Test Plan: WorkFlow Pro (B2B SaaS Platform)

Document Version: 1.0

Author: Mitanshu D. Shinde (mitanshushinde9@gmail.com)

Date: January 28, 2026 (Asia/Kolkata)

1. Purpose

This Test Plan defines the testing strategy, scope, resources, schedule, and deliverables for the WorkFlow Pro B2B SaaS platform. It covers web (Chrome, Firefox, Safari) and mobile (iOS, Android) testing, multi-tenant validation, API and UI integration, automation framework design, and CI/CD integration.

2. References

QA Automation Engineering Case Study (take-home + live): See attached case study for tasks and assumptions.

Source file: QA Automation Engineering Case Study.docx. □filecite□turn0file0□

3. Scope

In scope:

- Functional testing of login, tenant isolation, project creation and listing, roles & permissions, and integrations.

- API testing for core endpoints (auth, projects, users).

- Cross-browser compatibility (desktop browsers) and responsive/mobile validation.

- Automation for regression, smoke, and API+UI integration flows.

Out of scope:

- Deep performance/stress testing of entire platform (can be a separate plan).

- Third-party systems beyond contractually required validation (detailed vendor testing is separate).

4. Assumptions & Constraints

- Test environments (staging/preprod) will closely mirror production domain names and multi-tenant routing.

- Test accounts for roles (Admin, Manager, Employee) across tenants are pre-provisioned.

- BrowserStack account and credentials are available for cross-platform runs.

- CI runner has network access to BrowserStack and staging environment.

- Some users require 2FA; test accounts will either have 2FA disabled or test bypass configured.

5. Risks & Mitigations

- Flaky UI tests due to dynamic load: use explicit waits, stable selectors, and retries.

- Tenant data leakage risk: adopt dedicated test tenants + data cleanup.

- CI environment differences: standardize browser versions, viewport sizes, and timeouts; maintain test lab infra.

- BrowserStack/API rate limits: schedule runs and add throttling/backoff in tests.

6. Test Strategy

Testing types and approach:

- Sanity/Smoke: quick checks on deployment (login, health endpoints, basic navigation).

- Functional: UI and API tests for business flows (projects, users, permissions).

- Integration: API + UI end-to-end tests (sample: create project via API -> verify UI -> verify mobile).

- Regression: automated suite executed on each release/merge to main.

- Compatibility: BrowserStack matrix for browsers and mobile devices.

- Security & Tenant Isolation: tests to verify cross-tenant access is denied.

- Accessibility: basics (aria roles, contrast) for major pages.

- Exploratory: for new features and complex edge-cases.

7. Test Environments

- Staging (staging.workflowpro.com) – primary for automation.

- Preprod (preprod.workflowpro.com) – for final acceptance tests.

- BrowserStack project for cross-browser/device runs.

- Authentication: use dedicated test OAuth clients or API tokens; test 2FA bypass accounts for automation.

8. Roles & Responsibilities

- QA Lead: plan approval, test metrics, release gate decisions.

- Automation Engineer: framework, test implementation, CI integration.

- Manual QA: exploratory, usability, complex cases.

- DevOps: environment provisioning, CI runners, BrowserStack access.

- Product Owner: accept/reject major test scope changes.

9. Entry and Exit Criteria

Entry Criteria:

- Build deployed to staging with release notes.

- Required test accounts and test data provisioned.

- Smoke tests configured and passing.

Exit Criteria:

- All critical/severe defects resolved or accepted.

- Regression suite pass rate >= 95% (threshold configurable).

- Sign-off by QA Lead and Product Owner.

10. Test Deliverables

- Test Plan document (this document).

- Test cases and automated scripts in repository.

- Test data artifacts and cleanup scripts.

- CI pipeline jobs and test reports (HTML/Allure).

- Test closure report and metrics dashboard.

11. Test Data Management Strategy

- Use tenant-scoped test accounts and unique prefixes for entities to avoid collisions.

- API-based setup and teardown for test data (fixtures).

- Use idempotent create-or-update calls where possible.

- Maintain a test-data catalog that lists required accounts, roles, and tokens.

- Schedule periodic cleanup jobs and run deterministic teardown after tests.

12. Automation Strategy & Framework Design

Recommended stack: pytest + Playwright (Python) for web, Playwright + Appium/BrowserStack for mobile, requests/HTTPX for API tests.

Key principles: modular, reusable, fast, observable, and CI-friendly.

