

Taller - Momento 2 - Pruebas de Software I

DAVID SANTIAGO TORRES FONSECA



FUNDACIÓN UNIVERSITARIA COMPENSAR

Pruebas de Software I

2025

Taller Momento 2

Integrantes: David Santiago Torres F. Fecha de entrega: 22 de abril de 2025 Modo de entrega: Documento en PDF

Repositorio de GitHub- Taller momento 2

# Cuestionario de selección múltiple

A continuación se presenta un conjunto de 9 preguntas de selección múltiple diseñadas para evaluar tus conocimientos de acuerdo con los temas vistos en el Segundo Momento de Pruebas de Software.

1. ¿Cuál de los siguientes localizadores es considerado el más rápido y confiable en Selenium cuando está disponible correctamente en el HTML?
   * a) name
   * b) className
   * c) id
   * d) cssSelector
2. ¿Qué herramienta de Selenium permite grabar, editar y reproducir scripts de pruebas automatizadas directamente desde el navegador sin necesidad de escribir código manualmente?
   * a) Selenium WebDriver
   * b) Selenium IDE
   * c) Selenium Grid
   * d) Selenium RC
3. En Git, ¿qué comando se utiliza para añadir cambios al área de preparación?
   * a) git commit
   * b) git push
   * c) git add
   * d) git clone
4. En la estructura de un proyecto de automatización con Selenium WebDriver, ¿qué diferencia principal existe entre una estructura de APP y un Framework de Testing?
   * a) Un Framework separa claramente el código de producción y pruebas para facilitar el mantenimiento
   * b) Un Framework no requiere configuración de dependencias
   * c) Una APP siempre es más fácil de mantener que un Framework
   * d) Una APP siempre utiliza anotaciones @Test mientras que un Framework no
5. En Cucumber, ¿qué formato se utiliza para escribir los escenarios de prueba en lenguaje natural?
   * a) XML
   * b) Gherkin
   * c) JSON
   * d) Markdown
6. ¿Qué comando de Selenium IDE se utiliza para verificar que un texto visible en la página coincida exactamente con el valor esperado?
   * a) assert
   * b) assertText
   * c) verifyText
   * d) pause
7. En el concepto de los principios SOLID, ¿qué significa la "S"?
   * a) Señalamiento de dependencias
   * b) Sincronización de clases
   * c) Sustitución de Liskov
   * d) Single Responsibility Principle (Principio de Responsabilidad Única)
8. En Gradle, ¿qué dependencia se utiliza para agregar únicamente las bibliotecas necesarias para las pruebas y no para el código principal de la aplicación?
   * a) implementation
   * b) testImplementation
   * c) compileOnly
   * d) runtimeOnly
9. En el contexto de los principios SOLID, ¿qué principio establece que las clases deben depender de abstracciones en lugar de implementaciones concretas?
   * a) Principio de Sustitución de Liskov
   * b) Principio de Responsabilidad Única
   * c) Principio de Inversión de Dependencias
   * d) Principio de Segregación de Interfaces

# Consulta por internet

#### Pregunta abierta 1:

Investiga y explica las diferencias entre los diferentes tipos de esperas en Selenium WebDriver (explícita, implícita y fluent wait). ¿En qué situaciones es más conveniente utilizar cada una de ellas?.

**RTA:**

**Diferencias entre los tipos de esperas en Selenium WebDriver**

En Selenium WebDriver, existen tres tipos principales de esperas que se utilizan para gestionar el tiempo de espera antes de que un elemento esté disponible en la página: **espera implícita**, **espera explícita** y **fluent wait**. Cada una de ellas se adapta a diferentes necesidades dentro de las pruebas automatizadas.

**1. Espera Implícita (Implicit Wait)**  
La espera implícita se configura una sola vez al inicio de la sesión del WebDriver. Con esta configuración, Selenium esperará un tiempo máximo definido cada vez que intente encontrar un elemento en la página antes de lanzar una excepción del tipo NoSuchElementException.  
Ejemplo:

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

Esto significa que si el elemento no aparece de inmediato, el WebDriver seguirá buscando hasta por 10 segundos. Es recomendable usar esta espera en proyectos sencillos, donde la latencia en la carga de elementos sea relativamente constante y no se requiera un control detallado sobre cada elemento específico.

