

PRUEBA TÉCNICA – AUTOMATIZADOR QA

Nombre: July Paola Chitiva Ortiz

Fecha: 14 diciembre 2025

Lugar: Bogotá

1. Introducción

Este documento presenta la solución de automatización desarrollada para la prueba técnica de QA, incluyendo la arquitectura del framework, librerías utilizadas y las instrucciones para ejecutar los módulos de pruebas API y Web. Además, se incluyen evidencias de ejecución y reportes generados.

2. Arquitectura del Framework

El framework se divide en dos módulos principales: Automatización API y Automatización Web, diseñados bajo buenas prácticas para garantizar mantenibilidad, escalabilidad y reutilización.

2.1 Módulo de Automatización API

El módulo de pruebas API está estructurado en capas que separan responsabilidades como configuración, modelos de datos, clientes de servicios y pruebas.

2.2 Módulo de Automatización Web

El módulo web utiliza el patrón Page Object Model (POM), separando la lógica de interacción con la UI de los casos de prueba.

3. Librerías Utilizadas y Por Qué

3.1 Automatización API

- RestAssured: Permite la automatización de servicios REST de forma clara y eficiente.
- JUnit/TestNG: Framework de pruebas para ejecución y aserciones.
- Maven: Gestión de dependencias y ciclo de vida del proyecto.

3.2 Automatización Web

- Playwright: Automatización de pruebas web multiplataforma.
- Node.js y npm: Gestión de dependencias y ejecución de pruebas.

4. Cómo Ejecutar Cada Módulo

4.1 Ejecución Pruebas API

1. Clonar el repositorio desde : <https://github.com/paolajpco/PagoElectronicoTest>
2. Ingresar a la carpeta api-tests
3. Ejecutar el comando: mvn clean test

4.2 Ejecución Pruebas Web

1. Ingresar a la carpeta web-tests
2. Ejecutar npm install
3. Ejecutar npx playwright install
4. Ejecutar npx playwright test

4.3 Ejecución Pruebas Web

Ejecutar todas las pruebas (API + Web)

Desde la raíz del proyecto, ejecutar:

```
mvn clean test
```

5. Evidencias

En esta sección se deben adjuntar capturas de pantalla de la ejecución de pruebas API y Web, así como reportes generados por las herramientas.

5.1 Evidencias Pruebas API

De los métodos GET, POST, PUT Y DELETE sobre la API publica <https://reqres.in> ,

La respuesta común es HTTP/1.1 403 Forbidden, El código de estado HTTP 403 Forbidden indica que el servidor recibió y entendió correctamente la solicitud, pero decide no permitir el acceso al recurso solicitado debido a reglas de autorización o seguridad.

The screenshot shows an IDE with a project named 'Prueba_Credibanco'. The 'UserApiTest.java' file is open, showing a class that extends 'ApiConfig'. The code includes a test method 'testUpdateUser()' that uses 'UserRequest.builder()' to create a request with the name 'Paola Updated'. The 'Run' console shows the test results for 'UserApiTest (api.tests)'. The test 'testUpdateUser()' failed with a status of 403 Forbidden. The console output for the failed test is as follows:

```
Path params: <none>
Headers: Accept= */*
Cookies: <none>
Multiparts: <none>
Body: <none>
HTTP/1.1 403 Forbidden
Date: Sun, 14 Dec 2025 17:46:18 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
```

