

Objetivo

¡Hola! 🖐️

A continuación abordaremos cómo podemos realizar pruebas automáticas en sitios web utilizando Selenium como framework para ello.

En este encuentro **veremos cómo agregar la dependencia de Selenium a Visual Studio Code y cómo automatizar la apertura, el cierre, la navegación y manipulación de una ventana del navegador**. En encuentros posteriores, veremos cómo interactuar con los elementos de los sitios que estemos testeando.

¡Comencemos!

Verás que los ejercicios están marcados como **fundamentales** y **perfeccionamiento**. Concéntrate en **lograr la resolución de los fundamentales** para asegurar la comprensión de los conceptos abordados. Una vez completos, puedes abordar los de perfeccionamiento.

Configuración de Selenium en Visual Studio Code

Para agregar Selenium a Visual Studio Code, debes hacer lo siguiente:

1. Crea un proyecto nuevo de JAVA en Maven.
2. En la sección **<dependencies>** de tu archivo **pom.xml** agrega los siguientes códigos:

```

<modelVersion>4.0.0</modelVersion>

<groupId>com.egg</groupId>
<artifactId>selenium</artifactId>
<version>1</version>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>17</java.version>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <maven.compiler.target>${java.version}</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
  </dependency>
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.0.3</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.0.0-M8</version>
    </plugin>
  </plugins>
</build>
</project>

```

```
<dependency>
```

```
  <groupId>org.seleniumhq.selenium</groupId>
```

```
  <artifactId>selenium-java</artifactId>
```

```
  <version>3.141.59</version>
```

```
  </dependency>
```

```
<dependency>
```

```
  <groupId>io.github.bonigarcia</groupId>
```

```
  <artifactId>webdrivermanager</artifactId>
```

```
  <version>5.0.3</version>
```

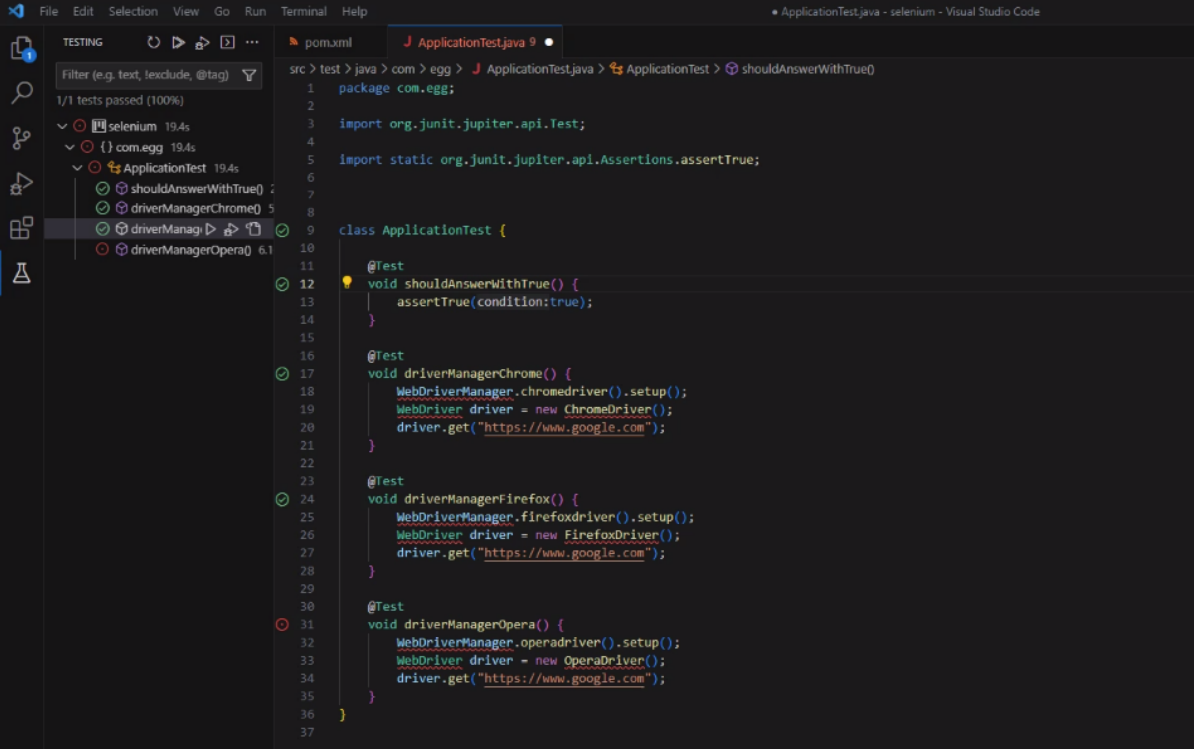
```
  <scope>test</scope>
```

```
</dependency>
```

Después de agregar esta dependencia a tu archivo pom.xml, Maven debería descargar automáticamente la biblioteca de Selenium y sus dependencias.

3. Guarda los cambios en tu archivo pom.xml

A partir de este momento, en tu archivo **.java** deberías ver lo siguiente:



The screenshot shows the Visual Studio Code interface with a Java file named `ApplicationTest.java` open. The file contains the following code:

```
src > test > java > com > egg > ApplicationTest.java
1 package com.egg;
2
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.assertTrue;
6
7
8
9 class ApplicationTest {
10
11     @Test
12     void shouldAnswerWithTrue() {
13         assertTrue(condition:true);
14     }
15
16     @Test
17     void driverManagerChrome() {
18         WebDriverManager.chromedriver().setup();
19         WebDriver driver = new ChromeDriver();
20         driver.get("https://www.google.com");
21     }
22
23     @Test
24     void driverManagerFirefox() {
25         WebDriverManager.firefoxdriver().setup();
26         WebDriver driver = new FirefoxDriver();
27         driver.get("https://www.google.com");
28     }
29
30     @Test
31     void driverManagerOpera() {
32         WebDriverManager.operadriver().setup();
33         WebDriver driver = new OperaDriver();
34         driver.get("https://www.google.com");
35     }
36 }
37
```

The left sidebar shows the 'TESTING' view with a tree structure indicating that all tests passed (100%). The tree structure is as follows:

- 1/1 tests passed (100%)
 - selenium 19.4s
 - com.egg 19.4s
 - ApplicationTest 19.4s
 - shouldAnswerWithTrue() 19.4s (passing)
 - driverManagerChrome() 19.4s (passing)
 - driverManagerFirefox() 19.4s (passing)
 - driverManagerOpera() 19.4s (passing)

En el caso que no puedas ver la clase correctamente, haz click derecho en el código y elige "Add Missing Imports" 🖱️

📁 Para que valides que tu archivo .xml y .java han sido creados correctamente, te compartimos [un ejemplo de cada uno](#) para que descargues y compares.

Con esto, ya tienes instalado Selenium en Visual Studio Code y estás listo para empezar a escribir tus scripts de prueba. En la siguiente sección, veremos en detalle cómo funciona Selenium.

🤔 ¿Te ha sonado familiar lo que acabas de hacer? Eso es porque ya hemos visto Maven en unidades anteriores lo que nos ha permitido hoy comprender cómo se automatizan pruebas. Las clases de este proyecto se han creado automáticamente bajo **ApplicationTest** como recordamos en este video.

https://youtu.be/9MHQbs1_sIU

Introducción a Selenium

¡Excelente! Ya tienes tu espacio de trabajo configurado y estás listo para comenzar a probar sitios web.

En grupo, **miren el siguiente video** que abordará:

- Cómo funciona Selenium
- Cómo nos ahorra tiempo al automatizar acciones en los navegadores
- Cómo se crean pruebas sencillas

! En el video verás cuáles son las funciones generales de Selenium con un sitio web como Wikipedia. Observarán que existen diversas maneras de instalar drivers y que se ejemplifica el uso de la herramienta utilizando acciones como *findElement*.

🧠 Concéntrate en entender el objetivo de Selenium y cómo éste interactúa con los navegadores. En este momento, **no necesitas replicar lo que ves en el video, iremos paso a paso** comprendiendo las diferentes acciones que puedes realizar con la herramienta e iremos profundizando a lo largo del curso.

<https://youtu.be/eISATKwvIS0>

Práctica I - Fundamental

🎯 Este ejercicio es de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

Ejercicio 1: Abrir el navegador y navegar hacia una página web

El primer paso en la automatización de la web es simplemente abrir una página web. Para este ejercicio, escribe un script que abra el navegador Chrome y vaya a la página principal de Google (u otra que prefieras).

🔑 A partir del minuto 2:24 en el video que has visto anteriormente tendrás una guía para poder lograrlo.

🔑 **Recurso necesario a partir de ahora:** esta es la [documentación oficial de Selenium](#). Puedes buscar los comandos/acciones que necesites para resolver los ejercicios siguientes desde la lupa de este sitio web.

¿Observas algo distinto en el video?

