TEST STRATEGY

DOCUMENT

[Project Name]

[Organization Name]

[Date]

Document ID	[DOC-TS-001]
Version	[1.0]
Author	[QA Lead Name]
Status	[Draft / In Review / Approved]
Classification	[Confidential / Internal]

Document Control

Revision History

Version	Date	Author	Change Description	Approved By
1.0	[Date]	[Name]	Initial creation	[Name]

Reviewers & Approvers

Name	Role	Signature	Date
[Name]	[QA Manager]		[Date]
[Name]	[Project Manager]		[Date]
[Name]	[Dev Lead]		[Date]
[Name]	[Product Owner]		[Date]

Distribution List

Name	Role	Distribution Method
[Name]	[Role]	[Email / Confluence / SharePoint]

Table of Contents

[Table of Contents — Auto-generated from heading styles. Update fields after document is finalized.]

1. Introduction

1.1 Purpose

This Test Strategy document defines the overall testing approach, methodology, and framework for the [Project Name] project. It establishes the guiding principles, standards, and processes that will govern all testing activities throughout the software development lifecycle (SDLC).

[Describe the specific purpose of this test strategy in the context of the project. What does it aim to achieve? Who is the intended audience?]

1.2 Scope

This strategy covers all testing activities for the [Project Name] project including:

- All application modules and components within the project boundary
- Integration points with external systems and third-party services
- All supported platforms, browsers, and device configurations
- Functional, non-functional, and specialized testing types

[Define what is in scope and explicitly state what is out of scope]

1.3 Objectives

- Ensure the application meets all specified business and functional requirements
- Identify and report defects early in the development lifecycle to reduce cost of quality
- Validate system reliability, performance, and security under expected and peak conditions
- Establish a repeatable, efficient, and measurable testing process
- Achieve defined quality gates before each release milestone
- Reduce production defect leakage to below [X]% of total defects found

1.4 References

Doc ID	Document Title	Location
[ID]	Business Requirements Document (BRD)	[Link/Path]
[ID]	System Requirements Specification (SRS)	[Link/Path]
[ID]	Architecture Design Document	[Link/Path]
[ID]	Project Plan	[Link/Path]

2. Project Overview

2.1 Project Description

[Provide a high-level description of the project, its business context, target users, and key capabilities being delivered.]

2.2 Key Stakeholders

Stakeholder	Role	Responsibility	Contact
[Name]	Product Owner	Requirements sign-off	[Email]
[Name]	QA Lead	Test Strategy ownership	[Email]
[Name]	Dev Lead	Technical feasibility	[Email]

2.3 Application Architecture Overview

[Describe or reference the system architecture: frontend, backend, APIs, databases, third-party integrations, microservices, cloud infrastructure, etc. Include an architecture diagram reference if available.]

2.4 Key Business Flows

[List the critical business workflows that must function correctly. Rank them by business impact (Critical / High / Medium / Low).]

3. Test Approach & Methodology

3.1 Overall Testing Philosophy

The testing approach follows a risk-based strategy with shift-left principles, emphasizing early defect detection and continuous quality validation. Testing is integrated into every phase of the SDLC, not treated as a separate phase.

[Describe the overarching testing philosophy: Agile testing quadrants, test pyramid, risk-based testing, shift-left approach, etc.]

3.2 Testing Levels

3.2.1 Unit Testing

Objective	Verify individual code units (functions, methods, classes) work correctly in isolation
Responsibility	Development Team
Tools	[e.g., JUnit, pytest, Jest, NUnit]
Coverage Target	[e.g., Minimum 80% code coverage]
Execution	Automated, runs on every commit via CI/CD pipeline

3.2.2 Integration Testing

Objective	Verify interactions between integrated components, APIs, and services
Responsibility	Development Team / QA Team
Tools	[e.g., Postman, REST Assured, WireMock, Testcontainers]
Approach	[Top-down / Bottom-up / Sandwich / Big Bang]
Focus Areas	API contracts, data flow, error handling between services

3.2.3 System Testing

Objective	Validate the complete system against functional and non-functional requirements
Responsibility	QA Team
Environment	[System Test / Staging environment details]
Approach	End-to-end testing of business scenarios and user journeys

3.2.4 User Acceptance Testing (UAT)

Objective	Validate business requirements are met from the end-user perspective
Responsibility	Business Users / Product Owner

Environment	[UAT environment details]
Entry Criteria	All critical and high-severity defects from System Testing resolved
Sign-off	[Product Owner / Business Sponsor]

3.3 Testing Types

3.3.1 Functional Testing

Validates that each feature and function of the application operates in conformance with the requirements specification.

