

# Proyecto QA Integral

## 1. Información General

**Nombre:** Alejandro Franco Acosta

**Rol:** QA Tester

**Periodo:** Sprint 1 – Sprint 2

**Herramientas:**

- Jira/Xray
- Python
- Selenium
- Pytest
- JUnit
- XML
- MongoDB
- Mongo Compass
- SoapUI
- Git y GitHub
- Docker
- DOM (Document Object Model)

## 2. Introducción

El presente informe documenta el trabajo realizado durante el Proyecto QA Integral, cuyo objetivo es demostrar mis conocimientos aplicados al aseguramiento de calidad sobre distintas capas de un sistema, combinando pruebas manuales y automatizadas. El proyecto abarcó validaciones a nivel de API, Base de Datos y UI, manteniendo trazabilidad completa mediante Jira/Xray que es nuestra herramienta de gestión clave para mantener el orden y la planeación.

### 3. Alcance del Proyecto

Las pruebas realizadas cubren las siguientes áreas:

- **API:** Validación de servicios SOAP mediante WSDL.
- **Base de Datos:** Validaciones sobre MongoDB (estructura, consistencia y reglas básicas).
- **UI:** Automatización del flujo de login en aplicación web.
- **Gestión QA:** Registro, ejecución y trazabilidad de pruebas en Jira/Xray.

***Fuera de alcance:***

- Pruebas de performance
- Pruebas de seguridad
- Integración CI/CD

### 4. Pruebas de API (SOAP)

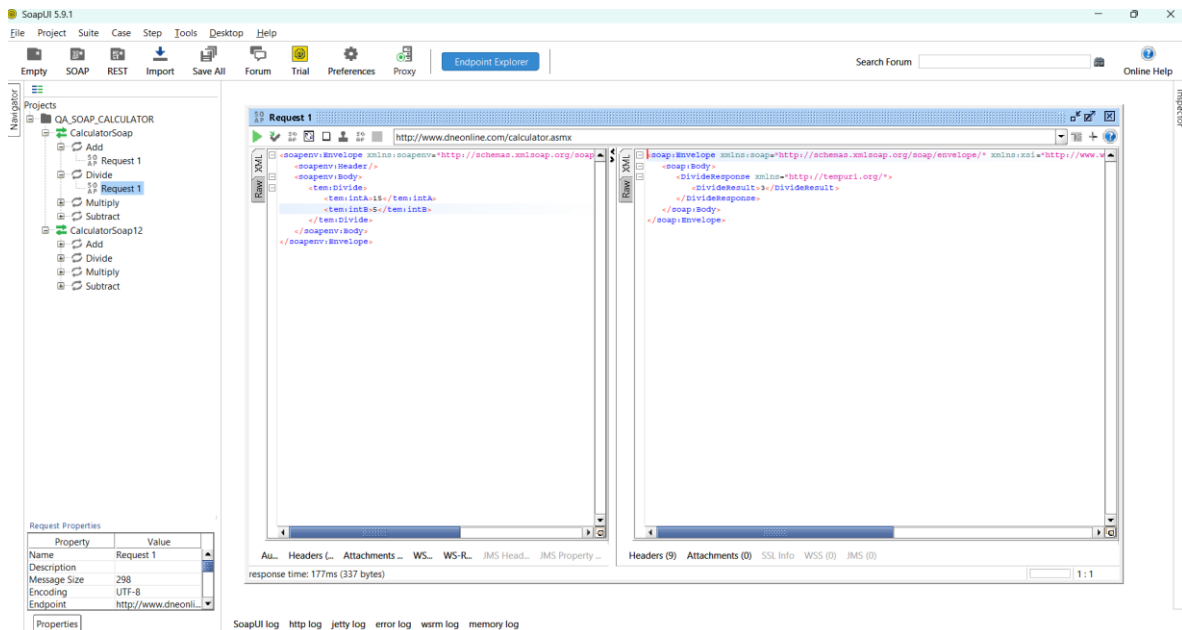
#### 4.1 Descripción

Se validó un servicio SOAP público utilizando SoapUI. A partir del WSDL se generaron operaciones y se crearon Test Suites y Test Cases para validar:

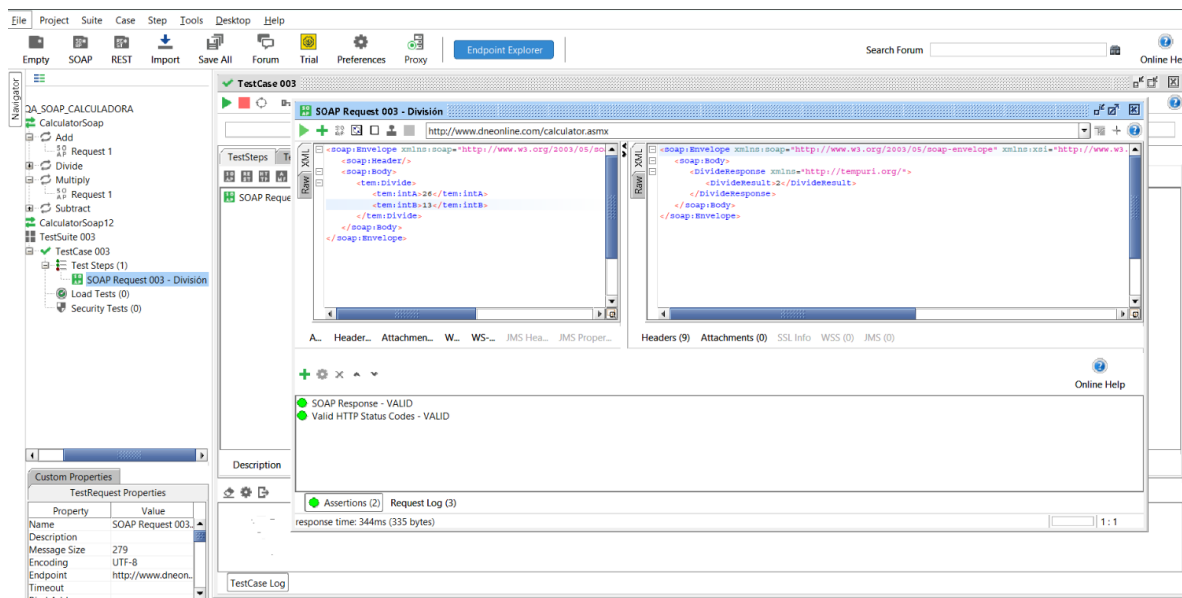
- Disponibilidad del servicio
- Código de respuesta HTTP
- Estructura del mensaje SOAP
- Resultado esperado de la operación

#### 4.2 Evidencias

- Request y Response de API en SoapUI (open source)



- Test Suite, Test Case y Assertions(contains)



- Se registra evidencias en Jira

QA2-1 / QA2-8

## 1.2 Pruebas de Servicios SOAP

**Description**  
Proyecto [GitHub - Alex-af-2024/proyecto-qa-integral](#)

**Subtasks** 100% Done

Work	Pri...	Stor...	As...	Status
QA2-9 1.2.1 Importar un WSDL en SoapUI	M		AF	DONE
QA2-10 1.2.2 Crear test suite + test case	M		AF	DONE
QA2-11 1.2.3 Crear assertions(status, XML output, reglas)	M		AF	DONE
QA2-12 1.2.4 Exportar evidencias (XML + capturas)	M		AF	DONE
QA2-13 1.2.5 Registrar caso de prueba manual en Xray	M		AF	DONE

**Linked work items**  
Add linked work item

**Details**

Assignee [AF alexandro franco](#)

Reporter [AF alexandro franco](#)

Development [Create branch](#)  
[Create commit](#)

Labels None

Due date Dec 16, 2025

Start date Dec 15, 2025

Sprint None +1

Priority Medium

Parent [QA2-11 Proyecto Integral de](#)

### 4.3 Conocimientos aplicados

- Lectura e interpretación de WSDL
- Estructura de mensajes SOAP
- Uso de assertions funcionales
- Organización de pruebas en suites y casos

### 4.4 Recomendaciones

- Versionar los proyectos SoapUI en el repositorio
- Separar datos de prueba cuando el servicio lo permita
- Documentar claramente los servicios probados

## 5. Pruebas de Base de Datos (MongoDB)

Se sugiere usar Docker mongo para mantener un entorno de pruebas limpio en el ordenador y hacer el proyecto portable.