En este caso, la persona está instalando un driver para Chrome, pero nosotros esto lo hemos hecho en la configuración anterior. Por esto, para abrir e ir hacia una página web, debes reemplazar la totalidad de las líneas de código que comienzan (e incluyen) **String** y **System.setProperty** del video:

```
no usages
@Test
public void testWiki(){
    String driverPath = "/Users/navvir.zerpa/glb/WebAutomation copy/src/utils/chromedriver";
    System.setProperty("webdriver.chrome.driver", driverPath);
    |
}
```

Por lo siguiente:

```
WebDriverManager.chromedriver().setup();
```

El resto de las acciones puedes ejecutarlas de la misma forma que ves en el video. 😊

Práctica II - Fundamental

🎯 Este ejercicio es de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

Ejercicio 2: Manipular la ventana del navegador

Ahora que sabemos cómo abrir una página web, podemos empezar a interactuar con la ventana del navegador.

Para este ejercicio, escribe un script que abra la página principal de Google, maximice la ventana del navegador y luego cierre el navegador.

🔑 Prueba buscar la acciones que necesitas luego de escribir **driver**. usando palabras claves como "window" o "maximize". ¡Sólo mediante la práctica y experimentación podrás ganar experiencia y velocidad en los comandos!

Práctica III - Perfeccionamiento

✨ Este ejercicio es de perfeccionamiento. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto.

Ejercicio 3: Navegación hacia atrás y hacia delante

El WebDriver de Selenium también nos permite navegar hacia atrás y hacia delante en nuestro historial del navegador, al igual que cuando hacemos clic en los botones de atrás y adelante en el mismo.

Para este ejercicio, escribe un script que abra la página principal de Google, luego vaya a la página principal de OpenAI, luego navegue hacia atrás a Google, luego hacia delante a OpenAI, y finalmente cierre el navegador.

🔑 Palabra clave para encontrar la acción de navegar: **navigate**

Práctica III - Perfeccionamiento

✨ Este ejercicio es de perfeccionamiento. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto.

Ejercicio 3: Navegación hacia atrás y hacia delante

El WebDriver de Selenium también nos permite navegar hacia atrás y hacia delante en nuestro historial del navegador, al igual que cuando hacemos clic en los botones de atrás y adelante en el mismo.

Para este ejercicio, escribe un script que abra la página principal de Google, luego vaya a la página principal de OpenAI, luego navegue hacia atrás a Google, luego hacia delante a OpenAI, y finalmente cierre el navegador.

🔑 Palabra clave para encontrar la acción de navegar: **navigate**

Práctica Integradora - Perfeccionamiento

✨ Este ejercicio es de perfeccionamiento. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto.

Práctica Integradora

En este ejercicio, vamos a combinar varios de los conceptos que hemos cubierto hasta ahora: la apertura de un navegador, la navegación a varias páginas web, la manipulación del tamaño y la posición de la ventana del navegador y la navegación hacia atrás y hacia delante en el historial del navegador. **El objetivo del ejercicio es:** 1. Abrir el navegador y navegar a la página de inicio de Google. 2. Maximizar la ventana del navegador. 3. Navegar a la página de inicio de OpenAI. 4. Reducir la ventana del

navegador a la mitad de su tamaño y centrarla en la pantalla. 5. Navegar a la página de inicio de Wikipedia. 6. Cambiar el tamaño de la ventana del navegador a su tamaño original 7. Cerrar el navegador

En las próximas clases veremos como manipular elementos dentro de los sitios web.

¡Nos vemos pronto!

Objetivos del encuentro

¡Hola!

El objetivo de este encuentro es que puedas aprender cómo identificar elementos en una página web para poder realizar acciones en pruebas automatizadas.

Repasaremos cómo encontrar elementos en el HTML y cómo trasladar éstos a Selenium para iniciar pruebas.

¡Comencemos!

Verás que los ejercicios están marcados como **fundamentales** y **perfeccionamiento**. Concéntrate en **lograr la resolución de los fundamentales** para asegurar la comprensión de los conceptos abordados. Una vez completos, puedes abordar los de perfeccionamiento.

Inspección de páginas web

Para poder comenzar a automatizar, necesitas comprender qué elementos componen una página web y cómo están estructurados.

En el siguiente video repasamos cómo hacerlo mediante la inspección en el navegador. 🖱️

[Inspeccionar elementos HTML](#)

Elementos Web

Ahora que sabes cómo encontrar el código HTML y CSS de las páginas web, veremos cómo incorporar estos a las pruebas automatizadas.

Primero, observa como en este ejemplo podemos reconocer e indicarle a nuestro proyecto de testing cuáles son los elementos que queremos utilizar para automatizar.


El objetivo de este video es que puedas entender cómo están organizados jerárquicamente los locators (localizadores) basándose en su especificidad. No te preocupes, veremos cómo trabajar con ellos de forma puntual en breve.

https://youtu.be/WGX_MprLOMc

Identificación por ID

A continuación, veremos cómo localizar elementos por su ID, el elemento más específico y preferible a la hora de automatizar.

<https://youtu.be/-yIGaKFJPxE>

 Este ejercicio es de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

En grupo, encuentren al menos 4 páginas web que contengan atributos del tipo ID y utilicen las acciones **getElement** y **sendKeys** para identificarlos y llenar con datos en caso que el elemento lo permita.

Identificar por Name

Excelente, ahora veamos cómo hacerlo con el atributo **Name**.

<https://youtu.be/kJrrgCPzXbM>

🎯 Este ejercicio es de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

Localiza al menos 3 elementos distintos por Name y envía un valor o haz click en ellos según los elementos lo permitan.

Identificar por LinkText

Ahora veremos cómo identificar links por el texto asociado a ellos. También veremos la diferencia entre **LinkText** y **partialLinkText**.

<https://youtu.be/LI8YnC5L9e0>

🎯 Este ejercicio es de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

Toma las páginas que has utilizado anteriormente y localiza al menos un link en cada una de ellas y automatiza la acción **click**.

Integración

✨ Este ejercicio es de perfeccionamiento. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto.

¡Excelente! Has aprendido cómo identificar elementos por ID, Name y LinkText.

Ahora crea una prueba que:

1. Abra el navegador en Chrome.
2. Vaya a una URL de tu preferencia.
3. Identifique un elemento por ID.
4. Vaya a otra URL.
5. Identifique un elemento por LinkText.
6. Haga click en ese elemento.
7. Cierre el navegador.

Introducción

¡Hola!

Ahora que sabes cómo identificar elementos web por su especificidad, vamos a trabajar localizando elementos web por atributos inespecíficos.

En este caso, veremos cómo buscar elementos por className y tagName.

¡Comencemos!

ClassName

Una forma de localizar elementos web es a través del nombre de la clase a la que pertenecen.

En el siguiente video, te mostramos cómo hacerlo.

[Video ClassName](#)

Práctica className

🎯 Estos ejercicios son de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

¡Es momento de que apliques lo que aprendiste!

1. Encuentra todos los botones que existan en una página web de tu preferencia y cuéntalos.
2. Encuentra cuántas imágenes tiene [este producto](#).
3. Encuentra todos los productos en este [link](#) y cuenta cuáles pertenecen a Star Wars.

TagName

En esta instancia veremos cómo encontrar elementos por su etiqueta HTML. Mira el siguiente video para aprender cómo hacerlo:

[Video TagName](#)

Práctica TagName

🎯 Estos ejercicios son de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

1. Identifica cuántos encabezados de jerarquía 2 hay en este [link](#).
2. Encuentra y haz click en el primer link de esa página.
3. Identifica cuántas imágenes hay en este [link](#).
4. Encuentra el primer botón de la página anterior y haz click en el.

Práctica Integradora

🎯 Estos ejercicios son de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

Ingresa a tu proveedor de mail (Gmail, Hotmail, etc.) e introduce tu usuario y contraseña para loguearte.

Ingresa a Wikipedia, busca un elemento químico de tu preferencia y haz click en el primer enlace. Luego, cierra el navegador. ✨ **Estos ejercicios son de tipo complementario. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto, pero no son obligatorios.** 1. Encuentra todos los enlaces que contengan el texto "Descargar" y haz clic en el segundo: [LINK](#) 2. Encuentra todos los párrafos y cuenta cuántas palabras tiene el primer párrafo: [LINK](#) 3. Encuentra un elemento por su clase CSS y muestra su texto: [LINK](#)

4. Ingresa a este [link](#), haz click en "Pen Tablets" y encuentra un menú dropdown para luego seleccionar una opción por su texto visible.

Introducción a Selectores CSS

¡Hola!

Hoy veremos cómo localizar elementos por selectores CSS.

Te compartimos el siguiente video como introducción al tema. Luego, entraremos en detalle a cada uno de ellos.

[VIDEO](#)

Tag + ID

[VIDEO](#)

Tag + className

[VIDEO](#)

Tag + Attribute

[VIDEO](#)

Tag + className + Attribute

[VIDEO](#)

Verificación de Elementos

[VIDEO](#)

Práctica Selectores CSS

Te compartimos la imagen sobre la que se ha explicado la teoría para que la usen de ayuda memoria al hacer los ejercicios.