5.1 Evidencias Pruebas Web

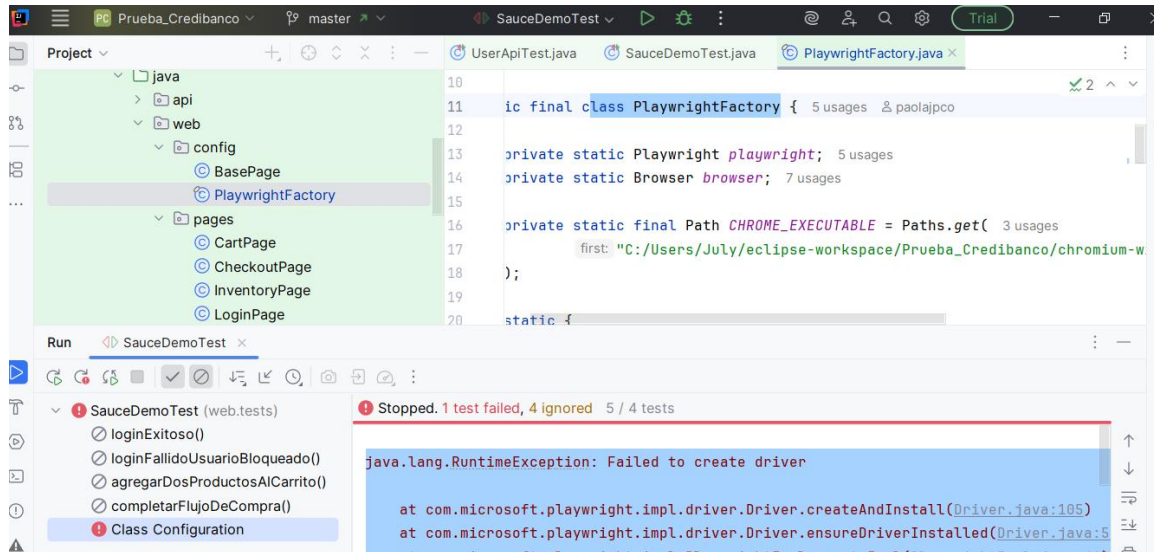
En las pruebas Web se implemento los test para

1. Login exitoso.

2. Login fallido (usuario bloqueado).
3. Agregar dos productos al carrito.
4. Completar flujo de compra.

En la ejecución se observa un fallo de ejecución al crear Driver de playwright, en mi código estoy usando un browser local de acuerdo a mi clase class PlaywrightFactory

java.lang.RuntimeException: Failed to create driver

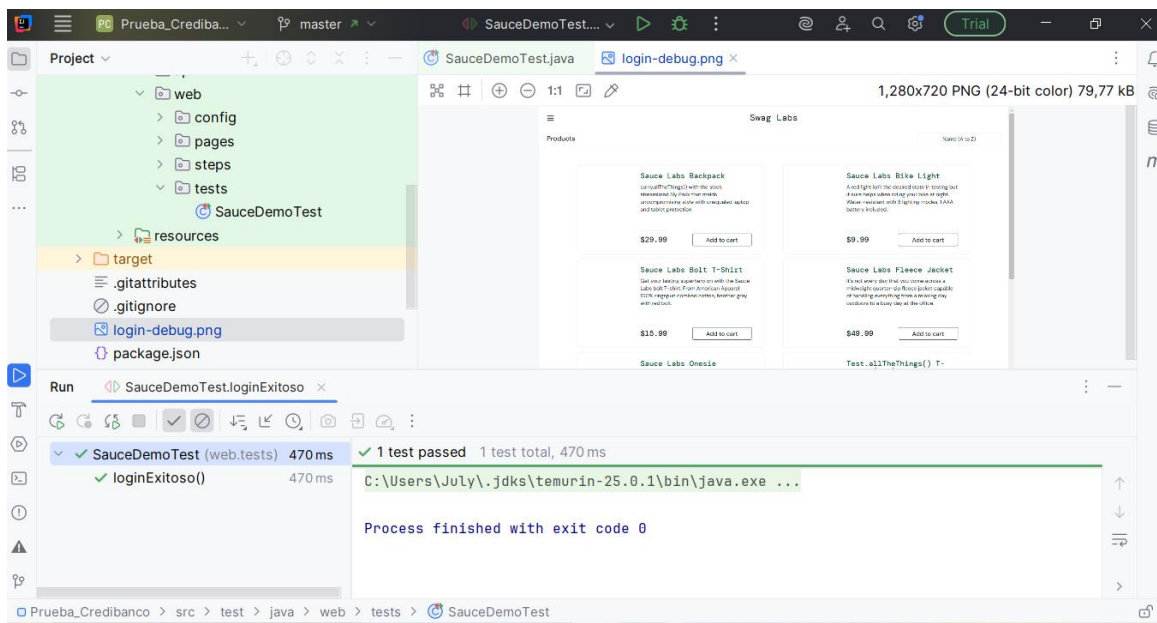
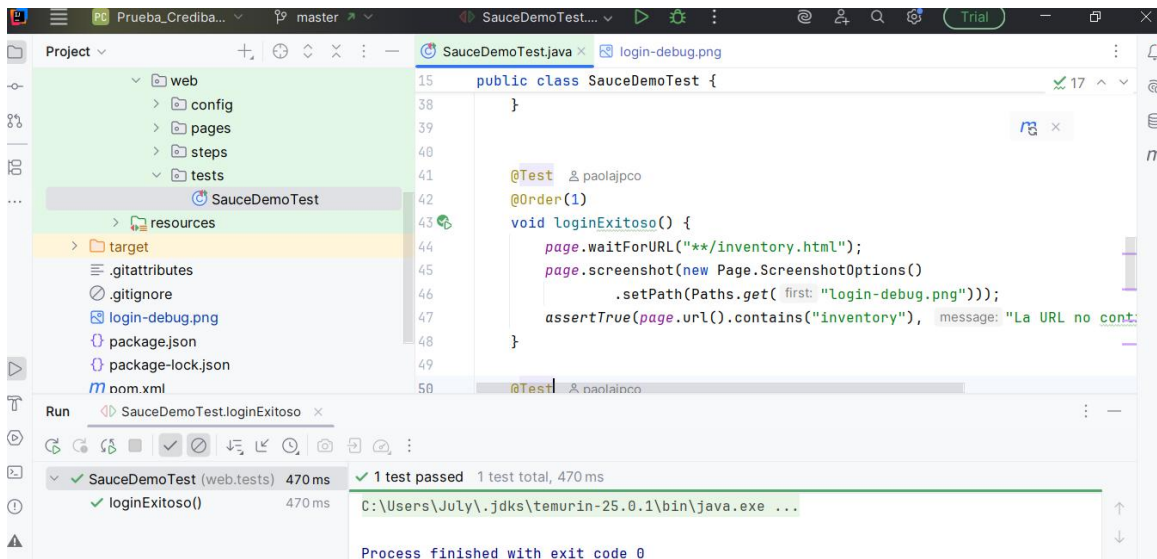