## 5.1 Descripción

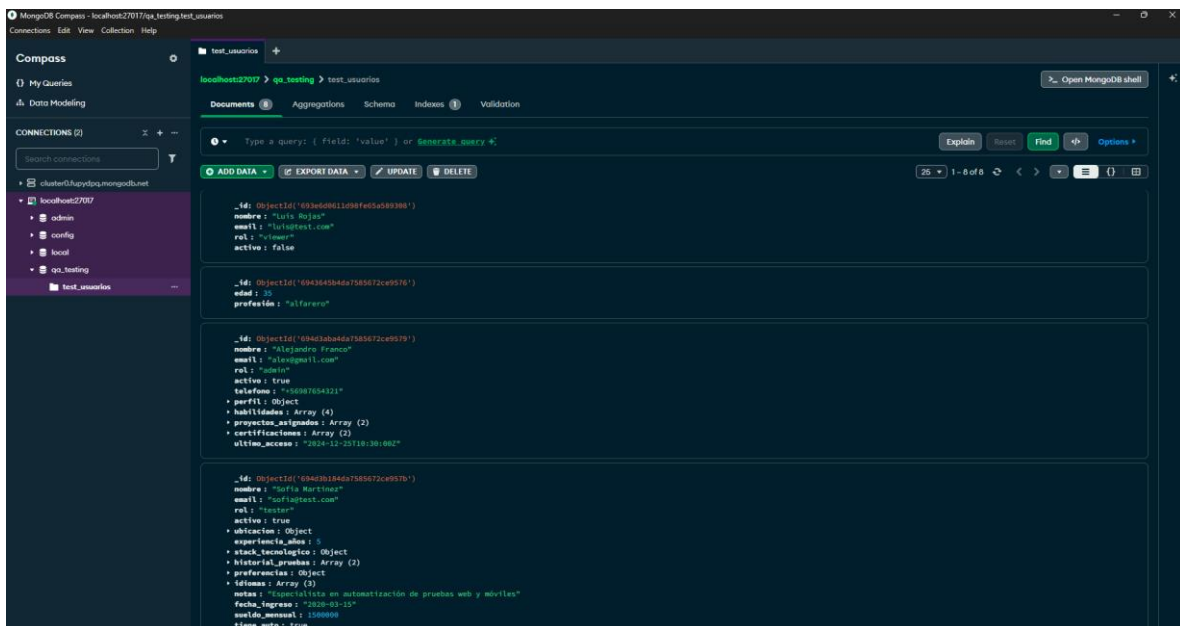
Se creó una base de datos de prueba en MongoDB para validar reglas básicas de integridad y consistencia de datos. Las validaciones se realizaron tanto de forma manual (MongoDB Compass) como automatizada mediante scripts en Python usando PyMongo.

### Las validaciones incluyeron:

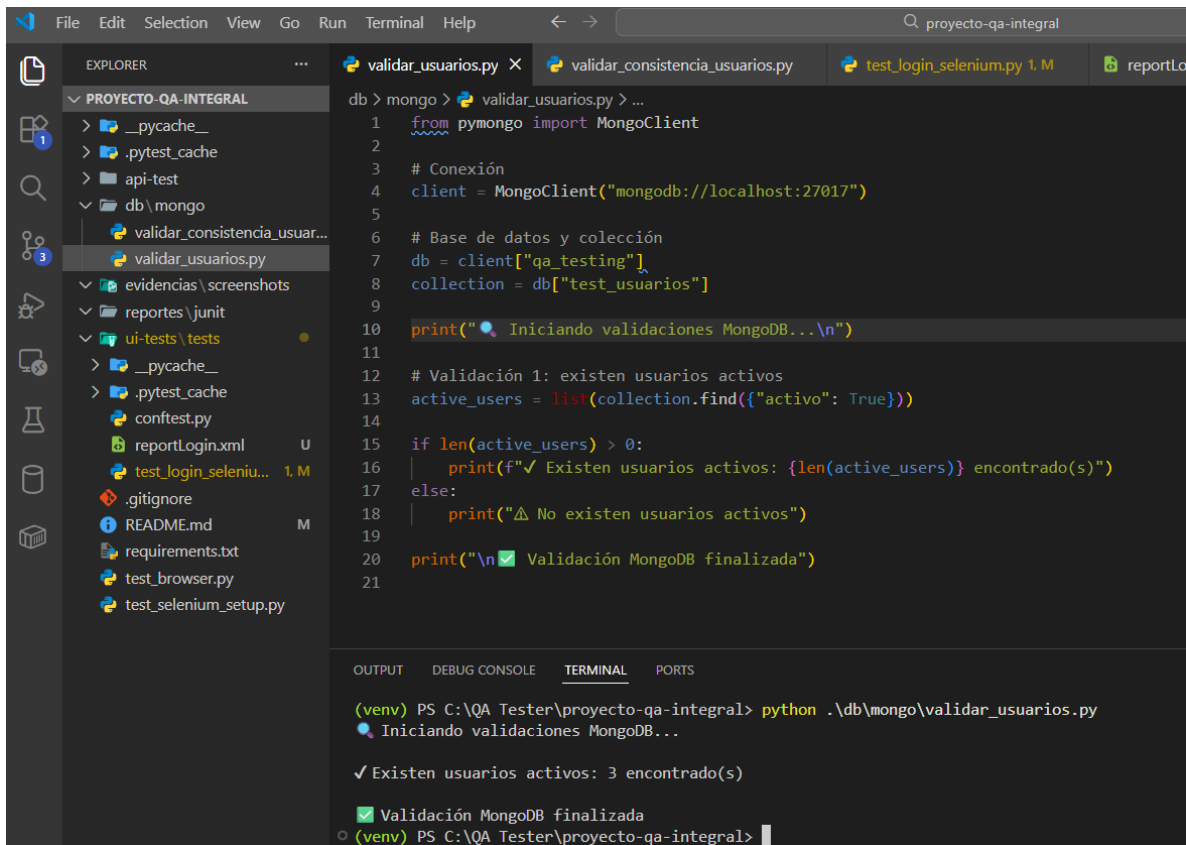
- Existencia de registros
- Validación de usuarios activos
- Validación de formato de emails
- Verificación de campos obligatorios

