B2B SaaS Platform Testing - Multi-Platform Automation

1. **Candidate** : Akshay Pawar
2. **Role** : Automation Engineering Intern

Overview

You're joining the QA team at "WorkFlow Pro" - a B2B project management SaaS platform. The platform serves multiple clients (multi-tenant), works across web browsers and mobile devices, and integrates with various third-party services.

Part 1: Debugging Flaky Test Code

A previous intern wrote this Playwright test for user login functionality. **The test is flaky** - sometimes passes, sometimes fails in CI/CD pipeline.

```
import pytest
from playwright.sync_api import sync_playwright

def test_user_login():
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()

        # Navigate to login page
        page.goto("https://app.workflowpro.com/login")

        # Fill login form
        page.fill("#email", "admin@company1.com")
        page.fill("#password", "password123")
        page.click("#login-btn")

        # Verify successful login
        assert page.url == "https://app.workflowpro.com/dashboard"
        assert page.locator(".welcome-message").is_visible()

        browser.close()

def test_multi_tenant_access():
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()

        page.goto("https://app.workflowpro.com/login")
        page.fill("#email", "user@company2.com")
        page.fill("#password", "password123")
        page.click("#login-btn")

        # User should only see Company2 data
        projects = page.locator(".project-card").all()
        for project in projects:
            assert "Company2" in project.text_content()

        browser.close()
```

A. Issues That Can Cause Flaky Tests

After reviewing the given Playwright test, I noticed multiple reasons why the test can behave inconsistently:

- No Explicit waits
  - The test does not wait for the page to fully load after login
- URL assertion immediately after click
  - URL assertion is done immediately after clicking the login button
- Dynamic dashboard elements
  - Dashboard elements load dynamically, but no wait is applied
- 2FA not handled
  - Some users have 2FA enabled, which is not handled in the test
- Hardcoded credentials
  - Test assumes same performance locally and in CI
- No viewport / browser config
  - No browser or screen size configuration is defined
- .all() used without wait
  - .all() is used without ensuring elements are visible

B. Why These Issues Appear in CI/CD

These problems happen more often in CI/CD because:

a. CI servers are slower than local machines
b. Headless browsers behave differently than headed browsers
c. Network latency is higher in CI
d. CI may run tests on different browsers and screen size
e. Dashboard data depends on tenant-specific loading time

C. Improved and Stable Test Code

I fixed the test by adding explicit waits, stable assertions, and reusable fixtures.

CODE:

```
import pytest
from playwright.sync_api import sync_playwright, expect
@pytest.fixture

def page():
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=True)
        context = browser.new_context(viewport={"width": 1280, "height": 720})
        page = context.new_page()
        yield page
        browser.close()

def test_user_login(page):
    page.goto("https://app.workflowpro.com/login")
```

```
    page.fill("#email", "admin@company1.com")
    page.fill("#password", "password123")
    page.click("#login-btn")

#wait for dashboard redirect
    page.wait_for_url("**/dashboard", timeout=10000)

#wait for dynamic dashboard load
    expect(page.locator(".welcome-message")).to_be_visible(timeout=10000)

def test_multi_tenant_access(page):
    page.goto("https://app.workflowpro.com/login")

    page.fill("#email", "user@company2.com")
    page.fill("#password", "password123")
    page.click("#login-btn")

#wait until projects load
    page.wait_for_selector(".project-card", timeout=10000)

    projects = page.locator(".project-card")
    for i in range(projects.count()):
        expect(projects.nth(i)).to_contain_text("Company2")
```

Part 2: Test Framework Design (25 minutes)

1. Design a test automation framework structure for this B2B SaaS platform. **Note**: Requirements are intentionally incomplete.
2. **Given Requirements:**

- Support web (Chrome, Firefox, Safari) and mobile (iOS, Android) testing
- Handle multiple tenant environments (company1.workflowpro.com, company2.workflowpro.com)
- Test different user roles (Admin, Manager, Employee) with varying permissions
- API testing for backend services
- Integration with BrowserStack for cross-platform testing
- CI/CD pipeline integration

1. **Your Tasks:**
2. **Framework Structure**: Design folder structure, base classes, and utilities
3. **Configuration Management**: How to handle multiple environments, browsers, and test data
4. **Identify Missing Requirements**: What questions would you ask about test data management, reporting, parallel execution, etc.?
5. **Format**: Use text/pseudocode/diagrams to explain your design