Customized Locators

CSS SELECTORS

XPATH

TAG + ID
TAG + CLASSNAME
TAG + ATTRIBUTE
TAG + CLASSNAME + ATTRIBUTE

TAG + ID → tag#id

```
driver.findElement(By.cssSelector("input#email")).sendKeys("abcd@test.com");  
driver.findElement(By.cssSelector("#email")).sendKeys("abcd@test.com");
```

TAG + CLASS → tag.clase

```
driver.findElement(By.cssSelector("input.inputtext")).sendKeys("abcd@test.com");  
driver.findElement(By.cssSelector(".inputtext")).sendKeys("abcd@test.com");
```

TAG + ATTRIBUTE → tag[nombreatributo=valoratributo]

```
driver.findElement(By.cssSelector("input[placeholder=Email or phone number]")).sendKeys("abcd@test.com");  
driver.findElement(By.cssSelector("[placeholder=Email or phone number]")).sendKeys("abcd@test.com");
```

TAG + CLASS + ATTRIBUTE → tag.class[nombreatributo=valoratributo]

```
driver.findElement(By.cssSelector("input.inputtext[data-testid=royal_email]")).sendKeys("abcd@test.com");
```

🎯 Estos ejercicios son de tipo **fundamental**, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

1. Etiqueta + ID: Encuentra el cuadro de búsqueda en Google por su etiqueta y ID.

2. Etiqueta + Clase: Encuentra y haz clic en el primer enlace en la barra de navegación en la página de inicio de GitHub. **3. Etiqueta + Atributo:** Encuentra el botón "Sign Up" en la página de inicio de Twitter.

4. Etiqueta + Clase + Atributo: Encuentra el enlace "Forgot password?" en la página de inicio de sesión de LinkedIn. **5. Etiqueta + Clase:** Encuentra y haz clic en el primer artículo "Trending" en Medium. **6. Etiqueta + Clase:** Encuentra y haz clic en el enlace "Contact" en el pie de página del sitio Stack Overflow. **7. Etiqueta + ID:** Encuentra y muestra la descripción del primer producto en la página de inicio de Amazon.

8. Etiqueta + Atributo: Encuentra y muestra el título del primer video en la página de inicio de YouTube.

✨ Estos ejercicios son de tipo **complementario**. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto, pero **no son obligatorios**.

1. Inicia sesión en LinkedIn y navega hasta tu perfil.
2. Navega a una lista de reproducción en YouTube y muestra el título de los primeros 5 videos.
3. Navega a Amazon, realiza una búsqueda, luego filtra los resultados por una categoría específica.
4. Navega a una página de Wikipedia, selecciona una sección y muestra las referencias utilizadas en esa sección.
5. Navega a Pixabay, realiza una búsqueda de imágenes y explora la primera galería.
6. Navega a Best Buy, busca un producto y verifica si su precio está por debajo de un valor determinado.
7. Navega a FlightStats, rastrea un vuelo por su número y muestra el estado.
8. Navega a AllRecipes, busca una receta y muestra los ingredientes.

Práctica Integradora de Selectores

✨ Estos ejercicios son de tipo **complementario**. Esto quiere decir que te ayudará a avanzar en profundidad en el tema visto, pero **no son obligatorios**.

1. Navegar a la página de Wikipedia de una ciudad, leer el primer párrafo y acceder a la página de su país.

2. Búsqueda de Empleo en LinkedIn:

- Navega a LinkedIn.
- Busca empleos de "Desarrollador de software" en una ubicación específica.
- Enumera las 5 primeras ofertas de trabajo, incluyendo el título y la empresa.

3. Iniciar sesión en Instagram, acceder al perfil y obtener el número de publicaciones, seguidores y seguidos:

selenium 4 tiene conflictos con wait

Introducción Waits

Hoy veremos qué son los **waits (esperas)** y por qué son importantes.

En el recorrido de un usuario, cuando una persona ingresa a una página web, seguramente deba esperar algunos momentos para que esta termine de cargar antes de interactuar con algún elemento, como por ejemplo, hacer click.

Nosotros, como testers, debemos emular estas condiciones para generar pruebas confiables. Para esto, podemos usar algunos comandos en Selenium que replicarán los tiempos de espera.

¡Vamos a ver cómo trabajar con ellos!

Tipos de Waits

Existen **tres tipos de waits**:

1. Implicit
2. Explicit
3. Fluent

Vamos a verlos en este video 

[Video](#)

Implicit Waits

Algunas características de los Implicit Waits:

1. **Aplican para todos los elementos** de la página web.
2. Le indica al WebDriver que tiene que esperar X tiempo antes de ejecutar el próximo comando.
3. Una vez que está configurado, **aplica para todo el script** de automation.
4. El implicit wait le indica al WebDriver que espere una cantidad determinada de tiempo antes de lanzar una excepción de "Elemento no encontrado" (**NoSuchElementException**) si el elemento no está disponible inmediatamente.

Es decir, si el elemento que nosotros estamos buscando NO está disponible (no cargó) en la cantidad de tiempo que nosotros indicamos, entonces tendremos el mensaje de **NoSuchElementException**.

Para aplicar un implicit wait, debemos agregarlo de esta manera:

```
WebDriver driver = new ChromeDriver(); // Establecer la espera
implicita de 10 segundos
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("https://www.lapaginaquequieras.com");

WebElement element = driver.findElement(By.id("unElemento"));

// Continuar con las acciones en el elemento...
```

En este ejemplo, si el elemento con el ID "**unElemento**" no está disponible de inmediato, el WebDriver esperará hasta 10 segundos para que aparezca. Si el elemento se encuentra disponible antes de que transcurran los 10 segundos, el código continuará ejecutándose sin esperar todo el tiempo establecido.

Práctica Implicit Waits

🎯 Estos ejercicios son de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

1. Esperar 15 segundos a que aparezca el cuadro de búsqueda en la página principal de Twitter.
2. Esperar un minuto a que se cargue el botón "Load more" en Medium y hacer clic en él.
3. Esperar 45 segundos a que se cargue la sección de comentarios de un video que quieras en YouTube y contar la cantidad de comentarios.
4. Esperar a que se carguen los resultados de búsqueda en Amazon y obtener el precio del primer producto.

Explicit Waits

Algunas características de los Explicit Waits:

- No aplican a todos los elementos de la página web, solo a los específicos para los cuales están configurados.
- Permiten esperar hasta que se cumpla una cierta condición, como la visibilidad de un elemento o la actualización de un atributo.
- No se aplican a todo el script de automatización, solo donde los defines.
- Puedes especificar distintas condiciones y tiempos de espera para diferentes elementos o acciones.

Aquí tienes un ejemplo de cómo aplicar un explicit wait:

```
WebDriver driver = new ChromeDriver();
driver.get("https://www.lapaginaquequieras.com");
WebDriverWait wait = new WebDriverWait(driver, 10); // Esperar hasta 10 segundos
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("un
Elemento"))); // Continuar con las acciones en el elemento...
```

En este ejemplo, el WebDriver esperará hasta 10 segundos para que el elemento con el ID **"unElemento"** sea visible. Si el elemento se vuelve visible antes de que

transcurran los 10 segundos, el código continuará ejecutándose sin esperar todo el tiempo establecido. Si el elemento no se vuelve visible en 10 segundos, se lanzará una excepción de tiempo de espera (**TimeoutException**).

Las esperas explícitas son especialmente útiles cuando trabajas con elementos que pueden tardar en cargarse o cambiar de estado, y te permiten especificar condiciones complejas y tiempos de espera personalizados. A diferencia de las esperas implícitas, las explícitas solo se aplican a los elementos y condiciones específicas para los cuales están configuradas, lo que te brinda un mayor control sobre la sincronización de tu script.

Expected Conditions

Habrás notado que el wait en este caso tiene la sintaxis: **Espera N segundos + hasta que + ocurra algo**

Este "**ocurra algo**" es lo que en el código anterior vemos como

ExpectedConditions.ALGO

Aquí te compartimos una serie de métodos que definen ese "algo", junto con su descripción:

Las condiciones esperadas (**ExpectedConditions**) en Selenium son una serie de métodos predefinidos utilizados junto con **WebDriverWait** para esperar a que ocurra una cierta condición antes de continuar con la ejecución del script. Hay varias condiciones esperadas disponibles que puedes usar, dependiendo de lo que necesites en tu caso de prueba:

1. **elementToBeClickable(By locator)**
: Espera hasta que un elemento sea clickeable.
2. **elementToBeClickable(WebElement element)**
: Espera hasta que un elemento específico sea clickeable.
3. **elementToBeSelected(By locator)**
: Espera hasta que un elemento sea seleccionado.

