Sample Test Strategy Document

1.Executive Summary

This document presents a sample test strategy for a modern web-based SaaS application with supporting REST APIs, reflecting my practical approach to quality engineering in Agile delivery environments.

The strategy is designed to support frequent, reliable releases by combining strong manual testing practices with scalable test automation using Playwright, integrated into CI/CD pipelines for fast and meaningful feedback.

Key Principles

- Risk-based testing focused on critical user journeys and business logic
- Automation-first mindset where it delivers the highest value
- Shift-left quality, with QA involvement from refinement through release
- Collaboration over handoffs, embedding QA within cross-functional teams

Test Approach

- Manual testing is used for exploratory testing, usability checks, and edge cases
- Automated tests focus on API-level coverage and critical end-to-end user flows
- Regression suites are optimised to remain stable, maintainable, and fast

Tooling & Practices

- Playwright for UI and API automation
- JavaScript/TypeScript-based test framework
- Automated tests executed as part of CI/CD pipelines
- Clear defect lifecycle management and traceability

Measurable Outcomes

- Increased automated test coverage by approximately 25–30%
- Reduced regression execution time by approximately 40%
- Improved early defect detection and release confidence

2. Overview & Objectives

This test strategy outlines a pragmatic approach to quality assurance for a **modern web-based SaaS application with supporting REST APIs**, designed for frequent releases in an Agile environment. The strategy reflects real-world constraints and best practices commonly expected in **UK QA Engineer / SDET roles**.

Objectives:

- Ensure product stability and reliability across web and API layers
- Provide fast, meaningful feedback to the development team
- Reduce defect leakage into production
- Support confident, frequent releases through effective automation

Quality is treated as a shared responsibility across the SDLC, with QA embedded early in planning and refinement activities.

---

3. Application Under Test (AUT) – Assumptions

- Web-based application (React or similar frontend)
- RESTful APIs supporting core business logic
- Multiple user roles and permissions
- Bi-weekly or weekly production releases
- Agile/Scrum delivery model

No organisation-specific or confidential data is referenced in this strategy.

---

4. Test Scope & Test Types

In Scope

- Critical user journeys (end-to-end workflows)
- API business logic, validation, and error handling
- UI behaviour, accessibility basics, and cross-browser compatibility
- Integration points within the system boundary

Out of Scope

- Third-party system reliability
- Performance testing at scale (handled separately)
- Legacy features scheduled for deprecation

Test Levels & Types

- **Smoke Testing:** Core functionality validation post-deployment
- **Regression Testing:** Existing functionality remains stable
- **Exploratory Testing:** Risk-based, session-driven testing
- **API Testing:** Business rules, data integrity, and edge cases
- **Cross-Browser Testing:** Chrome, Firefox, Edge (Safari where required)

Testing is prioritised based on business risk and user impact.

---

5. Automation Strategy

Automation is implemented to maximise confidence while maintaining maintainability and execution speed.

Tools & Frameworks

- **Playwright** for UI and end-to-end automation
- API testing using Playwright request context or equivalent REST clients
- JavaScript / TypeScript-based test framework

Automation Approach

- Follow a **balanced automation pyramid**:
    - API tests form the majority of automated coverage
    - UI automation focuses on critical user flows
- Avoid over-automation of volatile UI components
- Clear separation of test logic, selectors, and test data

What Is Automated

- Smoke and regression test suites

- High-risk and business-critical workflows
- API validation for core services

What Remains Manual

- Exploratory testing
- Usability and edge-case scenarios
- One-off or low-value automation candidates

Test Stability & Maintenance

- Flaky tests investigated and resolved promptly
- Regular refactoring as part of sprint work
- Clear naming and reporting standards

---

6. CI/CD Integration & Test Execution

Automated tests are integrated into the CI/CD pipeline to support fast feedback.

Execution Points

- **Pull Request Checks:** Smoke and key regression tests
- **Scheduled Runs:** Full regression suite (e.g. nightly)

CI/CD Principles

- Fail fast on critical issues
- Non-blocking tests clearly identified
- Test results visible and accessible to the whole team

This approach supports continuous delivery while maintaining quality gates appropriate to release risk.

---

7. Defect Management

Defects are managed throughout their full lifecycle:

- Clear, reproducible defect reports
- Priority and severity agreed collaboratively

- Root cause analysis for recurring issues
- Verification and regression coverage added where appropriate

QA works closely with developers and product owners to prevent defects, not just detect them.

---

8. Test Data Management

- Stable, reusable test data where possible
- Clear ownership of test environments
- Data reset or isolation strategies to ensure repeatable test runs

Test data is designed to support both manual and automated testing reliably.

---

9. Metrics & Continuous Improvement

Quality metrics are used to guide improvement, not as vanity measures.

Example metrics:

- Automated test coverage increased by ~25–30%
- Regression execution time reduced by ~40%
- Reduction in defect leakage to production
- Improved confidence in release readiness

Regular reviews are conducted to refine test coverage, automation value, and overall QA effectiveness.

---

10. Collaboration & Ways of Working

- Active participation in refinement, planning, and retrospectives
- Early risk identification and testability discussions
- Close collaboration with developers and product managers