A. Framework Structure

I would design a modular framework to support web, mobile and API testing.

```
automation-framework/
│
├── tests/
│   ├── ui/
│   ├── api/
│   ├── integration/
│
├── pages/
│   ├── login_page.py
│   ├── dashboard_page.py
│
├── api_clients/
│   ├── project_api.py
│
├── utils/
│   ├── auth.py
│   ├── test_data.py
│   ├── wait_helpers.py
│
├── config/
│   ├── config.yaml
│   ├── tenants.yaml
│
├── fixtures/
│   ├── browser.py
│   ├── api_fixtures.py
│
├── reports/
│
└── pytest.ini
```

B. Configuration Management

- Config.yaml (Environment details stored in YAML files)
  - Yaml
    env: staging
    browser: chromium
    headless: true

- Tenants.yaml (Tenants-specific URLs separated from test logic)
  - Yaml
    company1:
      base_url: https://company1.workflowpro.com

Company2:
base_url: https://company2.workflowpro.com

- Environment variables:
  - Tokens
  - BrowserStack credentials
  - Secrets

C. Missing Requirements / Questions I Would Ask

1. How should test data be cleaned after execution?
2. Can automation bypass 2FA in non-production?
3. How many tests should run in parallel?
4. Which reporting tool should be used?
5. BrowserStack device limits?

Part 3: API + UI Integration Test (35 minutes)

Create a test that validates the complete flow of project creation through both API and UI layers.

1. **Business Scenario:**
2. Create a project via API
3. Verify project appears correctly in web UI
4. Check project is accessible on mobile
5. Validate proper tenant isolation (project only visible to correct company)
6. **API Endpoint Information:**

POST /api/v1/projects
Headers: Authorization: Bearer {token}, X-Tenant-ID: {company_id}
Body: {"name": "Test Project", "description": "...", "team_members": [...]}
Response: {"id": 123, "name": "Test Project", "status": "active"}

1. **Your Tasks:**
2. **Write Integration Test**: Combine pytest (API) + Playwright (UI) + BrowserStack mobile testing
3. **Handle Test Data**: Show how you'd manage test data across API/UI
4. **Cross-Platform Validation**: Ensure the test works on multiple browsers and mobile devices
5. **Tenant Isolation**: Verify security boundaries between companies
6. **Expected Test Structure:**

```
def test_project_creation_flow():
    # 1. API: Create project
    # 2. Web UI: Verify project display
    # 3. Mobile: Check mobile accessibility
    # 4. Security: Verify tenant isolation
    Pass
```

12. **Your Tasks:**
13. **Implement the test** using pytest + Playwright + BrowserStack concepts
14. **Handle edge cases** (network failures, slow loading, mobile responsiveness)
15. **Explain approach** with comments on your testing strategy
16. **Hints**: You'll need to make assumptions about authentication, test data cleanup, and mobile testing setup.

A. Test Strategy

- API used for fast setup
- UI validates real user behavior
- Mobile validates responsiveness
- Tenant isolation ensures security

B. Integration Test Example

CODE:

```
def test_project_creation_flow(api_client, page, mobile_driver):
    # 1. API: Create project
    project = api_client.create_project(
        name="Automation Test Project",
        tenant_id="company1"
    )
    project_id = project["id"]

    # 2. Web UI: Verify project
    page.goto("https://company1.workflowpro.com/dashboard")
        page.wait_for_selector(".project-card")
        expect(page.locator(".project-card")).to_contain_text("Automation Test
Project")

    # 3. Mobile Validation (BrowserStack)
        mobile_driver.get("https://company1.workflowpro.com/dashboard")
        assert "Automation Test Project" in mobile_driver.page_source

    # 4. Tenant Isolation Check
        page.goto("https://company2.workflowpro.com/dashboard")
        assert "Automation Test Project" not in page.content()
```

C. Edge Case Handling

- Retry logic for API failures
- Slow network response
- Screenshot & logs on failure
- Cleanup project post-test

D. Assumptions Made

- Token-based auth available
- API cleanup endpoints exist
- BrowserStack devices pre-configured
- Separate tenants for testing

E. Conclusion

- This approach ensures:
    - Stable automation
    - Scalable framework
    - Real-world SaaS testing coverage
    - Secure multi-tenant validation

F. Final Thoughts

- I have faced similar flaky issues during my learning and tried to solve them using explicit waits.
- I focused on writing automation that is reliable in CI, easy to maintain, and suitable for a multi-tenant SaaS product.
- My goal was to reduce flaky tests while ensuring realistic coverage across web, mobile, and API layers.
- I focused on maintainability, CI reliability, and business risk reduction, not just passing tests.