4. `elementToBeSelected(WebElement element)`
: Espera hasta que un elemento específico sea seleccionado.
5. `frameToBeAvailableAndSwitchToIt(By locator)`
: Espera hasta que esté disponible un marco (frame) y cambia a él.
6. `invisibilityOf(WebElement element)`
: Espera hasta que un elemento específico sea invisible.
7. `invisibilityOfElementLocated(By locator)`
: Espera hasta que un elemento sea invisible.
8. `presenceOfAllElementsLocatedBy(By locator)`
: Espera hasta que estén presentes todos los elementos encontrados por el localizador.
9. `presenceOfElementLocated(By locator)`
: Espera hasta que un elemento esté presente en el DOM.
10. `stalenessOf(WebElement element)`
: Espera hasta que un elemento específico ya no esté presente en el DOM.
11. `textToBePresentInElement(By locator, String text)`
: Espera hasta que el texto especificado esté presente en el elemento encontrado por el localizador.
12. `textToBePresentInElementLocated(By locator, String text)`
: Espera hasta que el texto especificado esté presente en el elemento especificado por el localizador.
13. `textToBePresentInElementValue(By locator, String text)`
: Espera hasta que el texto especificado esté presente en el valor del atributo del elemento encontrado por el localizador.
14. `visibilityOf(WebElement element)`
: Espera hasta que un elemento específico sea visible.
15. `visibilityOfAllElements(List<WebElement> elements)`
: Espera hasta que todos los elementos en la lista estén visibles.
16. `visibilityOfAllElementsLocatedBy(By locator)`
: Espera hasta que todos los elementos encontrados por el localizador estén visibles.
17. `visibilityOfElementLocated(By locator)`
: Espera hasta que un elemento esté visible.

Estas condiciones esperadas ofrecen una gran flexibilidad y permiten manejar una variedad de situaciones en las que podrías necesitar esperar a que algo ocurra en tu página antes de continuar.

Práctica Explicit Waits

🎯 Estos ejercicios son de tipo **fundamental**, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema.

- **Esperar a que aparezca el cuadro de búsqueda en Google**
- **Esperar a que el botón "Login" sea clickeable en GitHub**
- **Esperar a que se cargue la lista de categorías en Wikipedia**
- **Esperar a que esté disponible el menú desplegable de idioma en la página de Facebook**

Fluent Waits

Características de Fluent Wait

- **Personalización de Tiempo de Espera:** Puedes definir el tiempo máximo que deseas esperar una condición y la frecuencia con la que quieres verificar la condición.
- **Ignorar Excepciones Específicas:** Puedes configurarlo para ignorar excepciones específicas mientras espera que se cumpla una condición, como `NoSuchElementException`.
- **Función Personalizada:** Puedes utilizar una función lambda o una implementación de la interfaz `Function` para definir la condición de espera.
- **Aplicación en Elemento Específico:** A diferencia de la espera implícita, que se aplica a todo el controlador, Fluent Wait se puede aplicar a condiciones muy específicas.

Ejemplo de Uso de Fluent Wait

Supongamos que quieres esperar a que aparezca un botón de inicio de sesión en una página web, y quieres verificar su presencia cada medio segundo durante un máximo de 10 segundos. Aquí tienes un código de ejemplo:

```
WebDriver driver = new ChromeDriver();
driver.get("https://www.example.com");

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(Duration.ofSeconds(10))

    // Tiempo máximo de espera

    .pollingEvery(Duration.ofMillis(500))

    // Frecuencia de verificación

    .ignoring(NoSuchElementException.class);

// Ignorar esta excepción durante la espera WebElement loginButton
= wait.until(new Function<WebDriver, WebElement>() { public
WebElement apply(WebDriver driver) { return
driver.findElement(By.id("loginButton"));

    // Condición de espera } });

loginButton.click(); driver.quit();
```

En este ejemplo, si el elemento con el ID "loginButton" no está disponible de inmediato, WebDriver esperará hasta 10 segundos, verificando su presencia cada 500 milisegundos. Si el elemento se encuentra disponible antes de que transcurran los 10 segundos, el código continuará ejecutándose sin esperar todo el tiempo establecido.

Métodos para Fluent Wait

Fluent Wait en Selenium permite una gran personalización a través de varios métodos que se pueden utilizar. A continuación, te muestro algunos de los métodos más comunes:

1. **withTimeout(Duration duration)**: Establece la cantidad máxima de tiempo para esperar una condición. La espera terminará y lanzará una excepción si la condición no se cumple dentro del período de tiempo especificado.
2. **pollingEvery(Duration duration)**: Establece la frecuencia con la que se debe verificar la condición. Por ejemplo, si la configuras en 500 milisegundos, la condición se verificará cada 500 milisegundos.
3. **ignoring(Class<? extends Throwable> exceptionType)**: Le dice a Fluent Wait que ignore ciertas excepciones mientras espera que se cumpla una condición. Esto es útil si esperas que una excepción ocurra comúnmente y no quieres que termine la espera.
4. **until(Function<? super T, V> isTrue)**: Este método acepta una instancia de **Function** que debe devolver un valor diferente de **null** o **false** si la condición se cumple. La espera continuará hasta que se cumpla esta condición o se alcance el tiempo máximo de espera.
5. **withMessage(String message)**: Este método permite establecer un mensaje personalizado que se incluirá en la excepción si se alcanza el tiempo de espera sin que se cumpla la condición.
6. **until(Predicate<T> isTrue)**: Similar a **until(Function<? super T, V> isTrue)**, pero acepta un **Predicate** y continúa la espera hasta que el predicado devuelva **true**.

Aquí tienes un ejemplo que utiliza varios de estos métodos:

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(Duration.ofSeconds(30)) // Tiempo máximo de espera
    .pollingEvery(Duration.ofSeconds(5)) // Frecuencia de verificación
    .ignoring(NoSuchElementException.class) // Ignorar esta excepción
    .withMessage("Elemento no encontrado en el tiempo especificado");
// Mensaje personalizado WebElement element = wait.until(new
Function<WebDriver, WebElement>() { public WebElement
apply(WebDriver driver) { return
driver.findElement(By.id("someElement")); } });
```


Práctica Fluent Waits

🕒 Estos ejercicios son de tipo fundamental, esto quiere decir que es lo mínimo que necesitas resolver hoy para asegurar la comprensión del tema. 1. Esperar a que aparezca el logo en la página principal de Wikipedia 2. Esperar a que se cargue la sección de imágenes destacadas en Unsplash 3. Esperar a que se cargue el banner principal en la página de tecnología de BBC 4. Esperar a que se cargue la sección de "Top Stories" en la página de noticias de CNN Estos ejercicios te ayudarán a comprender cómo aplicar Fluent Wait en situaciones reales sin la necesidad de interactuar con páginas que requieran inicio de sesión o creación de una cuenta.

Ejercicio 1: Búsqueda en Google con Implicit Wait

Objetivo: Realizar una búsqueda en Google y hacer clic en el primer enlace de resultados, utilizando Implicit Wait.

Ejercicio 2: Esperar por la Imagen Principal en la Página de la BBC con Explicit Wait

Objetivo: Navegar a la página de tecnología de la BBC, esperar a que se cargue la imagen principal utilizando Explicit Wait, y luego hacer clic en ella.

Ejercicio 3: Interacción con Wikipedia usando Fluent Wait

Objetivo: Navegar a la página de búsqueda en Wikipedia, buscar un término y esperar a que se cargue la página de resultados utilizando Fluent Wait.

Introducción a Assertions

Una "assertion" (afirmación en español) es **una declaración en programación que permite verificar si una condición particular es verdadera o falsa**. Si la afirmación se evalúa como verdadera, el programa continúa su ejecución normalmente. Si se evalúa como falsa, el programa generalmente arroja una excepción y se detiene, o bien maneja el fallo de alguna otra manera. Por ejemplo, en el contexto de pruebas automatizadas, las afirmaciones **se utilizan para validar si una operación en un navegador web se ha realizado correctamente**. Si estás comprobando que un texto específico está presente en una página web, puedes usar una afirmación para comparar el texto esperado con el texto real en la página. En Java, un **ejemplo** de una afirmación en un marco de pruebas como JUnit podría ser: `assertEquals("Texto esperado", elementoWeb.getText());`

Si el texto del `elementoWeb` no coincide con "Texto esperado", esta afirmación fallará, y la prueba se marcará como fallida. Mira este video para conocer más sobre Assertions

[Video Assertions](#)

Tipos de Assertions

En el contexto de las pruebas en Java, se suelen distinguir dos tipos de afirmaciones: **Hard Assertions** y **Soft Assertions**. Estos términos se refieren a cómo se maneja una afirmación fallida. **Hard Assertions**

Estas son las afirmaciones tradicionales y se encuentran en la mayoría de los frameworks de pruebas como JUnit. Si una Hard Assertion falla, la ejecución de la prueba se detiene inmediatamente, y el código siguiente en esa prueba no se ejecutará. Esto significa que si tienes múltiples afirmaciones en tu prueba y la primera falla, las siguientes no se evaluarán.

Ejemplo en JUnit: `assertEquals(5, suma(2, 3)); // Si falla aquí, la prueba se detiene`
`assertTrue(isElementPresent(elemento));`

Soft Assertions

A diferencia de las Hard Assertions, las Soft Assertions **no detienen la ejecución de la prueba si una afirmación falla**. Esto es útil si deseas que una prueba continúe incluso si una de las condiciones intermedias no se cumple, lo que te permite recopilar información sobre múltiples fallas en una sola ejecución de la prueba.

En JUnit, no hay una clase `SoftAssert` integrada, pero aún puedes lograr un comportamiento similar a las Soft Assertions usando otras estrategias, como recolectar errores en una lista y luego verificarlos todos al final.

Ejemplo en JUnit:

```
import static
org.junit.jupiter.api.Assertions.assertEquals; import
org.junit.jupiter.api.Test; import java.util.ArrayList;
import java.util.List; public class MySoftAssertTest {
@Test public void testWithSoftAssertions() {
List<Throwable> errors = new ArrayList<>(); try {
assertEquals(5, suma(2, 2)); // Si falla aquí, se añade a
errores } catch (Throwable t) { errors.add(t); } try {
assertEquals("Texto esperado", "Texto actual"); // Si
falla aquí, se añade a errores } catch (Throwable t) {
errors.add(t); } // Lanza todas las afirmaciones fallidas
al final if (!errors.isEmpty()) { errors.forEach(error ->
System.err.println(error.getMessage())); throw
errors.get(0); // Lanza la primera excepción para marcar
la prueba como fallida } } // Método auxiliar para el
ejemplo private int suma(int a, int b) { return a + b; }
}
```

Métodos para Assertions

Ahora que sabemos cómo comprobar un valor que determinamos, necesitamos conocer qué tipos de verificaciones podemos hacer. Esto es:

```
Assert.tipodeverificacionquequiero(esperado, real)
```

Dentro de los métodos que podemos usar en reemplazo de "tipodeverificacionquequiero", tenemos:

1. **assertEquals(expected, actual)**: Comprueba si dos valores son iguales.
2. **assertNotEquals(expected, actual)**: Comprueba si dos valores no son iguales.
3. **assertTrue(condition)**: Comprueba si una condición es verdadera.
4. **assertFalse(condition)**: Comprueba si una condición es falsa.
5. **assertNull(value)**: Comprueba si un objeto es nulo.
6. **assertNotNull(value)**: Comprueba si un objeto no es nulo.
7. **assertSame(expected, actual)**: Comprueba si dos referencias de objetos apuntan al mismo objeto.
8. **assertNotSame(expected, actual)**: Comprueba si dos referencias de objetos no apuntan al mismo objeto.
9. **assertArrayEquals(expectedArray, actualArray)**: Comprueba si dos matrices son iguales en términos de longitud y contenido.
10. **assertIterableEquals(expectedIterable, actualIterable)**: Comprueba si dos iterables son iguales.
11. **assertThrows(expectedType, executable)**: Comprueba si una excepción del tipo esperado es lanzada por el código ejecutable.
12. **fail()**: Utilizado para marcar una prueba como fallida manualmente.

Ejemplo de una prueba con Selenium que utiliza una afirmación (assertion) para verificar el título de una página web. Supongamos que quieres comprobar que el título de la página de inicio de Google es correcto.

```
import static
org.junit.jupiter.api.Assertions.assertEquals; import
org.junit.jupiter.api.Test; import
org.openqa.selenium.WebDriver; import
```

```
org.openqa.selenium.chrome.ChromeDriver; public class
GoogleTitleTest { @Test public void testGoogleTitle() {
// Inicializar el navegador Chrome WebDriver driver = new
ChromeDriver(); // Ir a la página de inicio de Google
driver.get("https://www.google.com"); // Obtener el
título de la página String title = driver.getTitle(); //
Asegurar que el título sea el esperado
assertEquals("Google", title); // Cerrar el navegador
driver.quit(); } }
```

En este ejemplo, la prueba abrirá Chrome, navegará hasta la página de inicio de Google, y luego utilizará **assertEquals** para asegurarse de que el título de la página es "Google". Si el título es correcto, la prueba pasará. Si el título es cualquier otra cosa, la prueba fallará y JUnit mostrará un mensaje indicando la discrepancia entre el valor esperado ("Google") y el valor real.

Práctica assertEquals

Te compartimos tres ejercicios prácticos que serán útiles para aprender a escribir pruebas en Selenium utilizando `assertEquals`. Estos ejercicios se centran en verificar diferentes aspectos de una página web.

En el primero te dejamos una pequeña ayuda para que comprendas la lógica.

Ejercicio 1: Verificar el Título de una Página

Objetivo: Verificar que el

título de la página principal de Wikipedia sea correcto. @Test public void testWikipediaTitle() { WebDriver driver = new ChromeDriver(); driver.get("https://www.wikipedia.org"); //buscar elemento title y guardar como variable de texto; //assertEquals("Wikipedia", variable de texto);

```
driver.quit(); }
```

Ejercicio 2: Verificar el Texto de un Botón Objetivo:

Verificar que el texto del botón "Buscar con Google" en la página principal de Google sea correcto. **Ejercicio 3: Verificar la URL de un Enlace**

Objetivo: Verificar que el enlace a la página de inicio de sesión en Twitter tenga la URL correcta.

Verificación de resultado en consola

- **Si la Prueba Pasa:** Por lo general, no verás un mensaje detallado en la consola. Muchos IDEs (como Visual Studio Code con plugins adecuados) mostrarán una marca de verificación verde junto a la prueba para indicar que ha pasado. En la consola, es posible que veas un resumen de cuántas pruebas se ejecutaron y cuántas pasaron.
- **Si la Prueba Falla:** Verás un mensaje en la consola que indica qué falló. Esto incluirá detalles sobre cuál era el valor esperado y cuál era el valor real. Por ejemplo, podrías ver algo así:

```
lessCopy codeorg.junit.ComparisonFailure:
expected:<[Wikipedia]> but was:<[WikipediA]> at
org.junit.Assert.assertEquals(Assert.java:123) at
com.example.tests.MyTest.testWikipediaTitle(MyTest.java:1
5)
```

Este mensaje te dice que la prueba esperaba el valor "Wikipedia", pero recibió "WikipediA" en su lugar. También te muestra la línea de código donde falló la prueba.

Práctica assertEquals

Ahora practicarás `assertNotEquals` para verificar que ciertos valores no sean iguales en pruebas con Selenium. Estos pueden ser útiles para validar que ciertos elementos o propiedades de una página no tengan un valor particular. **Ejercicio 1: Verificar que el título de una página no sea una cadena vacía** Objetivo: Asegurarse de que el título de la página de inicio de YouTube no sea una cadena vacía. **Ejercicio 2: Verificar que el texto de un encabezado no sea un valor particular** Objetivo: Verificar que el texto del encabezado principal en la página de Wikipedia en inglés no sea "Enciclopedia Libre". **Ejercicio 3: Verificar que el texto del botón de Inicio de Sesión no sea "Iniciar Sesión"** Objetivo: Verificar que el texto del botón de inicio de sesión en la página de inicio de Twitter no sea "Iniciar Sesión" (suponiendo que el texto correcto sea diferente, como "Log in"). Este ejercicio verifica que el texto del botón de inicio de sesión en Twitter no sea una traducción incorrecta o un valor inesperado. Si la página de inicio de Twitter utiliza un texto diferente para el botón de inicio de sesión, como "Log in", esta prueba asegurará que el texto incorrecto "Iniciar Sesión" no esté presente.

Práctica assertTrue

En estos ejercicios deberás usar el método `assertTrue` para realizar las siguientes verificaciones.

Ejercicio 1: Verificar que la página de inicio de Google esté cargada. **Ejercicio 2:** Verificar que un producto esté disponible en Amazon. **Ejercicio 3:** Verificar que un enlace a "Contacto" exista en un sitio web de ejemplo. *Tip: deberás usar la expresión booleana para cada uno de los ejercicios y las acciones `.isDisplayed()`, `.isEnabled()` y `.size()` para cada ejercicio respectivamente.*

Práctica assertFalse

Ejercicio 1: Verificar que el botón de búsqueda está deshabilitado en Google sin texto en la barra de búsqueda

Ejercicio 2: Verificar que el checkbox "Recuérdame" no está marcado en la página de inicio de sesión de GitHub

Ejercicio 3: Verificar que una imagen específica no está visible en la página de Wikipedia (puedes elegir una imagen que sepas que no esté en la página) *Tip: deberás utilizar las acciones `.isEnabled()`, `is.Selected()` y `.size()` respectivamente*

Práctica assertNull

Ejercicio 1: Verificar que un elemento inexistente en una página no se encuentre (ejemplo, en la página de Google). **Ejercicio 2:** Verificar que una imagen sin atributo "alt" no tenga dicho atributo (puedes elegir cualquier URL). **Ejercicio 3:** Verificar que un campo de texto no tenga valor por defecto (puedes usar el search box de Wikipedia o los campos de inicio de sesión en Facebook). *Tip: para el primer ejercicio deberás utilizar `try` y `catch` dentro de tu código.*

Parte 1

Ahora tendrás que integrar los assertions que has visto y practicado en scripts completos. Verás que combinan partes de ejercicios que ya has hecho.

Ejercicio 1: Verificar que el botón de búsqueda está deshabilitado en Google sin texto en la barra de búsqueda.

Ejercicio 2: Verificar que el checkbox "Recuérdame" no está marcado en la página de inicio de sesión de GitHub **Ejercicio 3:** Verificar el título de la página de Wikipedia y la ausencia de una imagen específica

Parte 2

Ejercicio 4: Verificar la presencia de la barra de búsqueda en Bing y que la sugerencia no aparezca inicialmente

Ejercicio 5: Verificar la presencia del logotipo de Mozilla y el enlace de "Technology" en la barra de navegación.

Parte 3

Desafío complementario: verificar la disponibilidad de una posición de "Software Developer" y la ausencia de un trabajo de Pastelero Profesional en la página de empleos de GitHub.

assertSame, assertEquals, assertIterableEquals, assertThrows

assertSame

Ejemplo: Comprobar que dos referencias de objetos apuntan al mismo objeto.