[Define approach: positive/negative testing, boundary value analysis, equivalence partitioning, decision table testing, state transition testing, use case testing.]

3.3.2 Regression Testing

Ensures existing functionality remains unaffected after code changes, bug fixes, or new feature additions.

Strategy	[Full regression / Risk-based selective regression / Automated regression suite]
Frequency	[Every sprint / Before each release / On-demand]
Automation Target	[e.g., 70% of regression suite automated by Sprint X]

3.3.3 Performance Testing

Load Testing	Validate system under expected concurrent user load
Stress Testing	Determine system breaking point and recovery behavior
Endurance Testing	Validate stability over sustained load periods
Scalability Testing	Validate horizontal/vertical scaling capabilities
Tools	[e.g., JMeter, Gatling, k6, Locust, LoadRunner]
KPIs	[Response time < Xs, Throughput > Y TPS, Error rate < Z%]

3.3.4 Security Testing

SAST	Static Application Security Testing integrated into CI/CD
DAST	Dynamic testing against running application
Penetration Testing	[Internal team / Third-party vendor]
Compliance	[OWASP Top 10, SOC 2, PCI-DSS, HIPAA, GDPR as applicable]
Tools	[e.g., SonarQube, OWASP ZAP, Burp Suite, Snyk, Veracode]

3.3.5 API Testing

[Define approach for API contract testing, schema validation, payload testing, authentication/authorization, rate limiting, error handling, and backward compatibility.]

3.3.6 Accessibility Testing

Standard	[WCAG 2.1 Level AA / Section 508]
Tools	[e.g., axe-core, WAVE, Lighthouse, screen readers]

Scope	[All user-facing pages / Critical workflows]

3.3.7 Compatibility Testing

[Define the browser matrix, OS versions, device types, screen resolutions, and mobile platforms that must be tested.]

3.3.8 Disaster Recovery & Failover Testing

[Define approach for testing backup/restore procedures, failover mechanisms, data integrity after recovery, and RTO/RPO validation.]

4. Test Automation Strategy

4.1 Automation Approach

The automation strategy follows the test pyramid model: a large base of unit tests, a moderate layer of API/integration tests, and a focused set of UI end-to-end tests for critical user journeys.

[Describe the overall automation philosophy, what to automate vs. what to keep manual, and the expected ROI timeline.]

4.2 Automation Framework & Tools

Layer	Tool / Framework	Language
Unit Tests	[JUnit / pytest / Jest]	[Java / Python / JS]
API Tests	[REST Assured / Postman / Karate]	[Java / JS]
UI E2E Tests	[Selenium / Cypress / Playwright]	[Java / JS / Python]
Mobile Tests	[Appium / Detox / XCUITest]	[Java / JS / Swift]
Performance	[JMeter / Gatling / k6]	[Java / Scala / JS]

4.3 Automation Coverage Targets

Test Level	Current Coverage	Target Coverage
Unit Tests	[X]%	[80]%
API / Integration	[X]%	[70]%
UI E2E (Critical Paths)	[X]%	[50]%
Regression Suite	[X]%	[70]%

4.4 CI/CD Integration

[Describe how automated tests are integrated into the CI/CD pipeline: triggers, stages, gates, parallelization, reporting, and failure handling.]

5. Test Environment Strategy

5.1 Environment Topology

Environment	Purpose	Configuration	Data Source
DEV	Development & unit testing	[Config details]	Synthetic data
QA / SIT	System & integration testing	[Config details]	Sanitized production data
STAGING	Pre-production validation	Production-mirror	Anonymized prod data
UAT	Business acceptance	[Config details]	Business-defined data
PERF	Performance testing	Production-equivalent	Volume-scaled data

5.2 Test Data Management

[Define the test data strategy: data creation, masking/anonymization, refresh cadence, synthetic data generation, data-driven testing approach, and GDPR/PII compliance.]