Acciones y diagnostico

Elemento	Estado
Dependencias Maven	✓ Correctas
Versiones Playwright	✓ Alineadas
Java	✓ 17 LTS
Error persiste	✗ Sí

Mediante inclusión de variable en el run del IDE

PLAYWRIGHT_SKIP_BROWSER_DOWNLOAD=1, se logra la ejecución local de los test



6. Conclusiones y Oportunidades de Mejora

La solución implementada demuestra el uso de POM Y modularidad en automatización de pruebas, cubriendo tanto servicios API y pruebas WEB evidenciando manejo de IDEs, lenguajes de programación, Lógica de programación, uso de clases, variables, funciones etc.

La separación de los módulos de pruebas API y Web facilita la mantenibilidad del proyecto y permite ejecutar las validaciones de manera independiente según la necesidad del proceso de pruebas.

Como oportunidades de mejora y evolución del framework,

Implementación del patrón Screenplay: Migrar o complementar la arquitectura actual hacia el patrón Screenplay permitiría una mayor expresividad en los casos de prueba, enfocándolos en el comportamiento del usuario y no en la interacción técnica. Esto facilitaría la lectura de los escenarios por perfiles no técnicos y mejoraría la reutilización de tareas, interacciones y preguntas.

Mejora de reportes: Integrar herramientas de reporting más visuales como Allure Report, que permitan una mejor visualización de resultados, evidencias automáticas y métricas de ejecución.

Manejo de datos dinámicos:

Implementar estrategias avanzadas de manejo de datos de prueba (data-driven testing) para aumentar la cobertura de escenarios sin duplicar código.

Pruebas de seguridad y rendimiento:

Ampliar el alcance del framework incorporando pruebas de seguridad básicas y pruebas de rendimiento para fortalecer la calidad integral del sistema.

7. Anexos

Repositorio GitHub: <https://github.com/paolajpco/PagoElectronicoTest>