Maria Fanny Giraldo - A00399948
John Hortua - A00373357
Santiago Velasquez - A00372725

“Proyecto Transversal – Automatización de pruebas sobre OpenCart”

Estrategia de automatización

Objetivo

Esta estrategia de automatización se centra en validar de forma sistemática cuatro funcionalidades clave del sitio demo de OpenCart (http://opencart.abstracta.us/): registro de usuario, inicio de sesión, búsqueda y agregado de productos al carrito, y verificación de productos en el carrito.

El objetivo principal es garantizar la **calidad funcional** de estos flujos críticos utilizando **buenas prácticas de automatización**, manteniendo trazabilidad de los datos de prueba y de los resultados mediante archivos Excel.

Arquitectura y patrón de diseño

El framework está implementado en Java con Maven y sigue el patrón **Page Object Model (POM)**:

- Cada página relevante del sistema tiene su propia clase en el paquete com.opencart.pages:
    - HomePage, RegisterPage, LoginPage, ProductPage, CartPage.
- Existe una clase base BasePage que encapsula:
    - La referencia al WebDriver.
    - Métodos reutilizables para interactuar con la UI (click, type, getText, isDisplayed).
    - Integración con WaitUtil para las esperas explícitas.

De esta manera, las clases de prueba (com.opencart.test) no acceden directamente a los localizadores de Selenium, sino que consumen métodos de alto nivel de las pages. Esto mejora la **mantenibilidad** y **legibilidad** del código, y desacopla la lógica de negocio de los detalles de la UI.

Manejo de datos externos con Excel (Apache POI)

Maria Fanny Giraldo - A00399948
John Hortua - A00373357
Santiago Velasquez - A00372725

Para garantizar la **parametrización** y la **trazabilidad** de los datos de prueba se utiliza **Apache POI**:

- Un archivo de entrada

inputData.xlsx contiene:

    - Hoja RegistroUsuarios con los datos de los usuarios a registrar.
    - Hoja DataLogin con combinaciones de email, contraseña y resultado esperado.
    - Hoja ProductosBusqueda con categoría, subcategoría, nombre del producto y cantidad.

- Un archivo de salida

Logs.xlsx centraliza los resultados de ejecución, separado por hojas:

    - Registro, Login, carrito y VerificacionCarrito.

La clase ExcelUtil ofrece dos operaciones principales:

- leerDatos(ruta, hoja): lee una hoja de Excel y retorna una lista de filas (List<String[]>) que es usada por los DataProvider de TestNG o por ciclos for en los tests.
- escribirLog(ruta, hoja, valores): escribe una nueva fila al final de la hoja (o la crea si no existe), registrando los resultados de cada caso: datos de entrada, estado (EXITOSO / FALLIDO / etc.) y mensajes de detalle.

Este diseño permite que los **datos de prueba** se mantengan fuera del código y facilita su modificación sin recompilar el proyecto.

Sincronización y selectores

Para manejar la sincronización con la aplicación web se implementó la clase WaitUtil, que envuelve WebDriverWait y ExpectedConditions:

- esperavisible(By locator): espera a que un elemento sea visible antes de interactuar con él.
- esperaClickeable(By locator): espera a que un elemento sea clickeable antes de hacer click.

Los selectores se definieron buscando un equilibrio entre **estabilidad** y **claridad**:

- Prioridad en By.id cuando está disponible.

- Uso de By.cssSelector para botones, alertas y elementos con clases identificables.
- Uso limitado de By.xpath y By.linkText cuando son más expresivos o necesarios (por ejemplo, enlaces con el texto del producto).

Se evita el uso de XPaths frágiles con índices absolutos o dependientes del layout, con el fin de reducir la sensibilidad a cambios menores en la UI.

Casos de prueba automatizados

Se implementaron cuatro clases de prueba principales, alineadas con el enunciado del proyecto:

1. **Registro de usuario (**RegisterTest**)**
    - Lee los datos desde la hoja RegistroUsuarios.
    - Utiliza RegisterPage para completar y enviar el formulario de registro.
    - Verifica:
        - Si aparece una alerta de error, registra el caso como fallido (o “YA_REGISTRADO” si el correo ya existía).
        - Si no hay error, valida el mensaje de éxito en la página de confirmación.
    - Escribe el resultado de cada intento en la hoja Registro de

Logs.xlsx.

2. **Inicio de sesión (**LoginTest**)**
    - Usa un DataProvider que lee la hoja DataLogin.
    - Para cada fila:
        - Intenta iniciar sesión con el email y la contraseña proporcionados.
        - Si el ExpectedResult es SUCCESS, verifica que no se muestre el mensaje de advertencia.
        - Si se espera error, valida que el warning esté visible.
    - Registra en la hoja Login los datos ingresados, el resultado esperado, el estado real y el mensaje asociado.

3. **Búsqueda y agregado al carrito (**SearchAndCartTest**)**
    - Recorre la hoja ProductosBusqueda.
    - Para cada producto válido:
        - Realiza la búsqueda desde HomePage.
        - Abre el detalle del producto y ajusta la cantidad requerida.
        - Si el producto tiene opciones desplegables, selecciona una opción por defecto.
        - Agrega el producto al carrito y valida la aparición del mensaje de éxito.
    - Cada iteración escribe una fila en la hoja carrito indicando categoría, subcategoría, producto, cantidad, estado y mensaje.

4. **Verificación de productos en el carrito (**CartVerificationTest**)**
    - En la misma sesión de navegador:
        - Vuelve a agregar los productos definidos en ProductosBusqueda.
        - Construye un mapa de productos esperados con sus cantidades totales.
    - Navega a la página de Shopping Cart y, mediante CartPage, lee los productos y cantidades reales presentes en el carrito.
    - Compara ambos mapas, validando que:
        - Todos los productos esperados estén presentes.
        - Las cantidades reales coincidan con las esperadas.
    - Registra cada verificación en la hoja VerificacionCarrito de

Logs.xlsx, con el estado y un mensaje de detalle.

Validaciones y reporting

Para las validaciones se utilizan principalmente aserciones de **TestNG**:

- Assert.assertTrue, Assert.assertFalse y Assert.assertEquals para comprobar:
    - Mensajes de éxito y de error.

- Presencia o ausencia de warnings.
- Existencia y cantidad de productos en el carrito.

Los resultados se reflejan en dos niveles:

1. **A nivel de framework de pruebas**
    - Aserciones que hacen fallar el test en caso de discrepancia.
    - Reportes estándar generados por Maven/TestNG (por ejemplo, en target/surefire-reports).
2. **A nivel de trazabilidad**
    - Escritura de logs en

Logs.xlsx para cada caso ejecutado, permitiendo:

- Analizar después qué datos pasaron o fallaron.
- Mostrar evidencia directa basada en los mismos datos de entrada.

Esta combinación de POM, datos externos en Excel, esperas explícitas y logging en Excel entrega una solución de automatización **modular, reutilizable y trazable**, adecuada para evaluar atributos de calidad en un sistema de dominio específico como OpenCart.

Conclusiones

Tenicamente

- Se logró automatizar los 4 casos funcionales requeridos.
- Se aplicó POM, esperas explícitas y datos externos en Excel.

En términos de calidad:

- El framework permite reutilizar y mantener fácilmente las pruebas.
- Los logs en Excel permiten analizar resultados de forma cómoda.