```
@Testpublic void testSameObject() {  
    WebDriver driver = new ChromeDriver();  
    WebElement firstElement =  
driver.findElement(By.id("someID"));  
    WebElement secondElement =  
driver.findElement(By.id("someID"));  
    assertEquals(firstElement, secondElement);  
    driver.quit();  
}
```

assertEquals

Ejemplo: Comprobar si dos matrices son iguales en términos de longitud y contenido.

```
@Testpublic void testArrayEquality() {  
    int[] expectedArray = {1, 2, 3};  
    int[] actualArray = {1, 2, 3};  
    assertEquals(expectedArray, actualArray);  
}
```

assertIterableEquals

Ejemplo: Comprobar si dos iterables son iguales.

```
@Testpublic void testIterableEquality() { List<String>  
expectedList = Arrays.asList("one", "two", "three"); List<String>
```

```
actualList = Arrays.asList("one", "two", "three");  
assertIterableEquals(expectedList, actualList); }
```

assertThrows

Ejemplo: Comprobar si una excepción del tipo esperado es lanzada.

```
@Test  
public void testExceptionThrown() {  
    assertThrows(NoSuchElementException.class, () -> {  
        WebDriver driver = new ChromeDriver();  
        driver.findElement(By.id("nonExistentElement"));  
        driver.quit();  
    });  
}
```

Práctica

Ejercicio 1: Verificar el título de la página de Wikipedia y el logo

En este ejercicio combinarás la localización de elementos mediante ID, la utilización de assertions para verificar el título de la página y un Implicit Wait para asegurar que la página se haya cargado correctamente.

Ejercicio 2: Comprobar la existencia de un enlace y realizar una búsqueda en Google

Para este ejercicio utilizarás un Explicit Wait para esperar a que el elemento link esté presente, la ejecución de una acción (enviar una búsqueda) y la verificación del resultado utilizando una aserción.

Práctica

Ejercicio 3: Verificar el cambio de idioma en la página de Wikipedia

En este ejercicio deberás emplear un implicit wait, una acción para hacer click y un assertion de igualdad para comparar un tipo de texto que ha sido traducido al español. Recomendamos probar con el título de la página.

Ejercicio 4: Verificar la función de búsqueda en YouTube

Aquí deberás usar un explicit wait que espere a que el resultado de una búsqueda se haga presente y un assert que indique que sí hay elementos presentes luego de la búsqueda. No olvides la acción de .sendKeys.

Introducción a Page Object Model

En este encuentro verás qué es el Page Object Model y cómo es utilizado en automation para escalar nuestros tests.

https://youtu.be/M6Xu_5YEzVs

Primeros pasos con POM

Actividades

1. Discutan en equipo por qué es necesario estructurar sus tests en una POM.
2. Revisen todos los ejercicios que hicieron sobre la página Wikipedia y recopílenlos en un solo documento .java

Plantilla POM

Actividad

Esquematicen cómo podría ser una posible POM para el sitio de Wikipedia.

Pueden utilizar [Excalidraw](#) como pizarra para ayudarse y compartir pantalla.

Recursos que los pueden ayudar a realizar la actividad:

- [Documentación Oficial de Selenium sobre POMs](#)
- Recordar que los POMs se basan en tener **una clase para todos los elementos web y sus métodos y otra clase para todos los métodos de testing.**
- [Video corto sobre POM](#) con posibilidad de activar subtítulos en español.

Práctica POM I

En los próximos ejercicios, deberán aplicar POM para distintos tests de Wikipedia. Les damos un ejemplo de lo que deberán hacer:

Ejemplo: Navegar a la Página Principal de Wikipedia

Objetivo: Crear una clase de página para la página principal de Wikipedia y una prueba para navegar a la página. **Archivo donde crearán las clases:**

```
WikipediaHomePage.java public class WikipediaHomePage { private
WebDriver driver; public WikipediaHomePage(WebDriver driver) {
this.driver = driver; } public void navigateTo() {
driver.get("https://www.wikipedia.org"); } }
```

Archivo esperado de testing utilizando la clase creada:

```
WikipediaHomePageTest.java @Test public void
navigateToWikipediaHomePage() { WebDriver driver = new
ChromeDriver(); WikipediaHomePage homePage = new
WikipediaHomePage(driver); homePage.navigateTo();
assertEquals("Wikipedia", driver.getTitle()); driver.quit(); }
```

Ejercicio 1: Buscar en Wikipedia

Objetivo: Crear una clase de página para buscar en Wikipedia y una prueba para realizar una búsqueda.

Práctica POM II

Ejercicio 2: Navegar a una Página de Categoría en Wikipedia

Objetivo: Crear una clase de página para una categoría en Wikipedia y una prueba para navegar a dicha categoría.

Práctica POM III

Ejercicio 3: Verificar Contenido de Página en Wikipedia

Objetivo: Crear una clase de página para un artículo específico en Wikipedia y una prueba para verificar el contenido presente (puede ser la etiqueta P) de dicho artículo.

Ejercicio 1

Tomando como punto de partida los ejercicios anteriores, ahora tendrán la oportunidad de realizar prácticas integradoras con POM. La metodología de trabajo es la misma que los ejercicios anteriores.

Ejercicio 1: Navegar y comprobar el título de un artículo en Wikipedia

Objetivo: Crear una página de objeto para navegar a un artículo específico en Wikipedia y una prueba para comprobar que el título del artículo sea correcto.

Ejercicio 2

Ejercicio 2: Buscar y verificar un resultado en Wikipedia

Objetivo: Crear una página de objeto para realizar una búsqueda en Wikipedia y una prueba para verificar que un artículo específico aparezca en los resultados de búsqueda.

Ejercicio 3

Ejercicio 3: Navegar a una categoría y verificar un artículo en Wikipedia

Objetivo: Crear una página de objeto para navegar a una categoría específica en Wikipedia y una prueba para verificar que un artículo específico sea parte de esa categoría.

Introducción

Algunos elementos en los sitios web son atípicos (pero no por eso, poco frecuentes).

Los elementos a los que hacemos referencias son **las alertas, los iframes y los menús dropdowns**.

En este video, te contamos un poco más acerca de ellos.

https://youtu.be/V8W7xK_DsBE

Vamos a darte un ejemplo de cómo se interactúa con alert en Selenium:

Ejemplo

Objetivo: Navegar a la página de W3Schools que tiene un botón para activar una alerta. Al hacer clic en el botón, aparecerá una alerta, y tu tarea es aceptar la alerta utilizando Selenium.

URL de la página: [W3Schools Alert Example](#)

Resolución

Paso 1: Abre la página en tu navegador y ubica el botón que activa la alerta.

Paso 2: Haz clic en el botón utilizando Selenium.

Paso 3: Aceptar la alerta utilizando el método `alert.accept()` en Selenium.

```
codeimport org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class AlertExample {
    public static void main(String[] args) {
        // Iniciar el navegador
        WebDriver driver = new ChromeDriver();
```

```
// Navegar a la página

driver.get("https://www.w3schools.com/js/tryit.asp?filename=tr
yjs_alert");

// Cambiar al iframe que contiene el botón
driver.switchTo().frame("iframeResult");

// Hacer clic en el botón para activar la alerta

driver.findElement(By.xpath("/html/body/button")).click();

// Cambiar a la alerta
Alert alert = driver.switchTo().alert();

// Aceptar la alerta
alert.accept();

// Cerrar el navegador
driver.quit();
}
```

```
}
```

Prueba corriendo este script en Visual Studio Code para entender que hace el navegador. Puedes utilizar un wait antes de cerrar el navegador para tener más tiempo de visualización.

Ejercicio 1: Interactuar con una confirmación en W3Schools

Objetivo: Navegar a la página de W3Schools que tiene un botón para activar una confirmación. Al hacer clic en el botón, aparecerá una confirmación, y tu tarea es aceptar o cancelar la confirmación utilizando Selenium.

URL de la página: [W3Schools Confirm Example](https://www.w3schools.com/js/tryit.asp?filename=tryjs_confirm)

Ejercicio 2: Interactuar con una ventana de entrada (Prompt) en W3Schools

Objetivo: Navegar a la página de W3Schools que tiene un botón para activar una ventana de entrada. Al hacer clic en el botón, aparecerá una ventana de entrada, y tu tarea es ingresar un texto en la ventana y aceptarla utilizando Selenium.

URL de la página: [W3Schools Prompt Example](#)

Ejemplo

Ejemplo: Interactuar con un iFrame en W3Schools

Objetivo: Navegar a la página de W3Schools que contiene un botón para "Try it Yourself". Al hacer clic en ese botón, aparecerá un editor dentro de un iframe. Tu tarea es cambiar el contenido HTML dentro del iframe utilizando Selenium.

URL de la página: [W3Schools HTML Tryit Editor](#)

Instrucciones:

1. Navegar a la URL dada.
2. Hacer clic en el botón "Try it Yourself" para abrir el editor en el iframe.
3. Cambiar el contenido del elemento `<h1>` dentro del iframe a "¡Hola Mundo!".
4. Asegúrate de que tu código interactúe correctamente con el iframe y cambie el contenido como se describe.

Solución:

```
WebDriver driver = new ChromeDriver();  
driver.get("https://www.w3schools.com/html/tryit.asp?filename=tryhtml_default");
```

```
// Cambiar al iframe que contiene el editor  
driver.switchTo().frame("iframeResult");
```

```
// Encontrar el elemento <h1> y cambiar su contenido
```



```
WebElement h1Element = driver.findElement(By.tagName("h1"));
((JavascriptExecutor)
driver).executeScript("arguments[0].innerText = '¡Hola
Mundo!'", h1Element);

// (Opcional) Cambiar de nuevo al contenido principal fuera
del iframe
driver.switchTo().defaultContent();

driver.quit();
```

Ejercicios

Ejercicio 1: Modificar contenido en el Iframe de Quackit

Objetivo: Navegar a una página de Quackit que contiene un ejemplo de iframe. La tarea es encontrar y modificar el contenido dentro del iframe.

URL de la página: [Quackit Iframe Example](#)

Instrucciones:

1. Navegar a la URL dada.
2. Cambiar al iframe utilizando Selenium.
3. Modificar el texto dentro de un elemento `<p>` del iframe con el contenido "Texto modificado con Selenium".

Ejercicio 2: Interacción con el iframe de W3Schools

Objetivo: Navegar a otra página de W3Schools que contiene una calculadora dentro de un iframe. Tu tarea es realizar una simple suma utilizando esta calculadora.

URL de la página: [W3Schools Calculator](#)

Instrucciones:

1. Navegar a la URL dada.
2. Cambiar al iframe que contiene la calculadora.
3. Usar Selenium para hacer clic en los botones necesarios para sumar 5 + 5 y asegurarte de que la calculadora muestre "10" como resultado.

Ejemplo

Ejercicio: Selección de opción en Dropdown de W3Schools

Objetivo: Navegar a una página de W3Schools que contiene un dropdown y seleccionar una opción específica.

URL de la página: [W3Schools HTML](https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_select)

Instrucciones:

1. Navegar a la URL dada.
2. Cambiar al iframe que contiene el formulario con el dropdown.
3. Usar Selenium para seleccionar la opción "Saab" del dropdown.

Solución:

```
WebDriver driver = new ChromeDriver();
driver.get("https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_select");
driver.switchTo().frame("iframeResult");
Select dropdown = new
Select(driver.findElement(By.tagName("select")));
dropdown.selectByVisibleText("Saab");
String selectedOption =
dropdown.getFirstSelectedOption().getText();
assert selectedOption.equals("Saab");
driver.quit();
```

Ejercicios

Ejercicio 1: Selección de opción en Dropdown de la página Bootstrap

Objetivo: Navegar a una página de Bootstrap que contiene un dropdown y seleccionar una opción específica.

URL de la página: [Bootstrap Dropdown Example](#)

Instrucciones:

1. Navegar a la URL dada.
2. Usar Selenium para hacer clic en el botón del dropdown en el encabezado "Single button dropdowns."
3. Seleccionar la opción "Action" del dropdown.

Ejercicio 2: Selección de opción en Dropdown de la página de W3Schools

Objetivo: Navegar a otra página de W3Schools que contiene un dropdown y seleccionar una opción específica.

URL de la página: [W3Schools Tryit Editor](#)

Instrucciones:

1. Navegar a la URL dada.
2. Cambiar al iframe que contiene el formulario con el dropdown.
3. Usar Selenium para seleccionar la opción "Volvo" del dropdown.

Ejercicio 1

Ejercicio Integrador 1: W3Schools - Alertas, Dropdown y Iframe

Objetivo: Navegar a W3Schools, interactuar con un dropdown, un iframe y finalmente con una alerta.

URL de la página: [W3Schools - Tryit Editor](#)

Instrucciones:

1. Navegar a la página.
2. Cambiar al iframe "iframeResult".
3. Hacer clic en el botón "Try it" para generar una alerta.
4. Aceptar la alerta.
5. Volver a cambiar al contexto de la página principal.
6. Ir a la página de HTML select tags ([W3Schools - HTML Select](#)).
7. Cambiar al iframe "iframeResult" y seleccionar "Volvo" del dropdown.

Ejercicio 2

Ejercicio Integrador 2: W3Schools - Alerta, Iframe y Dropdown en una misma página

Objetivo: Navegar a W3Schools, interactuar con un dropdown dentro de un iframe, y también interactuar con una alerta.

URL de la página: [W3Schools - HTML Select](#)

Instrucciones:

1. Navegar a la página.
2. Cambiar al iframe "iframeResult".
3. Seleccionar "Saab" del dropdown con el id "cars".
4. Usar JavaScript para generar una alerta que diga "Seleccionado Saab".
5. Aceptar la alerta.

Práctica de Localizadores CSS

Parte I

Entra en este enlace <https://flukeout.github.io/> y observa los ejemplos gráficos y el código HTML. Escribe los selectores necesarios para localizar únicamente los elementos requeridos. Asegúrate de completar los 32 niveles.

Parte II

Identifica y escribe en un documento todos los selectores CSS necesarios para ejecutar los siguientes escenarios en el sitio web <https://demoblaze.com/>:

1. Registro exitoso.
2. Navegación a la sección "About Us" (Acerca de nosotros) y reproducción del video allí.
3. Selección y finalización del proceso de compra de cualquier laptop en el sitio web.

Asegúrate de agrupar en el documento los selectores respectivos de acuerdo con la página o vista en la que se encuentran. El CSS debe ser lo más eficiente y mantenible posible.

Integración Alerts, Iframes, Dropdowns

Estás automatizando la aplicación web

<https://www.globalsqa.com/demo-site/frames-and-windows/>. Crea un nuevo proyecto Selenium - Java y configúralo para lanzar y probar la página web requerida en el navegador Chrome. Implementa en tu marco de trabajo el siguiente proceso, utilizando objetos de página según sea necesario.

1. Haz clic en la pestaña "Open new tab" (Abrir nueva pestaña).
2. Haz clic en el botón azul "Click Here" (Haz clic aquí) para abrir la nueva pestaña.
3. Cambia a la pestaña recién abierta y haz clic en la pestaña Iframe.
4. En el menú desplegable "Trainings" (Formaciones), dentro del iframe, selecciona "Software Testing" (Pruebas de software).
5. Haz clic en la formación "Manual Testing" (Pruebas manuales).

6. Verifica con las aserciones requeridas que la página de formación en pruebas manuales se muestra correctamente.
7. En la barra lateral derecha, en la sección "Miscellaneous" (Varios), haz clic en el enlace "AlertBox" (Cuadro de alerta).
8. Cierra la pestaña actual y cambia a la pestaña inicial del navegador.
9. Navega a la página web
<https://demo.automationtesting.in/Alerts.html>
10. Haz clic en la pestaña "Alert with OK & Cancel" (Alerta con OK y Cancelar).
11. Haz clic en el botón para mostrar el cuadro de confirmación.
12. Haz clic en el botón Cancelar en el cuadro de alerta.
13. Asegúrate con aserciones de que el texto "You Pressed Cancel" (Presionaste Cancelar) se muestra debajo del botón.

Práctica de Esperas (Waits)

Tu tarea es automatizar la funcionalidad de inicio de sesión de la aplicación web <https://www.saucedemo.com/v1/>. El proceso de inicio de sesión implica ingresar un nombre de usuario y contraseña, y luego hacer clic en el botón de inicio de sesión. Sin embargo, podría haber retrasos en la renderización de la página de inicio de sesión o en la carga de los elementos, por lo que necesitas implementar estrategias de espera apropiadas para manejar estos escenarios.

Comienza creando un nuevo proyecto de Selenium con Java y, desde allí, navega a la URL de la aplicación web. Identifica y declara como WebElements todos los elementos necesarios para ejecutar correctamente el proceso de inicio de sesión. Una vez que tienes el proyecto configurado y en funcionamiento, implementa las siguientes estrategias de espera en diferentes pruebas.

a. Espera implícita (Implicit Wait):

- Establece una espera implícita de 10 segundos usando la instancia de WebDriver.

- Encuentra los elementos necesarios usando los localizadores y realiza acciones como enviar teclas a los campos de nombre de usuario y contraseña.
- Haz clic en el botón de inicio de sesión.
- Observa cómo la espera implícita maneja el retraso entre búsquedas de elementos y acciones.

b. Espera explícita con Condiciones Esperadas (Explicit Wait with Expected Conditions):

- Establece una espera explícita usando la clase `WebDriverWait`.
- Implementa una condición de espera para la visibilidad del campo de entrada del nombre de usuario.
- Una vez que el elemento es visible, envía teclas al campo de nombre de usuario.
- Implementa una condición de espera para la visibilidad del campo de entrada de la contraseña.
- Una vez que el elemento es visible, envía teclas al campo de contraseña.
- Implementa una condición de espera para que el botón de inicio de sesión sea "clicable".
- Una vez que el botón es "clicable", haz clic en él.
- Observa cómo la espera explícita espera las condiciones específicas antes de realizar acciones.

c. Espera Fluida (Fluent Wait):

- Implementa una espera fluida utilizando las clases `Wait` y `FluentWait`.
- Personaliza las condiciones de espera y el intervalo de sondeo según el comportamiento de la página web.
- Usa la espera fluida para esperar la presencia del campo de entrada del nombre de usuario.
- Una vez que el elemento está presente, envía teclas al campo de nombre de usuario.
- Usa la espera fluida para esperar la presencia del campo de entrada de la contraseña.
- Una vez que el elemento está presente, envía teclas al campo de contraseña.
- Usa la espera fluida para esperar que el botón de inicio de sesión sea "clicable".

- Una vez que el botón es "clicable", haz clic en él.
- Observa cómo la espera fluida proporciona flexibilidad al establecer condiciones de espera.

En todas las pruebas definidas, después de hacer clic en el botón de inicio de sesión, identifica situaciones de éxito/fallo y capta estos resultados usando elementos web y condicionales. Luego, imprime mensajes de acuerdo con los resultados de las pruebas. Usa el driver para cerrar el navegador después de que se ejecuten todas las pruebas.

En todos los escenarios de prueba que utilizan diferentes estrategias de espera, identifica las diferencias en los tiempos de espera, el manejo de errores u otras observaciones relevantes.

Introducción a Appium

¡Te damos la bienvenida a Mobile Automation!

A continuación, te compartimos un video de introducción para que puedas adentrarte a este nuevo módulo:

<https://youtu.be/JBGdXQXaX64>

También te compartimos la [documentación oficial de Appium](#) para que siempre tengas a mano.

Instalación

Necesitamos aplicar los conceptos básicos de Appium para instalar la herramienta de manera adecuada, junto con el controlador UIAutomator, y empezar a identificar y conectarnos con dispositivos Android.

Para comenzar, asegúrate de tener instalada la versión LTS más reciente de Node.js.

Windows:

1. Ve al sitio web oficial de Node.js:
[Node.js](https://nodejs.org/)
2. Descarga el instalador de la versión LTS (Long Term Support) para Windows.
3. Ejecuta el instalador y sigue las instrucciones en pantalla.

macOS:

1. Si ya tienes instalado Homebrew, puedes instalar Node.js con el siguiente comando:
2.

