# QA Automation Case Study- Solution

**Candidate: Varsha Chavan**

Part 1:Debugging Flaky Playwright Test Code

**Identify Flakyness Issues:**

The Following Issues Can cause Intermittent Failure:

1) No Explicit Waits:
   -After Clicking login, The test immediatlly checks URL and elements.
   -Dashboards Loads Dynamically - Elements May not be ready.

2) Hard Assertion on exact URL:
   -Redirect may take time or include query params.
   -CI Environments are slower than local.

3) No Handlling of 2FA:
   -Some users may triggers 2FA->Login flow changes.

4) Single Browser Context:
   -No isolation between Tests.
   -Session leakage possible.

5) Headless vs Headed Difference:

- CI Usually runs headless->timing and rendering diffrences.

6) No Timeout Configuration:
-Default Timeout maybe insufficient for CI.

7) Dynamic Tenant Loading:
-Difftrent tenants have different load times.
-Project cards may appear late.

**2) Why These Fail CI/CD but Pass Locally**

1) Reason:Network Speed
CI/CD:Slower
Local: Faster

2) Browser Mode:
CI/CD:Headless
Local:Headed

3) Machine Load:
CI/CD:High
Local:Low

4) Screen Size:
CI/CD:High
Local:Low

5) Screen Size:
CI/CD:Varies
Local:Stable

6) Parallel Execution:
CI/CD:Common
Local: Rare

CI exposes **timing and synchronization issues**, which local runs often hide.

**3) Corrected 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()

       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 URL
    page.wait_for_url("**/dashboard", timeout=15000)

    # Wait for dynamic dashboard element
    welcome = page.locator(".welcome-message")
    expect(welcome).to_be_visible(timeout=15000)

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")

    page.wait_for_url("**/dashboard")

    # Wait until projects load
    page.wait_for_selector(".project-card", timeout=20000)

    projects = page.locator(".project-card")
    count = projects.count()

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

**Improvements Made:**

1) Explicit Waits
2) Browser Context Isolation
3) Pattern-Based URL Checks

4) Dynamic Loading Handled
5) CI-Safe execution

**Part-2: Test Automation Framework Design:**

1) Framework Structure:

```
automation-framework/
|
├── tests/
|   ├── ui/
|   |   ├── test_login.py
|   |   ├── test_projects.py
|   ├── api/
|   |   ├── test_projects_api.py
|   ├── integration/
|       ├── test_project_flow.py
|
├── pages/
|   ├── login_page.py
```

```
|   ├── dashboard_page.py
|   ├── project_page.py
|
├── api_clients/
|   ├── project_api.py
|
├── config/
|   ├── env.yaml
|   ├── browserstack.yaml
|
├── utils/
|   ├── auth.py
|   ├── test_data.py
|   ├── wait_helpers.py
|
├── fixtures/
|   ├── browser.py
|   ├── api_fixtures.py
```

│

├── reports/

│

├── pytest.ini

└── requirements.txt

## 2)Configuration Management:

Environment config → env.yaml

company1:

base_url: https://company1.workflowpro.com

company2:

base_url: https://company2.workflowpro.com

**Browser And Device Selection:**

- pytest markers (@pytest.mark.mobile)
- BrowserStack capabilities

**User Roles:**

users = {

"admin": {"email": "...", "role": "Admin"},

```
  "manager": {...}
}
```

**2)Missing Requirements(Questions To Ask):**

1. How is **test data reset** after execution?
2. Do we need **parallel execution limits** (BrowserStack cost)?
3. Required **reporting format** (Allure / HTML)?
4. How is **2FA bypassed** in test environments?
5. Are **API rate limits** enforced?

## Part 3: API + UI Integration Test:

Assumptions**:**

1) Auth token available

2) Test tenant IDs known

3) Cleanup allowed after test

4) BrowserStack credentials configured

## Integration Test Implementation:

```
import requests
```

```
from playwright.sync_api import expect


def test_project_creation_flow(api_token, page,
mobile_page):
    # 1. API: Create project
    headers = {
        "Authorization": f"Bearer {api_token}",
        "X-Tenant-ID": "company1"
    }

    payload = {
        "name": "Test Project",
        "description": "API + UI Test",
        "team_members": []
    }

    response = requests.post(
        "https://api.workflowpro.com/api/v1/projects",
```

```
        json=payload,
        headers=headers
    )

    assert response.status_code == 200
    project_id = response.json()["id"]

    # 2. Web UI validation
page.goto("https://company1.workflowpro.com/dashboard")
    page.wait_for_selector(".project-card")

    project = page.locator(f"text=Test Project")
    expect(project).to_be_visible()

    # 3. Mobile validation (BrowserStack)
mobile_page.goto("https://company1.workflowpro.com/dashboard")
```

```
    expect(mobile_page.locator("text=Test
Project")).to_be_visible()


    # 4. Tenant isolation


page.goto("https://company2.workflowpro.com/dashboard")

    expect(page.locator("text=Test
Project")).not_to_be_visible()
```

**Edge Case Handlling Strategy:**

| **Scenario** | **Handlling** |
|---|---|
| 1) Slow Network | Increased timeouts+retries |
| 2) API Failure | Assertion+logging |
| 3) Mobile rendering | Responsive validation |
| 4) Tenant leakage | Negative assertion |

**Testing Strategy Explaination:**

API used for fast setup

UI used for real user validation

Mobile confirms cross-platform

Negative test ensures security

Cleanup done via API

This mirrors real production testing.

**Thank You…….**