# Comprehensive Test Plan

## 1. Overview

Tanami Capital is launching a Payment Processing Module designed to handle secure, multi-currency transactions. This module integrates with multiple APIs to process payments, refunds, and currency conversions. The objective of this test plan is to ensure that the module meets functionality, security, and compliance standards through a structured testing approach.

## 2. Testing Strategies

This test plan includes a combination of manual and automated testing to validate the module's reliability across different environments.

### 2.1 Manual Testing Strategy

Manual testing will be employed to validate core functionality, usability, exploratory testing, and edge cases.

- **Functional Testing:** Ensures that key operations like payments, refunds, and currency conversions function correctly.
- **Integration Testing:** Confirms seamless communication between the web platform, mobile apps, and backend APIs.
- **End-to-End Testing:** Validates user journeys from login to payment confirmation.
- **Regression Testing:** Ensures that new changes do not break existing functionality.

### 2.2 Automated Testing Strategy

Automation will be used to improve efficiency and ensure test coverage for frequently executed test cases.

- **Selenium WebDriver:** Automates UI interactions on the web platform.
- **Postman & Newman:** Automates API testing to validate request-response behaviors.
- **CI/CD Integration:** Enables continuous testing in the software delivery pipeline.

## 3. Functional Testing (Positive & Negative Cases)

Functional testing will include both valid (positive) and invalid (negative) scenarios to ensure correct system behavior.

### 3.1 Positive Test Cases

- User can successfully log in and process a transaction.
- Currency conversion calculations are accurate.
- Refund requests are processed correctly.
- Payment confirmation messages are displayed properly.

### 3.2 Negative Test Cases

- Invalid card details should be rejected.
- Unauthorized users should not be able to process payments.
- The system should handle expired session scenarios securely.
- Transactions should fail when the network is lost mid-process.
- Attempting duplicate payments should be prevented.
- Simulating failed currency conversion due to an API failure.

## 4. Security & Compliance Testing

Security and compliance testing ensure that financial data is securely handled and meets regulatory requirements.

### 4.1 KYC & AML Compliance

| Test Case | Scenario | Expected Behavior |
|---|---|---|
| **KYC Verification** | User uploads invalid ID | System rejects the document with an error message |
| **Brute Force Login** | 10 failed login attempts | Account gets locked for 15 minutes |
| **Session Timeout** | User is inactive for 10 minutes | System logs out user automatically |
| **Audit Logging** | User initiates a payment | System logs the event for security reviews |

### 4.2 Data Encryption & Security

- Ensure **sensitive data (e.g., card details, passwords) is encrypted** both in transit and at rest.
- Prevent unauthorized users from **accessing payment APIs**.
- **Token expiration testing** to ensure expired tokens do not allow transactions.

## 5. Integration Testing

Integration testing ensures proper communication between system components.

- **Web Frontend & Backend API:** Verifying transaction processing from the web interface to the backend.
- **Mobile App & API:** Ensuring the mobile version interacts properly with APIs.
- **Third-party API Integrations:** Testing currency exchange rates and external payment gateway processing.

## 6. End-to-End Testing

End-to-End tests validate complete workflows to ensure the system meets user expectations.

### Sample End-to-End Scenarios

- A user logs in, selects a currency, and completes a payment successfully.
- A user attempts a payment, experiences a failure, and retries successfully.
- A refund request is initiated and correctly processed.
- System logs and error handling function as expected.
- Session timeout occurs during an active transaction.

## 7. CI/CD Integration

To ensure continuous quality assurance, the testing strategy integrates into the CI/CD pipeline.

### 7.1 CI/CD Process

1. **GitHub Actions** triggers automated tests on code commits.
2. **Newman executes API tests** before deployment.
3. **Slack alerts** notify the team of failures.
4. **Rollback plan:** If a test fails, deployment is halted, and fixes are applied before retrying.

## 8. Next Steps

- Implement the test cases in Selenium & Postman.
- Develop test scripts for API automation.
- Integrate testing into the CI/CD pipeline for continuous monitoring.
- Perform security testing and evaluate compliance with KYC/AML regulations.