Objetivo del escenario

Automatizaremos **un flujo de login + validación de productos** en la página saucedemo.com:

1. Ingresar a la web.
2. Iniciar sesión con usuario y contraseña válidos.
3. Validar que aparece la lista de productos.
4. Añadir un producto al carrito.
5. Confirmar que el contador del carrito muestra "1".

Usaremos:

- **Java + Maven + Selenium + JUnit 5**
- **Page Object Model (POM)** para separar lógica.
- **WebDriverManager** para gestionar drivers automáticamente.
- **GitHub Actions** para ejecutar en CI con Chrome en modo headless.
- **Capturas de pantalla** en cada test.

Estructura del proyecto

```
mi-proyecto-saucedemo/
├── pom.xml
├── src/test/java/com/example/tests/
│   ├── BaseTest.java
│   ├── utils/ScreenshotUtil.java
│   ├── pages/
│   │   ├── LoginPage.java
│   │   ├── ProductsPage.java
│   │   └── CartPage.java
│   └── SauceDemoTest.java
└── .github/workflows/
    └── selenium-ci.yml
```

1. pom.xml — dependencias y configuración

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>saucedemo-selenium</artifactId>
<version>1.0.0</version>

<properties>
 <maven.compiler.source>17</maven.compiler.source>
 <maven.compiler.target>17</maven.compiler.target>
 <selenium.version>4.23.0</selenium.version>
 <junit.jupiter.version>5.10.0</junit.jupiter.version>
 <webdrivermanager.version>5.8.0</webdrivermanager.version>
</properties>

<dependencies>
 <dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>${selenium.version}</version>
 </dependency>

 <dependency>
  <groupId>io.github.bonigarcia</groupId>
  <artifactId>webdrivermanager</artifactId>
  <version>${webdrivermanager.version}</version>
 </dependency>

 <dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
```

```
    <version>${junit.jupiter.version}</version>
    <scope>test</scope>
   </dependency>
  </dependencies>

  <build>
   <plugins>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-surefire-plugin</artifactId>
     <version>3.1.2</version>
    </plugin>
   </plugins>
  </build>
</project>
```

2. BaseTest.java — inicialización del WebDriver

```
package com.example.tests;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.TestInfo;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import com.example.tests.utils.ScreenshotUtil;

import java.time.Duration;

public class BaseTest {
```

```
  protected WebDriver driver;

  @BeforeEach
  public void setUp() {
    // Descarga e instala el driver automáticamente
    WebDriverManager.chromedriver().setup();

    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless=new"); // modo headless para CI
    options.addArguments("--no-sandbox");
    options.addArguments("--disable-dev-shm-usage");
    options.addArguments("--window-size=1920,1080");

    driver = new ChromeDriver(options);
    driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
  }

  @AfterEach
  public void tearDown(TestInfo testInfo) {
    ScreenshotUtil.takeScreenshot(driver, testInfo.getDisplayName());
    if (driver != null) {
      driver.quit();
    }
  }
}
```

Qué hace:

- Configura Chrome headless para ejecutarse en servidores sin interfaz gráfica (como GitHub Actions).
- WebDriverManager.chromedriver().setup() se encarga de descargar el binario correcto de chromedriver.
- implicitlyWait(10) da 10 segundos para encontrar elementos antes de lanzar error.

- tearDown hace dos cosas: toma una captura y cierra el navegador, garantizando limpieza entre pruebas.

3. pages/LoginPage.java — Page Object Model

```
package com.example.tests.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class LoginPage {
  private WebDriver driver;

  // Selectores
  private By usernameField = By.id("user-name");
  private By passwordField = By.id("password");
  private By loginButton = By.id("login-button");

  public LoginPage(WebDriver driver) {
    this.driver = driver;
  }

  public void open() {
    driver.get("https://www.saucedemo.com/");
  }

  public void login(String username, String password) {
    WebElement user = driver.findElement(usernameField);
    WebElement pass = driver.findElement(passwordField);
    WebElement button = driver.findElement(loginButton);

    user.clear();
```

```
    user.sendKeys(username);
    pass.sendKeys(password);
    button.click();
  }
}
```

Qué hace:

- Encapsula las acciones del login (abrir la página e introducir credenciales).
- Si el selector cambia (por ejemplo, cambia el id del botón), solo se actualiza aquí.
- Reutilizable por múltiples tests.

4. pages/ProductsPage.java

```
package com.example.tests.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.util.List;

public class ProductsPage {
  private WebDriver driver;

  private By productTitles = By.className("inventory_item_name");
  private By addToCartButton = By.xpath("(//button[contains(@id,'add-to-cart')])[1]");
  private By cartBadge = By.className("shopping_cart_badge");

  public ProductsPage(WebDriver driver) {
    this.driver = driver;
  }

  public List<String> getAllProductTitles() {
    return driver.findElements(productTitles)
```