**2. Espera Explícita (Explicit Wait)**  
A diferencia de la espera implícita, la espera explícita se aplica en momentos específicos del código y para condiciones concretas. Se utiliza junto con clases como WebDriverWait y ExpectedConditions, permitiendo esperar, por ejemplo, hasta que un botón sea clickeable o que un texto esté presente en un elemento.  
Ejemplo:

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));

WebElement boton = wait.until(ExpectedConditions.elementToBeClickable(By.id("submit")));

Este tipo de espera es útil cuando estás trabajando con contenido dinámico que puede tardar en aparecer o cambiar, y necesitas garantizar que una condición específica se cumpla antes de continuar con la ejecución de la prueba.

**3. Fluent Wait**  
El FluentWait es una variante avanzada de la espera explícita que ofrece un mayor nivel de personalización. Permite definir el tiempo máximo de espera, el intervalo entre intentos de búsqueda (sondeo) y las excepciones que se deben ignorar durante ese proceso.  
Ejemplo:

Wait<WebDriver> fluentWait = new FluentWait<>(driver)

.withTimeout(Duration.ofSeconds(30))

.pollingEvery(Duration.ofSeconds(2))

.ignoring(NoSuchElementException.class);

WebElement elemento = fluentWait.until(d -> d.findElement(By.cssSelector(".item")));

Este tipo de espera es ideal cuando necesitas controlar con precisión la frecuencia con la que se verifica una condición, especialmente en aplicaciones que muestran elementos de forma irregular o con demoras inesperadas. Además, permite manejar errores temporales sin interrumpir la prueba de inmediato.

**¿Cuándo usar cada una?**

* La **espera implícita** es adecuada para proyectos simples, donde todos los elementos suelen cargarse de manera uniforme.
* La **espera explícita** es más útil en situaciones puntuales donde necesitas esperar a que se cumpla una condición específica antes de interactuar con el elemento.
* El **fluent wait** es recomendable cuando se requiere un control más detallado del proceso de espera, especialmente en interfaces con tiempos de carga impredecibles o donde es necesario manejar excepciones de forma personalizada.

#### Pregunta abierta 2:

Investiga sobre las mejores prácticas para la automatización de pruebas con Selenium y Cucumber en un proyecto real. ¿Cómo se estructura un proyecto que sigue el patrón Page Object Model y BDD? ¿Qué ventajas ofrece esta combinación de herramientas y enfoques para los equipos de QA y desarrollo?

RTA:

Una estructura típica de carpetas en general se realiza así:

/src

/main

/java

/app ← acá va el código de la app si lo estás mezclando con pruebas

/test

/java

/pages ← aquí van las clases de Page Object (una por cada página o componente)

/steps ← acá van los métodos que hacen match con los pasos de Cucumber

/runners ← las clases que se encargan de correr los tests

/resources

/features ← los archivos .feature escritos en Gherkin

/data ← datos para las pruebas (JSON, CSV, etc.)

### ¿Qué es el Page Object Model?

El POM es una forma súper ordenada de manejar las páginas de tu aplicación. Cada página se convierte en una clase Java donde defines:

* Los **localizadores** (usando @FindBy o By).
* Los **métodos** que interactúan con esa página (como hacer clic, escribir texto, leer algo, etc.).

La idea es no repetir código y que, si algo cambia en la UI, lo modifiques solo en un lugar y no en cada test.

### ¿BDD con Cucumber?

Con Cucumber, se escribe los escenarios en un lenguaje legible (Gherkin), usando los famosos Given, When, Then, y eso facilita que cualquier persona del equipo (QA, devs, analistas) entienda qué se está probando.

* Los **feature files** son esos archivos .feature donde escribís los escenarios.
* Las **Step Definitions** son las clases donde cada línea del feature se conecta con código Java que llama a los métodos de tus Page Objects.
* Y los **runners** se encargan de ejecutar todo, configurando paths, reportes, y demás cositas con @CucumberOptions.

### ¿Qué ventajas tiene mezclar todo esto?

* **Mejora la comunicación** con un lenguaje que se puede entender sin ser desarrollador.
* **Separación de responsabilidades**:
  + Los features explican **qué** se quiere probar.
  + Los steps definen **cómo** se hace.
  + Los page objects dicen **dónde** se hace.
* **Mantenibilidad**: Si cambian un botón o campo en la interfaz, solo se tiene que modificarlo en el Page Object.
* **Reutilización**: Los métodos que se hacen en los Page Objects te sirven para varios tests, así que no se escribe todo de nuevo.
* **Trabajo en equipo**: Personas con distintos perfiles pueden colaborar sin problemas, porque los escenarios están escritos de forma simple.

# Actividades Prácticas con Selenium IDE

El objetivo de estas actividades es aplicar los conocimientos adquiridos sobre automatización de pruebas utilizando Selenium IDE. A través de dos ejercicios prácticos, deberás diseñar y ejecutar casos de prueba para escenarios de uso común en aplicaciones web.

