Shufti Test Automation Plan (Finalized)

Introduction

Shufti is an AI-based verification platform providing API, Backoffice, and Verify (iFrame) services. This automation plan defines the strategy, execution model, sprinting approach, governance, and decay prevention methods for building and maintaining a scalable and stable automation framework integrated with GitLab CI.

Scope of Automation

- API Services: Biometric, Document, e-IDV, Address, AML, Due Diligence, Consent, MFA, Risk Assessment, Video KYC
- Backoffice: Features, Account Settings, Plug-and-Play Integration, Billing, Merchant Onboarding
- Verify (iFrame): Online (WebSocket), Offline uploads, service-based UI changes
- Exports: CSV, Excel, PDF

Objectives

- End-to-end automation across all sub-platforms
- Reliable CI/CD pipelines with GitLab
- High test coverage without overhead
- Prevent test decay with dedicated sprint allocation

Testing Strategy

- Hybrid Approach:
• BDD (Cucumber + Playwright) for AML, Verify, onboarding flows
• API testing with Playwright API & JSON schema validation
• UI testing for Verify iFrame and Backoffice
• Export validation (CSV/Excel/PDF)
- Test Pyramid:
• 70% API & component tests
• 20% UI/integration tests
• 10% E2E critical BDD flows

Execution Model

- Default: Batch feature files in groups of 5, run in parallel
- Optional: Shard large suites across runners with Playwright’s --shard
- Future: Cloud execution (BrowserStack, Playwright Cloud Grid) for large regression packs or cross-browser/device

CI/CD (GitLab)

- Pipelines:
• Smoke suite on MR
• Regression suite nightly
• Performance suite weekly

- Optimizations:
• Headless Chromium + context reuse
• API/UI pipelines separated
• Retry failed jobs once
• Reporting with Allure + ReportPortal

Test Data & Environment

- Synthetic test data via Faker.js
- Payload builders using JSON/YAML templates
- WireMock for unavailable 3rd-party services
- Stable QA10 environment & dockerized execution

Risks & Mitigations

- Pipeline failures → batching + sharding
- Postman crashes → externalized payloads
- Test decay → sprint maintenance allocation
- Branching chaos → GitLab Flow
- Flaky tests → retry mechanism + Allure tracking

Governance

- ESLint + Prettier enforced
- Modularization + SRP principle
- GitLab Flow branching
- Quarterly automation audits

Sprinting Approach

- 1-week sprints, 40h capacity (2 QAs)
- Allocation:
• 40% New features
• 40% Enhancements/backlog
• 20% CI/CD & maintenance
- Example sprint:
• New features: Consent Verification (16h)
• Enhancements: CSV/Excel export assertions (16h)
• CI/CD: Flaky fixes & runtime improvements (8h)

Current Problems & Fixes

Problem	Solution	Owner	ETA
Pipeline Failures	Batch execution + sharding	DevOps	1 sprint
Postman Crashes	Externalize payloads to JSON/YAML	QA Eng	1 sprint
No Naming Conventions	ESLint + Prettier enforcement	QA Lead	Ongoing
CI/CD Server Choke	Headless Chromium + separate pipelines	DevOps	1 sprint
Partial Coverage	Prioritized backlog automation	QA Lead	2 sprints

Branching Chaos	GitLab Flow adoption	PM + Dev Lead	Ongoing

Test Decay Prevention

- Allocate 10–15% sprint time for test maintenance
- Monitor flakiness via Allure/ReportPortal
- Enforce reviews, linting, modularization
- Stable test data with factories/mocks
- Quarterly audits to prune obsolete tests

Roadmap

- Phase 1 (Stabilization): CI/CD fixes, Postman cleanup, naming standards
- Phase 2 (Coverage Expansion): Decline reasons, billing, exports, Verify refactor
- Phase 3 (Optimization): Allure/ReportPortal, batching, retry logic
- Phase 4 (Future Scale): Cloud parallelization, ML-driven test validation

Conclusion

This finalized plan provides a balanced automation strategy with batch execution, sprint allocations, governance, and test decay prevention. By combining coverage, CI/CD optimization, and risk management, Shufti’s automation framework will remain scalable, compliant, and future-proof.