## 5.2 Evidencias

- Creación de base de datos local con MongoDB usando Mongo Compass como visualizador.
- Esta base de datos aprovecha el modelado flexible de documentos NoSQL.



- Ejecución de script para confirmar la existencia de usuarios activos en DDBB



The screenshot shows a Visual Studio Code editor window with the file explorer on the left and the code editor in the center. The file explorer shows a project named 'PROYECTO-QA-INTEGRAL' with various files and folders. The code editor displays a Python script named 'validar\_usuarios.py' which connects to a MongoDB database and performs a validation check. The terminal at the bottom shows the command to run the script and its output.

```
db > mongo > validar_usuarios.py > ...
1 from pymongo import MongoClient
2
3 # Conexión
4 client = MongoClient("mongodb://localhost:27017")
5
6 # Base de datos y colección
7 db = client["qa_testing"]
8 collection = db["test_usuarios"]
9
10 print("🔗 Iniciando validaciones MongoDB...\n")
11
12 # Validación 1: existen usuarios activos
13 active_users = list(collection.find({"activo": True}))
14
15 if len(active_users) > 0:
16     print(f"✓ Existen usuarios activos: {len(active_users)} encontrado(s)")
17 else:
18     print("⚠ No existen usuarios activos")
19
20 print("\n✅ Validación MongoDB finalizada")
21
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(venv) PS C:\QA Tester\proyecto-qa-integral> python .\db\mongo\validar_usuarios.py
🔗 Iniciando validaciones MongoDB...

✓ Existen usuarios activos: 3 encontrado(s)

✅ Validación MongoDB finalizada
(venv) PS C:\QA Tester\proyecto-qa-integral>
```

## 5.3 Conocimientos aplicados

- Uso de MongoDB Compass
- Modelado flexible de documentos.
- Conexión a MongoDB con PyMongo
- Validaciones lógicas mediante scripts

## 5.4 Recomendaciones

- Utilizar MongoDB en Docker para mayor portabilidad del proyecto
- Definir validaciones de esquema cuando el proyecto crezca
- Separar claramente datos de prueba y datos productivos

## 6. Pruebas UI Automatizadas (Selenium + Pytest)

### 6.1 Descripción

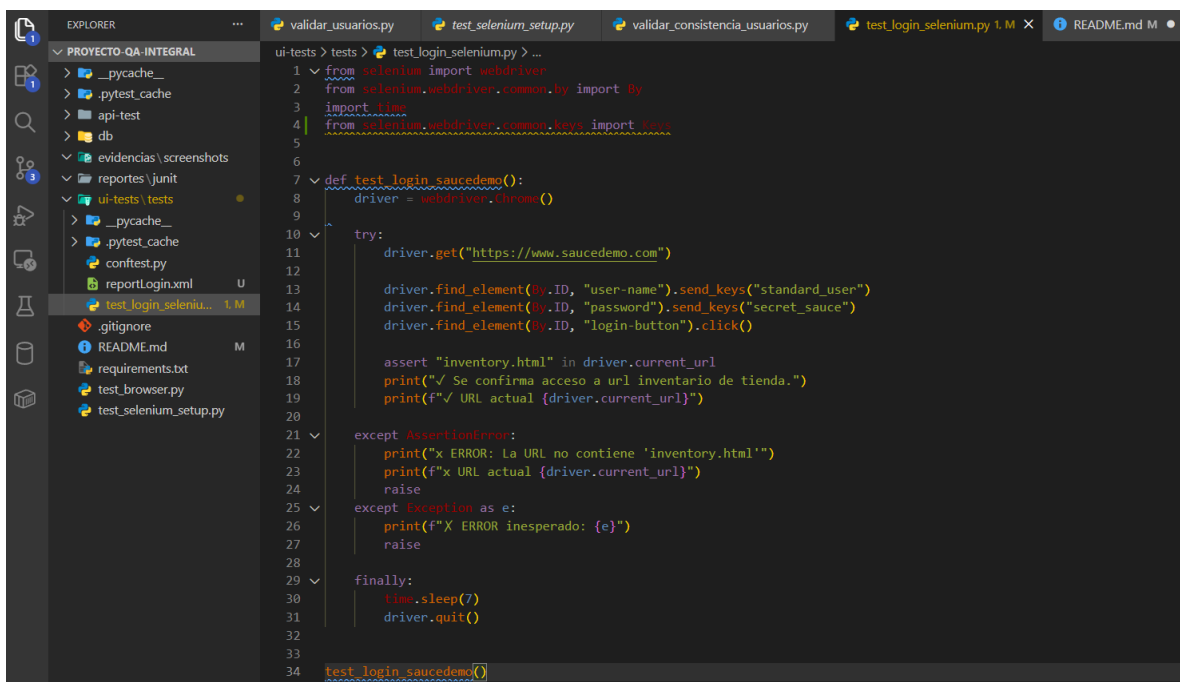
Se automatizó el flujo de login de una aplicación web utilizando Selenium WebDriver con Python y Pytest. El objetivo fue validar el acceso exitoso y comprobar la correcta redirección posterior al login.