### Para ambas actividades:

1. **Formato Gherkin obligatorio**: Todos los casos de prueba deben estar escritos siguiendo la sintaxis Gherkin (Given-When-Then), que facilita la comprensión de los escenarios para todos los miembros del equipo, sean técnicos o no.

### Ejemplo de caso de prueba en Gherkin:

**Feature**: Inicio de sesión en DemoShop

**Scenario**: Inicio de sesión exitoso

**Give** que estoy en la página de inicio de sesión

**When** ingreso el nombre de usuario "standard\_user"

**And** ingreso la contraseña "secret\_sauce"

**And** hago clic en el botón de inicio de sesión

**Then** debería ver la página de productos

**And** debería ver el título "Products" en la página

1. **Comandos de verificación**: Cada caso de prueba debe incluir al menos una verificación (assertion) que valide el resultado esperado. Utiliza comandos como assert, assertText, o verifyPresent según corresponda.
2. **Documentación**: Además de la automatización, deberás entregar una explicación de los casos de prueba propuestos y una descripción breve de los desafíos encontrados durante la implementación.

## Entregables

Para cada actividad deberás entregar:

1. Archivo de proyecto Selenium IDE (.side) con todos los casos implementados
2. Documento con los 5 casos de prueba escritos en formato Gherkin
3. Vídeo mostrando la ejecución exitosa de cada caso de prueba.
4. Breve análisis de desafíos y soluciones.

## Actividad 1: Automatización de un proceso de compra en tienda online

### Historia de Usuario:

**Como** usuario de la tienda online DemoShop,

**Quiero** poder buscar productos, agregarlos al carrito y completar un proceso de compra,

**Para** adquirir los artículos que necesito de manera eficiente.

### Pistas para formular los casos de prueba:

1. **Piensa en el flujo completo**: desde la búsqueda hasta la confirmación de compra.
2. **Considera casos positivos y negativos**: ¿qué pasa cuando las validaciones fallan?
3. **Identifica elementos críticos** que necesitan verificación como mensajes de confirmación o errores.

#### Utiliza la estructura Gherkin correctamente:

* + **Given**: contexto inicial (estoy en la página, tengo una cuenta, etc.)
  + **When**: acciones que realiza el usuario (búsqueda, clic, ingreso de datos)
  + **Then**: resultados esperados (aparece mensaje, cambia la URL, etc.)

1. **Aprovecha las funcionalidades de Selenium IDE**: comandos de verificación, pausas cuando sea necesario, variables para elementos comunes.

### URL para realizar las pruebas:

https://[www.saucedemo.com/](http://www.saucedemo.com/)

### Casos de prueba a formular:

Debes desarrollar 5 casos de prueba que cubran:

1. **Inicio de sesión (ejemplo anterior)**

RTA:

Feature: Inicio de sesión en Swag Labs

Scenario: Inicio de sesión exitoso

Given que estoy en la página de inicio de sesión

When ingreso el nombre de usuario "standard\_user"

And ingreso la contraseña "secret\_sauce"

And hago clic en el botón de inicio de sesión

Then debería ver la página de productos

And debería ver el título "Products" en la página

1. **Búsqueda/filtrado de productos**

RTA:

Scenario: Ordenar productos por precio de menor a mayor

Given que estoy en la página de productos

When selecciono la opción "Price (low to high)" del filtro de orden

Then los productos deberían mostrarse en orden ascendente por precio

And el primer producto debería ser el más económico

1. **Añadir productos al carrito**

RTA:

Feature: Añadir productos al carrito en Swag Labs

Scenario: Añadir producto al carrito

Given que estoy en la página de un producto específico

When hago clic en el botón "Add to Cart"

Then debería ver un mensaje de confirmación de que el producto fue añadido al carrito

And el icono de carrito en la barra de navegación debería mostrar "1"

1. **Proceso de checkout**

RTA:

Feature: Proceso de checkout en Swag Labs

Scenario: Proceso de checkout exitoso

Given que estoy en la página de carrito de compras

When hago clic en el botón "Checkout"

And Valido los detalles de envío y pago

And hago clic en el botón "Finish"

Then debería ver la página de confirmación de compra

And debería recibir un mensaje de confirmación de compra exitosa

1. **Validación de errores (campo requerido, datos inválidos, etc.)**

RTA:

Feature: Validación de errores en Swag Labs

Scenario: Validación de campo requerido en el formulario de checkout

Given que estoy en la página de checkout

When intento continuar sin completar los campos obligatorios

Then debería ver mensajes de error indicando campos requeridos

And no debería ser capaz de avanzar al siguiente paso del checkout

### Resumen

* + Trabajarás con un sitio de e-commerce demo (SauceDemo)
  + Deberás cubrir el flujo completo desde el login hasta la confirmación de compra
  + Los 5 casos de prueba deben estar interrelacionados siguiendo el flujo de compra
  + Debes implementar al menos un caso que valide mensajes de error

## Actividad 2: Automatización de formulario de registro con validaciones

### Historia de Usuario:

**Como** visitante del sitio web de OrangeHRM,

**Quiero** poder registrarme proporcionando mis datos personales,

**Para** crear una cuenta y acceder a las funcionalidades del sistema.

### Pistas para formular los casos de prueba:

1. **Identifica los campos obligatorios** y opcionales del formulario.
2. **Considera las validaciones** que debe tener cada campo (longitud mínima, formato de email, etc.).
3. **Piensa en diferentes escenarios**: registro exitoso, campos faltantes, formatos inválidos, contraseñas que no coinciden.
4. **Usa comandos de espera** cuando sea necesario para permitir que la página cargue correctamente.
5. **Verifica los mensajes de error o éxito** que aparecen después de enviar el formulario.
6. **Utiliza datos de prueba variados** para cada escenario.

### URL para realizar las pruebas:

https://opensource-demo.orangehrmlive.com/web/index.php/auth/login

### Casos de prueba a formular:

Debes desarrollar 5 casos de prueba que cubran:

1. Login exitoso

RTA:

Feature: Inicio de sesión en OrangeHRM

Scenario: Inicio de sesión con credenciales válidas

Given que estoy en la página de inicio de sesión de OrangeHRM

When ingreso el usuario "Admin"

And ingreso la contraseña "admin123"

And hago clic en el botón de login

Then debería acceder al panel principal del sistema

And debería ver el título "Dashboard"

1. Login fallido (credenciales incorrectas)

RTA:

Feature: Inicio de sesión en OrangeHRM

Scenario: Inicio de sesión con credenciales inválidas

Given que estoy en la página de inicio de sesión de OrangeHRM

When ingreso el usuario "Admin"

And ingreso la contraseña "admin12345678"

And hago clic en el botón de login

Then debería ver un mensaje de error

And el mensaje debería decir "Invalid credentials"

1. Recuperación de contraseña (ejemplo anterior)

RTA:

Feature: Recuperación de contraseña en OrangeHRM

Scenario: Solicitar recuperación de contraseña

Given que estoy en la página de inicio de sesión

When hago clic en el enlace "Forgot your password?"

And ingreso el nombre de usuario "Admin"

And hago clic en el botón "Reset Password"

Then debería ver un mensaje que indica que se ha enviado un correo para restablecer la contraseña

1. Validación de campos obligatorios vacíos

RTA:

1. Navegación entre diferentes secciones después del login

RTA:

Feature: Navegación en OrangeHRM

Scenario: Acceder a diferentes secciones del sistema

Given que he iniciado sesión exitosamente con el usuario "Admin"

When hago clic en el menú "PIM"

Then debería ser redirigido a la sección de gestión de empleados

And el título de la página debería ser "PIM"

### Resumen

* + Utilizarás una aplicación de gestión de recursos humanos (OrangeHRM)
  + Los casos se centrarán en validaciones de formularios y navegación
  + Debes incluir verificaciones para los mensajes de error cuando los datos son inválidos
  + Se valorará el uso de comandos de espera cuando sea necesario

**Breve análisis de desafíos y soluciones**

Durante la automatización de las pruebas con Selenium IDE, me encontré con varios desafíos que me obligaron a pensar en soluciones prácticas para lograr que los casos funcionaran correctamente.

Uno de los principales retos fue manejar los \*\*tiempos de carga\*\* de las páginas. En algunos casos, Selenium intentaba interactuar con los elementos antes de que aparecieran completamente, lo que causaba errores. Para solucionarlo, utilicé comandos como `waitForElementPresent` o `pause` para darle tiempo al sistema antes de continuar con la siguiente acción.

También fue un poco confuso al inicio familiarizarme con la estructura Gherkin\* pero una vez entendí que era solo una forma clara y ordenada de escribir los pasos (Given-When-Then), se volvió más fácil plazmar mis ideas en escenarios concretos.

En resumen, aunque al principio parecía complicado, los retos me ayudaron a entender mejor cómo funciona la automatización con Selenium IDE y a valorar la importancia de escribir pruebas bien estructuradas para asegurar que una aplicación web funcione correctamente en distintos escenarios.