```
brew install node@lts
```


Si no tienes Homebrew, puedes descargarlo desde [aquí](#).
3. Alternativamente, puedes descargar el instalador de la versión LTS desde el sitio web oficial de [Node.js](https://nodejs.org/) y seguir las instrucciones en pantalla.

Linux (Debian/Ubuntu):

1. Abre una terminal y ejecuta los siguientes comandos para instalar Node.js:
2.

```
sudo apt update  
sudo apt install nodejs npm
```

Para verificar que Node.js se haya instalado correctamente, puedes abrir una terminal y ejecutar:

```
node -v
```

¿Se agrega Node.js a Visual Studio Code?

No exactamente, Node.js no se "agrega" a Visual Studio Code (VS Code) en el sentido de un complemento o extensión, pero puedes ejecutar y depurar programas de Node.js directamente en VS Code. Visual Studio Code detecta automáticamente la instalación de Node.js en tu sistema y ofrece muchas funciones de desarrollo para acelerar tu flujo de trabajo, como la depuración integrada y la autocompletación de código para bibliotecas de Node.js.

¿Cómo utilizar Node.js en VS Code?

1. **Instalar Node.js:** Asegúrate de que Node.js esté instalado en tu sistema. Si no está instalado, sigue las instrucciones de instalación para tu sistema operativo como se indicó en la respuesta anterior.
2. **Verificar la instalación:** Abre una terminal y ejecuta `node -v` para asegurarte de que Node.js se haya instalado correctamente.
3. **Abrir VS Code:** Una vez que Node.js esté instalado, abre VS Code.
4. **Crear/Ejecutar Archivo:** Puedes crear un nuevo archivo JavaScript (`archivo.js`) y escribir tu código Node.js allí. Luego, puedes abrir una terminal en VS Code (**Terminal > Nuevo Terminal**) y ejecutar `node archivo.js` para ejecutar tu código.
5. **Depuración:** VS Code tiene un depurador integrado que puedes configurar para Node.js. Solo necesitas establecer puntos de interrupción en tu código y hacer clic en el botón de inicio (el triángulo verde) en la barra lateral de depuración.
6. **Extensiones:** Hay varias extensiones disponibles para mejorar el desarrollo de Node.js en VS Code. Puedes encontrarlas en el Marketplace de Extensiones dentro de VS Code.

Instalación Appium

Luego, sigue esta guía para instalar Appium:

<http://appium.io/docs/en/2.0/quickstart/install/>.

Una vez que Appium esté instalado, consulta la siguiente guía para instalar el controlador UIAutomator: <http://appium.io/docs/en/2.0/quickstart/uiauto2-driver/>.

Instalación Android Studio

Inmediatamente después de tener todo el software instalado, debemos conectar un dispositivo y asegurarnos de que el servidor de Appium lo detecte correctamente. Para hacerlo, instala la versión más reciente de Android Studio y crea y ejecuta cualquier

dispositivo Android de tu elección utilizando el emulador de Android incluido (los dispositivos más recientes suelen funcionar mejor).

Windows:

1. **Descargar el instalador:** Ve al sitio web oficial de Android Studio y descarga el instalador para Windows. [Descargar Android Studio](#)
2. **Ejecutar el Instalador:** Busca el archivo descargado y haz doble clic para iniciar el proceso de instalación.
3. **Asistente de Instalación:** Sigue las instrucciones del asistente de instalación. Esto instalará tanto Android Studio como el SDK de Android.
4. **Finalizar la Instalación:** Una vez que la instalación se haya completado, ejecuta Android Studio. El programa te guiará a través de cualquier configuración adicional que pueda ser necesaria.

macOS:

1. **Descargar el instalador:** Ve al sitio web oficial de Android Studio y descarga el paquete para macOS. [Descargar Android Studio](#)
2. **Abrir el Archivo DMG:** Haz doble clic en el archivo `.dmg` descargado.
3. **Arrastrar a la Carpeta de Aplicaciones:** Arrastra el icono de Android Studio al directorio de Aplicaciones.
4. **Ejecutar Android Studio:** Abre Android Studio desde la carpeta de Aplicaciones. La primera vez que lo hagas, se instalarán automáticamente varios componentes adicionales, como el SDK de Android.

Linux:

1. **Descargar el paquete:** Ve al sitio web oficial de Android Studio y descarga el paquete para Linux. [Descargar Android Studio](#)
2. **Extraer el Archivo:** Extrae el archivo `.zip` descargado en un directorio de tu elección.
3. **Ejecutar Android Studio:** Navega al directorio donde extrajiste Android Studio y ejecuta el archivo `studio.sh` para iniciar Android Studio.

```
codecd android-studio/bin ./studio.sh
```

4. **Configuración Adicional:** Sigue las instrucciones en pantalla para completar la instalación, que incluirá descargar componentes adicionales como el SDK de Android.

Una vez que el emulador esté configurado y en funcionamiento, abre Appium y conéctate al emulador utilizando las capacidades requeridas. Al iniciar la sesión en Appium, estaremos listos para desarrollar frameworks de automatización de pruebas para dispositivos móviles.

Conexión a Android

Conectar Appium con un dispositivo Android implica una serie de pasos que deberás seguir cuidadosamente.

Prerrequisitos:

- Asegúrate de tener instalados Node.js, Appium y Android Studio
- Habilita la "Depuración USB" en tu dispositivo Android desde las "Opciones para desarrolladores". Para dispositivos iOS, necesitarás un certificado de desarrollo.

Para dispositivos Android:

1. **Conectar el Dispositivo:** Conecta tu dispositivo Android a tu computadora a través de un cable USB.
2. **Verificar Conexión:** Abre una terminal y ejecuta `adb devices` para asegurarte de que tu dispositivo está correctamente conectado. Deberías ver un identificador de dispositivo en la lista.
3. **Iniciar Appium:** Abre el servidor de Appium. Puedes hacer esto desde la interfaz de usuario de Appium o ejecutando `appium` en la terminal.
4. **Configurar Capacidades:** Configura las capacidades deseadas en tu script de pruebas. Las capacidades son pares clave-valor que definen aspectos como el dispositivo, la plataforma y la aplicación que deseas probar. Aquí hay un ejemplo básico en Javascript para Android:

```
"platformName": "Android", "deviceName":
```

```
"TU_IDENTIFICADOR_DE_DISPOSITIVO", "app":  
"RUTA/AL/ARCHIVO/APK" }
```

5. **Ejecutar Pruebas:** Ejecuta tu script de pruebas usando un marco como Appium, Selenium o cualquier otro compatible.