El script incluye:

- Inicialización del navegador
- Interacción con elementos UI - DOM
- Aserciones funcionales
- Manejo básico de errores
- Captura de evidencias

## 6.2 Evidencias

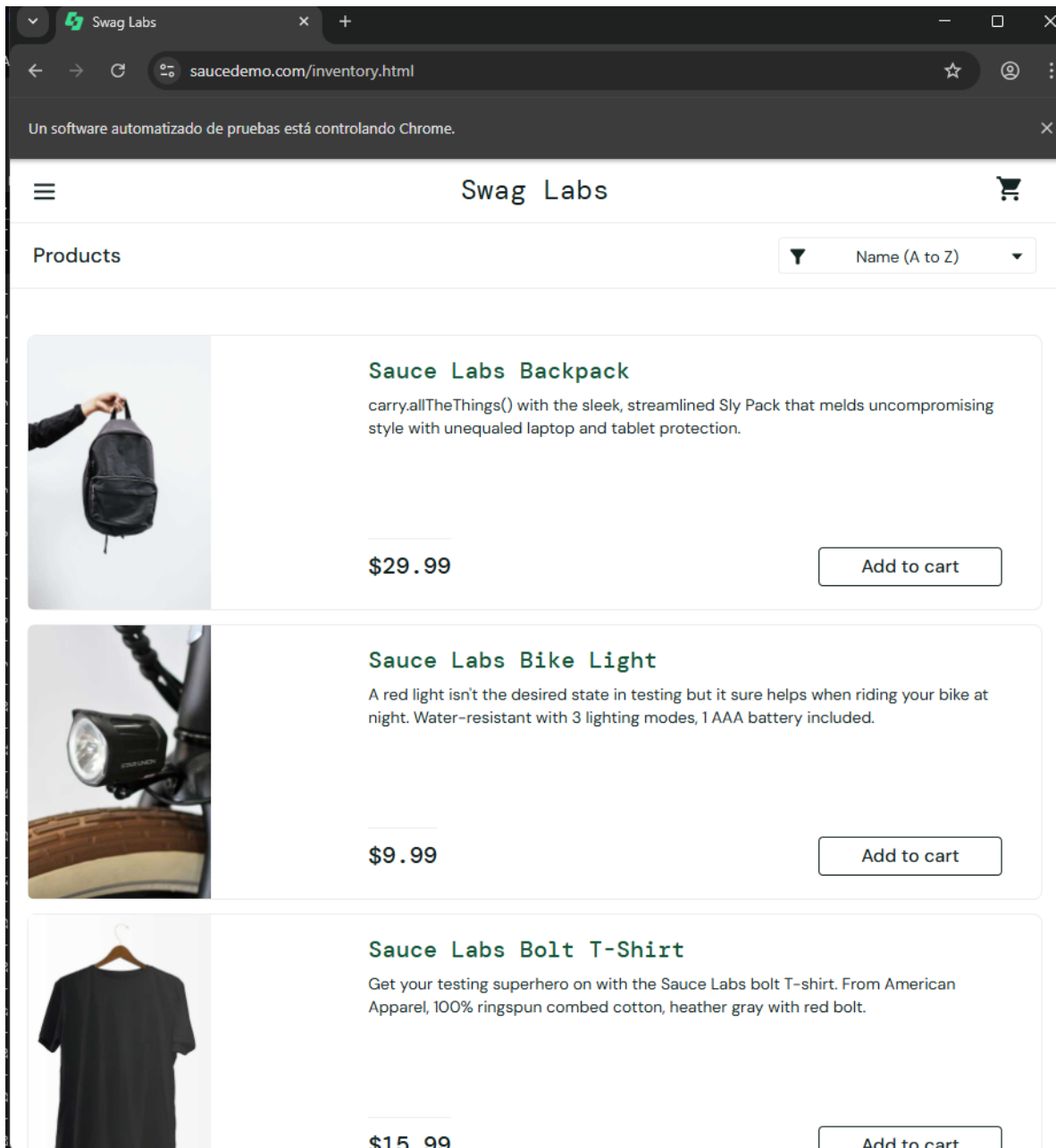
- Script de automatización para login



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'PROYECTO-QA-INTEGRAL', 'evidencias', and 'reports'. The code editor displays a Python script named 'test\_login\_selenium.py' with the following content:

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3 import time
4 from selenium.webdriver.common.keys import Keys
5
6
7 def test_login_saucedemo():
8     driver = webdriver.Chrome()
9
10    try:
11        driver.get("https://www.saucedemo.com")
12
13        driver.find_element(By.ID, "user-name").send_keys("standard_user")
14        driver.find_element(By.ID, "password").send_keys("secret_sauce")
15        driver.find_element(By.ID, "login-button").click()
16
17        assert "inventory.html" in driver.current_url
18        print("✓ Se confirma acceso a url inventario de tienda.")
19        print(f"✓ URL actual {driver.current_url}")
20
21    except AssertionError:
22        print("x ERROR: La URL no contiene 'inventory.html'")
23        print(f"x URL actual {driver.current_url}")
24        raise
25    except Exception as e:
26        print(f"x ERROR inesperado: {e}")
27        raise
28
29    finally:
30        time.sleep(7)
31        driver.quit()
32
33
34 test_login_saucedemo()
```

- Captura de acceso a página con login ejecutado con script



- Resultado usando Pytest en línea de comando

```
(venv) PS C:\QA Tester\proyecto-qa-integral> pytest .\ui-tests\tests\test_login_selenium.py -v
===== test session starts =====
platform win32 -- Python 3.13.3, pytest-9.0.2, pluggy-1.6.0 -- C:\venv_qa\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\QA Tester\proyecto-qa-integral
collected 1 item

ui-tests\tests\test_login_selenium.py::test_login_saucedemo PASSED [100%]

