B2B SaaS Platform Testing - Multi-Platform Automation

Part 1: Debugging Flaky Test Code

Flakiness Issues

- Hardcoded URL Assertions may fail due to redirects, trailing slashes, or query parameters.
- No Waits for Dynamic Elements: Dashboard elements load asynchronously; immediate assertions cause failures.
- 2FA Handling Missing: Some users require 2FA, test does not cover it.
- CI vs Local Differences: Timing issues, network latency, browser performance, viewport sizes.
- Direct `.all()` without waits: `projects = page.locator('.project-card').all()` may return empty before elements load.

Root Causes

- Dynamic Loading: CI environments are slower; lack of waits causes intermittent failures.
- Tenant-Specific Loading Times: Different data volumes per tenant.
- Auth Flow Differences: 2FA not handled consistently.
- Browser/Viewport Differences: Assertions may fail due to different rendering behaviors.

Fixed Test Code

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

@pytest.fixture(scope="session")
def browser():
    with sync_playwright() as p:
        browser = p.chromium.launch(headless=True)
        yield browser
        browser.close()

@pytest.fixture()
def page(browser):
    context = browser.new_context()
    page = context.new_page()
    yield page
    context.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 redirect & dashboard element
    expect(page).to_have_url(lambda url: "/dashboard" in url)
    expect(page.locator(".welcome-message")).to_be_visible()

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 for projects to load
    projects = page.locator(".project-card")
    expect(projects.first).to_be_visible()

    for i in range(projects.count()):
        assert "Company2" in projects.nth(i).text_content()
```

Part 2: Test Framework Design

Framework Structure

```
/tests
  /ui
    /web
    /mobile
  /api
  /integration
/framework
  /base
    base_test.py
    base_api.py
    base_ui.py
  /pages
    login_page.py
    dashboard_page.py
    project_page.py
  /utils
```

```
    config.py
    helpers.py
    browserstack.py
  /data
    test_data.json
    tenants.json
  /reports
pytest.ini
requirements.txt
```

Configuration Management

- Multiple Environments: config.yaml per environment (staging, prod, etc.).
- Browsers/Devices: Configurable via CLI args (--browser=chrome --device=iphone13).
- Test Data: Centralized JSON/YAML files per tenant.
- Secrets: Stored in environment variables / vault.

Missing Requirements (Questions)

- Test Data Management: Should we reset tenant DB or create new data per run?
- Reporting: Do we need Allure/HTML reports with screenshots & API logs?
- Parallel Execution: What is CI infra capacity? Run tests per tenant in parallel?
- 2FA Bypass: Can we disable 2FA for automation users?
- Cleanup: Should created test projects be deleted post-run?

Part 3: API + UI Integration Test

```
import pytest
import requests
from playwright.sync_api import sync_playwright, expect

API_BASE = "https://api.workflowpro.com/api/v1"
WEB_BASE = "https://app.workflowpro.com"
MOBILE_CAPS = {"device": "iPhone 13", "os": "iOS", "realMobile": "true"}  # BrowserStack example

@pytest.fixture(scope="session")
def api_token():
  # Assume we have a login endpoint to fetch token
  return "FAKE_API_TOKEN"

def test_project_creation_flow(api_token):
  tenant_id = "company1"

  # 1. API: Create project
  payload = {"name": "Test Project", "description": "UI/API validation", "team_members":
```

```
["user1"]}
    headers = {"Authorization": f"Bearer {api_token}", "X-Tenant-ID": tenant_id}
    response = requests.post(f"{API_BASE}/projects", json=payload, headers=headers)
    assert response.status_code == 200
    project_id = response.json()["id"]

    # 2. Web UI: Verify project display
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()
        page.goto(f"{WEB_BASE}/login")
        page.fill("#email", "admin@company1.com")
        page.fill("#password", "password123")
        page.click("#login-btn")

        projects = page.locator(".project-card")
        expect(projects.filter(has_text="Test Project")).to_be_visible()
        browser.close()

    # 3. Mobile: Check accessibility via BrowserStack
    # (pseudocode - assume BrowserStack integration utility exists)
    # mobile_driver = browserstack.launch(MOBILE_CAPS)
    # mobile_driver.login(tenant_id)
    # assert mobile_driver.find("Test Project")

    # 4. Tenant Isolation
    headers_other = {"Authorization": f"Bearer {api_token}", "X-Tenant-ID": "company2"}
    resp_other = requests.get(f"{API_BASE}/projects/{project_id}", headers=headers_other)
    assert resp_other.status_code == 403  # Forbidden
```

Testing Strategy

- API First: Verify backend before UI.
- UI Verification: Use waits to handle dynamic loads.
- Cross-Platform: Run on Chrome + iOS + Android via BrowserStack.
- Isolation: Always validate project visibility restricted to correct tenant.
- Resilience: Handle retries for network failures, add screenshots on UI failures.
- Cleanup: Optionally delete project after run.

Assumptions Made

- 2FA can be disabled for automation users.
- API tokens are reusable within session.
- Test tenants have isolated datasets.

- BrowserStack access & credentials available.
- Project deletion endpoint exists for cleanup.