Folder structure (example):

- tests/

  - api/

  - ui/

  - mobile/

  - integration/

- framework/

  - drivers/ # browser and mobile driver factory

  - pages/   # Page Objects (Playwright)

  - services/ # API clients, auth helpers

  - fixtures/ # pytest fixtures for env, tokens, data

  - utils/   # retry, wait, logging, selectors, config loader

- config/ # yaml/json for environments and BrowserStack capabilities

- reports/ # test result outputs (Allure, JUnit XML)

13. Configuration Management

- Centralized config file per environment (config/staging.yaml, config/preprod.yaml).

- Parameterize by ENV, BROWSER, TENANT_ID, ROLE, and DEVICE.

- Secrets stored in CI secret manager or Vault (tokens, BrowserStack creds).

- Use pytest CLI switches and environment variables to choose runs.

14. Handling Flaky Tests

- Use deterministic waits: prefer locator.wait_for(state="visible") over time.sleep.

- Add retry-at-test-level (pytest-rerunfailures) for known flaky tests, but only as temporary measure.

- Capture full screenshots, DOM snapshots, and network logs on failure for triage.

- Tag flaky tests and schedule dedicated stabilization work.

- Stabilize selectors (data-test-id attributes).

15. CI/CD Integration

- Trigger suites on pull requests and nightly:

  - Quick: smoke on PR (short time budget).

  - Full: nightly regression with broader BrowserStack matrix.

- Publish artifacts: test reports, screenshots, videos (from Playwright/BrowserStack).

- Gate releases: block merge if critical failures found (configurable).

16. Test Execution & Schedule

- Daily/nightly: full regression on a limited set of high-value browsers.

- On PR: smoke + targeted tests for changed components.

- Pre-release: full regression + integration + sanity on preprod.

17. Metrics & KPIs

- Test pass rate (by suite).

- Defect density and severity distribution.

- Test automation coverage (critical flows automated).

- Mean time to detect and fix breakages.

18. Defect Management

- Use Jira (or configured tool) for reporting defects.

- Required fields: steps, environment, logs, attachments (screenshots, DOM, HAR).

- Defect severity mapping and SLA for triage/resolution.

19. Traceability Matrix (Sample)

Requirement ID	Requirement Description	Test Cases	Status
REQ-01	User Login (multi-tenant)	TC-LOGIN-01, TC-LOGIN-02	Planned

20. Sample Test Cases (High-level)

TC-LOGIN-01: Verify user can log in (Admin role) for Company1 (UI).

TC-LOGIN-02: Verify tenant isolation - Company1 user cannot access Company2 projects (API + UI).

TC-PROJ-01: Create project via API and verify it appears in web UI and mobile (integration).

TC-PERF-01: Verify project listing loads within SLA (performance test, separate plan).

21. Example Automation Snippets and Guidelines

Playwright best practices:

- Use page.locator("[data-test-id=login-email]") selectors.

- Wait for navigation or element visibility after click: page.wait_for_url or locator.wait_for().

- Reuse authenticated sessions via storageState for performance.

API testing tips:

- Use dedicated test tokens and tenant-specific headers (X-Tenant-ID).

- Validate response schema and metadata.

22. Part-specific Responses (from Case Study)

Part 1 — Flaky Login Tests:

- Root causes: missing waits, no navigation or network readiness checks, not using headless vs headed differences, 2FA flakes, timing differences in CI.

- Fixes: use explicit waits (wait_for_url, wait_for_selector), stable selectors, storageState for logged-in sessions, mock or bypass 2FA for automation, add retries, increase timeouts in CI.

Part 2 — Framework Design:

- Included above: modular folder design, config management, fixtures for roles and tenants, BrowserStack integration, reporting (Allure/JUnit).

Part 3 — API + UI Integration Test (sample flow):

- Use an API step to create project with Authorization header and X-Tenant-ID, store returned project id, verify project via UI using Playwright, then verify via BrowserStack mobile run or device farm using a light smoke test that checks for project existence. Ensure cleanup with DELETE API call.

23. Sample Integration Test Pseudocode

```
def test_project_creation_flow(api_client, browser_context, mobile_device):
   # 1. API: create project (POST /api/v1/projects) with X-Tenant-ID header
   # 2. UI: open dashboard, use search/filter to locate project, assert details match
   # 3. Mobile: run minimal check on BrowserStack device to confirm project visibility
   # 4. Security: attempt to view project from another tenant and assert 403 or not found
   # 5. Cleanup: DELETE project via API
```

24. Appendices

A. Known test accounts and tenants (maintained in secure catalog).

B. CI job names and schedules.

C. Contact list: QA, DevOps, Product.

End of Test Plan.