimos
WebElement elementUser;
elementUser = driver.findElement(By.id("ap_email"));
elementUser.sendKeys("aaaa@gmail.com");

//Buscamos el elemento para introducir la  contraseña y la introducimos
WebElement elementPassword;
elementPassword = driver.findElement(By.id("ap_password"));
elementPassword.sendKeys("prueba");

//Pulsamos botón para iniciar sesión
driver.findElement(By.id("signInSubmit-input")).click();
```

Ejercicio.

Supongamos que queremos probar la funcionalidad de un formulario de registro en el sitio web https://www.example.com/registro. El formulario tiene tres campos: "nombre", "correo electrónico" y "contraseña"[2]. Queremos asegurarnos de que el formulario se envíe correctamente cuando se rellenen todos los campos. En el caso de que todos los campos se hayan introducido y se envíe el formulario, se recibe un mensaje de confirmación (cuyo id es confirmation-message) con el mensaje “Tu formulario ha sido enviado correctamente”. Si se dejan campos en blanco, aparece un mensaje de error (con id error), cuyo mensaje es “Por favor, rellene todos los campos”.

Prepara varias pruebas funcionales utilizando los conceptos básicos vistos de Selenium.

[2] Estos son los ids de los tres campos.