===== 1 passed in 22.92s =====
(venv) PS C:\QA Tester\proyecto-qa-integral>
```

## 6.3 Conocimientos aplicados



- Automatización UI con Selenium
- Uso de selectores - DOM
- Aserciones con Pytest
- Estructuración de pruebas automatizadas

#### **6.4 Recomendaciones**

- Implementar Page Object Model para mayor mantenibilidad
- Externalizar credenciales y datos sensibles
- Integrar ejecución en pipelines CI/CD

### **7. Gestión de Pruebas con Jira y Xray**

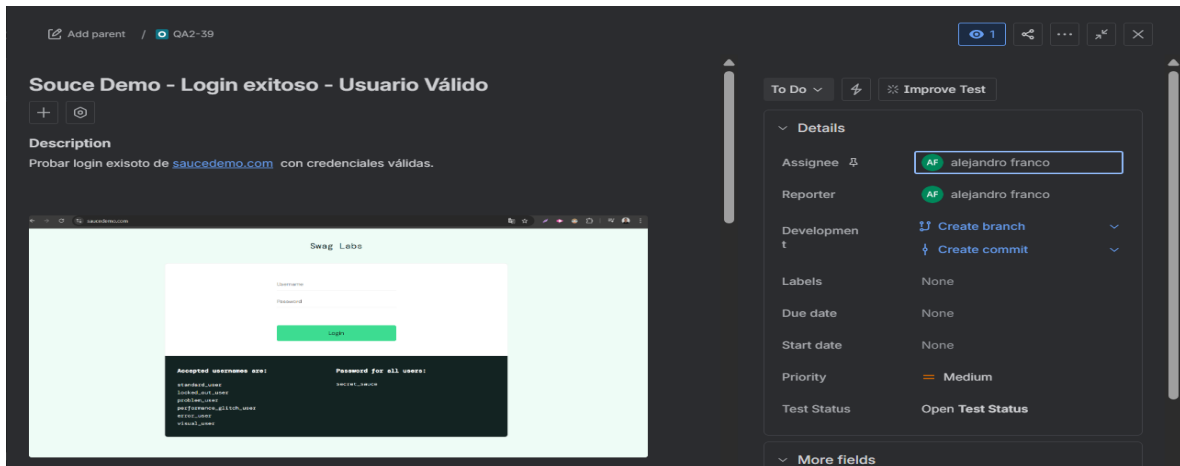
#### **7.1 Descripción**

Se utilizó Jira con el complemento Xray para:

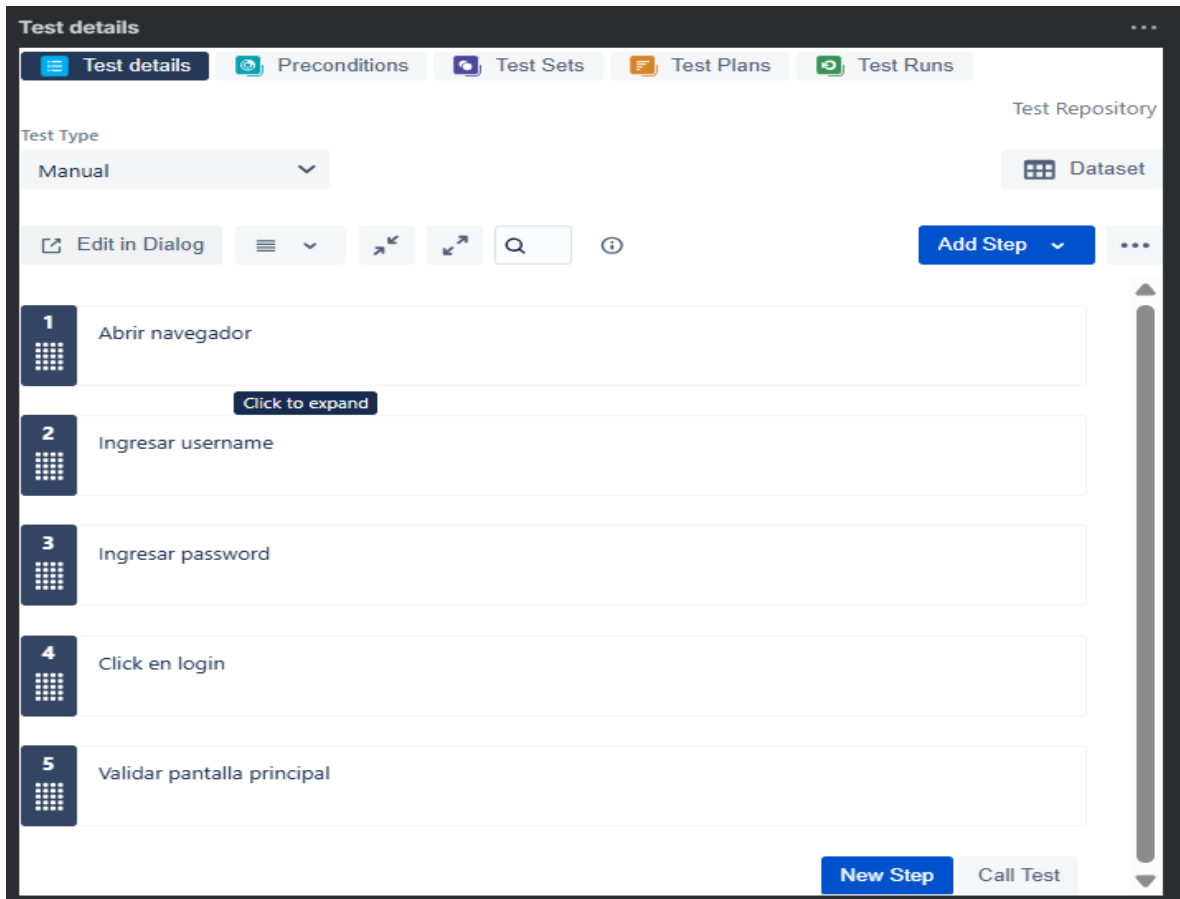
- Registrar casos de prueba manuales
- Registrar pruebas automatizadas
- Crear Test Plans y Test Executions
- Importar resultados automatizados vía XML (formato JUnit)
- Mantener trazabilidad entre tareas, tests y ejecuciones

#### **7.2 Evidencias**

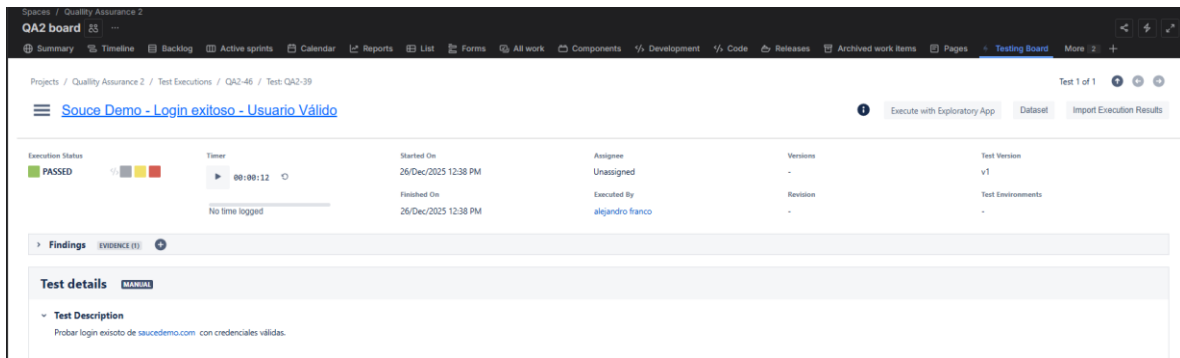
## Creación de Test con pasos detallados



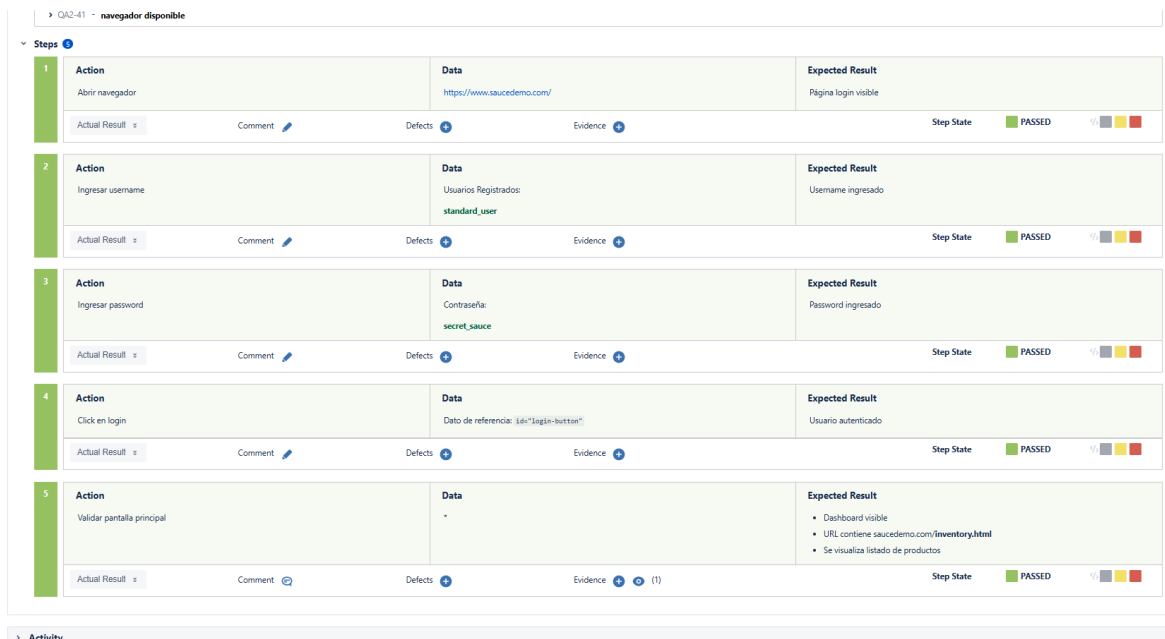
- Estructurar pasos de forma detallada y ordenada



- Se crea Test Execution y agrega Test Manual con sus pasos para su ejecución



- En la ejecución podemos registrar las evidencias según realicemos el paso a paso.



- En este caso tenemos PASSED para Test creado.

Se Se crea Test Execution y agrega XML de Test automatizado con Python

**EXE - Automatizado - Login Exitoso**

**Description**  
Add a description...

**Subtasks**  
Add subtask