```
            .stream()
            .map(WebElement::getText)
            .toList();
    }

    public void addFirstProductToCart() {
        driver.findElement(addToCartButton).click();
    }

    public String getCartCount() {
        return driver.findElement(cartBadge).getText();
    }
}
```

Qué hace:

- Localiza todos los títulos de productos para verificar el contenido.
- Simula agregar un producto al carrito.
- Devuelve el número del carrito (texto del ícono de carrito).

5. utils/ScreenshotUtil.java — capturas de pantalla

```
package com.example.tests.utils;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;

import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ScreenshotUtil {
```

```
  public static void takeScreenshot(WebDriver driver, String testName) {
    try {
      File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
      String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
      Path destDir = Path.of("target/screenshots");
      Files.createDirectories(destDir);
      Files.copy(screenshot.toPath(), destDir.resolve(testName + "_" + timestamp +
".png"));
    } catch (Exception ignored) {}
  }
}
```

Qué hace:

- Cada vez que termina un test (en @AfterEach), genera un archivo .png en target/screenshots/.
- El nombre incluye el nombre del test y timestamp.
- Muy útil cuando un test falla en CI y quieres ver qué pasó visualmente.

6. SauceDemoTest.java — flujo completo

```
package com.example.tests;

import com.example.tests.pages.LoginPage;
import com.example.tests.pages.ProductsPage;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class SauceDemoTest extends BaseTest {
```

```
  @Test
  @DisplayName("Login válido y verificación de productos y carrito")
  public void loginAndAddToCart() {
    LoginPage loginPage = new LoginPage(driver);
    ProductsPage productsPage = new ProductsPage(driver);

    loginPage.open();
    loginPage.login("standard_user", "secret_sauce");

    // Verificar que aparecen productos
    List<String> titles = productsPage.getAllProductTitles();
    assertTrue(titles.size() > 0, "Debería haber al menos un producto listado");

    // Añadir primer producto al carrito
    productsPage.addFirstProductToCart();
    assertEquals("1", productsPage.getCartCount(), "El contador del carrito debería
mostrar 1");
  }
}
```

Qué hace paso a paso:

1. Crea instancias de las páginas (LoginPage y ProductsPage).
2. Abre la web.
3. Realiza el login con credenciales válidas (standard_user / secret_sauce).
4. Verifica que aparecen productos → asegura que el login fue exitoso.
5. Añade un producto y verifica que el carrito muestra “1”.

Si el sitio cambia, el test falla y se genera una captura (target/screenshots/).

7. .github/workflows/selenium-ci.yml — CI/CD en GitHub Actions

```
name: Selenium Tests (SauceDemo)

on:
 push:
```

```
  branches: [ main ]
 pull_request:
  branches: [ main ]

jobs:
 selenium:
  runs-on: ubuntu-latest
  steps:
   - name: Checkout code
     uses: actions/checkout@v4

   - name: Set up Java 17
     uses: actions/setup-java@v4
     with:
      java-version: '17'
      distribution: 'temurin'
      cache: 'maven'

   - name: Install Google Chrome
     uses: browser-actions/setup-chrome@v3
     with:
      chrome-version: 'latest'

   - name: Run Selenium tests
     run: mvn -B clean test

   - name: Upload test reports
     if: always()
     uses: actions/upload-artifact@v4
     with:
      name: junit-reports
```

```
      path: target/surefire-reports/


  - name: Upload screenshots
    if: always()
    uses: actions/upload-artifact@v4
    with:
      name: screenshots
      path: target/screenshots/
```

Qué hace cada paso:

- **runs-on: ubuntu-latest**: usa un runner Linux con Chrome preinstalado.
- **actions/checkout**: baja tu código del repositorio.
- **setup-java**: instala Java y configura cache de Maven (para acelerar builds).
- **setup-chrome**: instala la última versión de Chrome (útil si tu código usa WebDriverManager).
- **mvn test**: ejecuta tus pruebas.
- **upload-artifact**: sube los resultados (JUnit y capturas) al panel de GitHub Actions.

Luego podrás descargar desde la pestaña **Artifacts** las capturas e informes tras cada ejecución.

Cómo entender la ejecución completa

1. Cuando haces un git push o PR, GitHub ejecuta el workflow automáticamente.
2. El runner instala Java y Chrome.
3. Maven descarga dependencias y ejecuta mvn test.
4. BaseTest abre Chrome en modo headless.
5. El test entra a saucedemo.com, hace login y verifica los pasos.
6. Si algo falla, ScreenshotUtil guarda la captura.
7. GitHub Actions publica los resultados y screenshots.

Qué se aprende con este ejemplo

Page Object Model (POM)
Separación de capas (test, pages, utils)
Ejecución headless en CI
Uso de WebDriverManager
Uso de JUnit5

Captura automática
Ejecución reproducible en GitHub Actions