5.3 Environment Management

[Define environment provisioning, refresh schedules, access control, configuration management, and environment parity with production.]

6. Defect Management Strategy

6.1 Defect Lifecycle

All defects follow the standard lifecycle: New → Triaged → Assigned → In Progress → Fixed → Ready for Retest → Verified → Closed (or Reopened).

[Include a defect lifecycle state diagram reference or describe any customizations to the standard flow.]

6.2 Defect Severity & Priority Matrix

Severity	Definition	Example
S1 - Critical	System crash, data loss, complete feature failure, no workaround	Application fails to load, payment processing broken
S2 - Major	Major feature impaired, workaround exists but is difficult	Search returns wrong results, export generates corrupt files
S3 - Moderate	Minor feature impaired, easy workaround available	Sorting not working on one column, tooltip shows wrong text
S4 - Minor	Cosmetic issue, no functional impact	Font inconsistency, minor alignment issue

6.3 Defect SLA & Resolution Targets

Severity	Response Time	Resolution Time	Escalation
S1 - Critical	< 2 hours	< 24 hours	Immediate to PM & Dev Lead
S2 - Major	< 4 hours	< 3 business days	After 24 hours unresolved
S3 - Moderate	< 1 business day	< 5 business days	After 3 days unresolved
S4 - Minor	< 2 business days	Next release	N/A

6.4 Defect Tracking & Reporting

[Define the defect tracking tool (JIRA, Azure DevOps, etc.), mandatory fields, triage process, defect review cadence, and dashboard/reporting requirements.]

7. Risk-Based Testing & Risk Management

7.1 Risk Assessment Framework

Testing effort is prioritized based on a risk matrix that evaluates business impact and likelihood of failure for each feature/module.

Impact ↓ / Likelihood →	Very Low	Low	Medium	High	Very High
Critical	Medium	High	Critical	Critical	Critical
High	Low	Medium	High	Critical	Critical
Medium	Low	Low	Medium	High	Critical
Low	Very Low	Low	Low	Medium	High

7.2 Risk Register

ID	Risk Description	Probability	Impact	Risk Level	Mitigation Strategy
R-01	[Description]	[H/M/L]	[H/M/L]	[Critical/High/Med/Low]	[Mitigation]

7.3 Test Prioritization Based on Risk

[Describe how risk levels translate to testing depth: Critical = exhaustive testing with maximum coverage, High = thorough testing, Medium = standard testing, Low = basic smoke testing.]

8. Entry & Exit Criteria

8.1 Entry Criteria

Test Phase	Entry Criteria	Verified By
Unit Testing	Code complete, code review passed, build successful	Dev Lead
Integration Testing	Unit tests passing >80%, APIs deployed to QA environment	Dev Lead / QA Lead
System Testing	Integration tests passing, test environment stable, test data available, test cases reviewed	QA Lead
UAT	All S1/S2 defects resolved, system testing exit criteria met, UAT environment ready	QA Lead / PM
Performance Testing	Functional testing complete, performance environment provisioned, test scripts ready	QA Lead / DevOps

8.2 Exit Criteria

Test Phase	Exit Criteria	Sign-off By
Unit Testing	>80% code coverage, all critical paths tested, no open S1/S2 defects	Dev Lead
System Testing	>95% test cases executed, >90% pass rate, 0 S1 defects, <3 S2 defects	QA Lead
UAT	All business scenarios validated, business sign-off obtained	Product Owner
Release	All exit criteria met, regression passed, release notes approved, rollback plan documented	Release Manager

9. Test Metrics, KPIs & Reporting

9.1 Key Quality Metrics

Metric	Definition	Target
Test Case Pass Rate	(Passed / Total Executed) × 100	> 95%
Defect Density	Defects per KLOC or per story point	< [X] per KLOC
Defect Leakage Rate	(Production defects / Total defects) × 100	< 5%
Defect Removal Efficiency	(Pre-release defects / Total defects) × 100	> 90%
Automation Coverage	(Automated tests / Total tests) × 100	> 70%
Requirement Coverage	(Requirements with tests / Total requirements) × 100	100%
Mean Time to Detect (MTTD)	Average time from defect introduction to detection	< [X] days
Mean Time to Resolve (MTTR)	Average time from defect report to resolution	< [X] days

