5 de agosto de 2025

Dante Escalona

Análisis de Caso

Selenium IDE

**Introducción**

Selenium IDE es una herramienta de automatización de pruebas funcionales que permite grabar, editar y reproducir interacciones con aplicaciones web. En el contexto del caso de estudio de ShopEasy, se utilizó para validar flujos básicos de usuario de forma rápida y sin necesidad de conocimientos avanzados de programación.

**Ventajas de Selenium IDE**

1. **Facilidad de uso**  
   Su interfaz gráfica permite grabar pruebas sin escribir código, lo que lo hace ideal para testers principiantes o para prototipar rápidamente.
2. **Grabación automática de acciones**  
   Captura clics, entradas de texto y navegación de forma automática, facilitando la creación de scripts funcionales.
3. **Exportación a múltiples lenguajes**  
   Permite exportar los scripts grabados a lenguajes como Java (JUnit/TestNG), lo que facilita su integración en entornos de desarrollo más robustos.
4. **Extensión de navegador**  
   Al funcionar como una extensión de Chrome o Firefox, no requiere instalación compleja ni configuración adicional.
5. **Depuración visual**  
   Permite ejecutar paso a paso los scripts grabados, lo que ayuda a identificar errores o comportamientos inesperados.

**Limitaciones de Selenium IDE**

1. **Dependencia del navegador**  
   Solo funciona en navegadores compatibles con la extensión, lo que limita su uso en pruebas cross-browser más avanzadas.
2. **Escalabilidad limitada**  
   No es adecuado para pruebas complejas, pruebas de carga o integración continua sin exportar el código.
3. **Manejo limitado de esperas y condiciones dinámicas**  
   Aunque permite agregar comandos de espera, no es tan flexible como Selenium WebDriver para manejar condiciones dinámicas o asincronía.
4. **Falta de control estructurado**  
   No permite estructuras de control complejas como bucles o condicionales directamente en la interfaz.
5. **Mantenimiento de scripts**  
   Los scripts grabados pueden volverse frágiles ante cambios en la interfaz de usuario, lo que requiere ajustes frecuentes.

**Conclusión**

Selenium IDE es una herramienta poderosa para la creación rápida de pruebas funcionales en aplicaciones web. Su simplicidad y capacidad de exportación lo convierten en un excelente punto de partida para testers que desean validar flujos básicos y luego escalar sus pruebas en entornos más avanzados como Visual Studio Code con Java. Sin embargo, sus limitaciones técnicas hacen que no sea la mejor opción para proyectos de automatización a gran escala.

**Descripción del flujo grabado:**

1. Inicio del grabador en Selenium IDE.
2. Acceso a la URL: https://the-internet.herokuapp.com/login
3. Interacción con el formulario:
   * Ingreso de usuario: tomsmith
   * Ingreso de contraseña: SuperSecretPassword!
   * Clic en el botón de login
4. Validación visual del mensaje de éxito: “You logged into a secure area!”
5. Finalización de la grabación.
6. Exportación del script a Java (JUnit).

Este flujo fue grabado utilizando únicamente comandos básicos como open, type, click, y assertText.

@Test

public void testLoginSuccess() {

    logs.add("🚀 Iniciando prueba de login exitoso...");

    driver.get("https://the-internet.herokuapp.com/login");

    WebElement usernameField = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));

    WebElement passwordField = driver.findElement(By.id("password"));

    WebElement loginButton = driver.findElement(By.cssSelector("button[type='submit']"));

    usernameField.sendKeys("tomsmith");

    passwordField.sendKeys("SuperSecretPassword!");

    loginButton.click();

    WebElement successMessage = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("flash")));

    logs.add("✅ Mensaje de éxito detectado: " + successMessage.getText());

}

**Código fuente exportado y modificado en Visual Studio Code**

El código fue exportado desde Selenium IDE y luego modificado para incluir:

* **Esperas explícitas** con WebDriverWait
* **Validaciones con Assertions**
* **Generación automática de informe HTML**

Ejemplo de clase modificada:

@Test

public void testLoginSuccess() {

    logs.add("🚀 Iniciando prueba de login exitoso...");

    driver.get("https://the-internet.herokuapp.com/login");

    WebElement usernameField = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));

    WebElement passwordField = driver.findElement(By.id("password"));

    WebElement loginButton = driver.findElement(By.cssSelector("button[type='submit']"));

    usernameField.sendKeys("tomsmith");

    passwordField.sendKeys("SuperSecretPassword!");

    loginButton.click();

    WebElement successMessage = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("flash")));

    logs.add("✅ Mensaje de éxito detectado: " + successMessage.getText());

}

**Reflexión sobre la experiencia de utilizar Selenium IDE y su integración con Visual Studio Code**

**Selenium IDE** resultó ser una herramienta muy útil para prototipar rápidamente flujos funcionales sin necesidad de escribir código. Su interfaz intuitiva y la capacidad de exportar scripts a lenguajes como Java permiten una transición fluida hacia entornos más robustos.

La integración con **Visual Studio Code** fue sencilla y eficiente. Una vez exportado el script, se pudieron aplicar mejoras como:

* Uso de esperas explícitas para evitar fallos por carga lenta.
* Validaciones más precisas con Assertions.
* Generación de informes HTML para documentar los resultados.

Esta experiencia demuestra que Selenium IDE es ideal para comenzar pruebas funcionales, mientras que Visual Studio Code y Selenium WebDriver permiten escalar y profesionalizar el proceso de automatización.