**Linked work items**  
Add linked work item

**Tests**

**Overall Execution Status**

1 PASSED

**TOTAL TESTS: 1**

Rank	Key	Summary	Test Type	Dataset	#Defects	Status
1	QA2-45	test_login_saucedemo	Generic		0	PASSED

**Details**

Assignee: alejandro franco

Reporter: alejandro franco

Development: Create branch, Create commit

Labels: None

Due date: None

Start date: None

Priority: Medium

Test Plans: Open Test Plans

Test Environments: Open Test Environments

**More fields**

Original estimate: 0m

Time tracking: No time logged

Components: Add components

Sprint: None

Team: None

### 7.3 Conocimientos aplicados

- Gestión de pruebas en Jira
- Uso de Xray para trazabilidad
- Importación de resultados automatizados (Documento estándar JUnit)
- Lectura e interpretación de reportes

### 7.4 Recomendaciones

- Definir convenciones de nombres para tests
- Centralizar evidencias en Test Executions
- Usar Test Plans para ciclos de prueba formales

## 8. Conclusiones

El Proyecto QA Integral permitió aplicar un enfoque completo de aseguramiento de calidad, abarcando múltiples capas del sistema y combinando pruebas manuales y automatizadas. Se adquirieron conocimientos prácticos en herramientas ampliamente utilizadas en la industria, así como en la gestión y trazabilidad de pruebas.

El proyecto deja una base sólida para evolucionar hacia prácticas más avanzadas como CI/CD, pruebas de performance y automatización a mayor escala.