9.2 Reporting Cadence

Report	Frequency	Audience
Daily Status	Daily (during active testing)	QA Team, Dev Team
Sprint Test Summary	End of each sprint	Scrum Team, Product Owner
Release Readiness Report	Before each release	All stakeholders
Quality Dashboard	Real-time (automated)	All stakeholders

9.3 Test Management & Reporting Tools

[Define the tools used for test case management (e.g., Zephyr, TestRail, qTest), dashboards (e.g., JIRA Dashboards, Grafana, Power BI), and communication channels for quality reporting.]

10. Roles and Responsibilities

Role	Responsibilities	Assigned To
QA Manager	Test strategy ownership, resource allocation, quality governance, stakeholder reporting	[Name]
QA Lead	Test planning, execution oversight, defect triage, team coordination	[Name]
Test Engineer	Test case design, execution, defect reporting, regression testing	[Names]
Automation Engineer	Framework development, script creation, CI/CD integration, maintenance	[Names]
Performance Engineer	Performance test design, execution, analysis, and optimization recommendations	[Name]
Dev Lead	Unit test ownership, defect resolution, environment support, technical guidance	[Name]
Product Owner	Requirements clarification, UAT coordination, acceptance sign-off	[Name]
DevOps Engineer	CI/CD pipeline, environment provisioning, monitoring, deployment support	[Name]

11. Test Schedule & Milestones

11.1 High-Level Test Schedule

Phase	Start Date	End Date	Duration	Owner
Test Planning	[Date]	[Date]	[X] weeks	QA Lead
Test Design & Preparation	[Date]	[Date]	[X] weeks	QA Team
Unit & Integration Testing	[Date]	[Date]	[X] weeks	Dev Team
System Testing	[Date]	[Date]	[X] weeks	QA Team
Performance Testing	[Date]	[Date]	[X] weeks	Perf Team
Security Testing	[Date]	[Date]	[X] weeks	Security Team
UAT	[Date]	[Date]	[X] weeks	Business
Release Regression	[Date]	[Date]	[X] days	QA Team

11.2 Key Milestones & Quality Gates

[Define the quality gate checkpoints that must be passed before proceeding to the next phase. Include Go/No-Go criteria for each gate.]

12. Communication & Escalation Plan

12.1 Communication Matrix

Communication	Frequency	Medium	Audience	Owner
Daily Standup	Daily	Slack / Teams	QA + Dev	Scrum Master
Defect Triage	3x per week	Video call	QA + Dev leads	QA Lead
Test Status Report	Weekly	Email + Dashboard	All stakeholders	QA Lead
Quality Review	Bi-weekly	Meeting	Leadership	QA Manager

12.2 Escalation Path

[Define the escalation hierarchy: Level 1 (QA Lead) → Level 2 (QA Manager / PM) → Level 3 (Director / VP). Include criteria for escalation and expected response times at each level.]

13. Appendices

13.1 Glossary

Term	Definition
SDLC	Software Development Lifecycle
SIT	System Integration Testing
UAT	User Acceptance Testing
CI/CD	Continuous Integration / Continuous Delivery
SAST/DAST	Static / Dynamic Application Security Testing
KPI	Key Performance Indicator
RTO/RPO	Recovery Time Objective / Recovery Point Objective

13.2 Assumptions

- Requirements are finalized and approved before test design begins
- Test environments will be available as per the agreed schedule
- Development team provides timely defect fixes during test execution cycles
- Third-party integration points have available test/sandbox environments
- Adequate test data is available or can be generated for all test scenarios

[Add project-specific assumptions]

13.3 Constraints

- Testing must be completed within the allocated sprint/release timelines
- Limited access to third-party systems for end-to-end integration testing
- Budget constraints on tool licensing and environment provisioning

[Add project-specific constraints]

13.4 Dependencies

[List all external dependencies: third-party APIs, vendor deliverables, infrastructure provisioning, license availability, etc.]