# Test Plan: TB\_TBOA\_PJ UNITY

### 1. Introduction

#### **1.1 Purpose**

This document outlines the comprehensive testing strategy for the **UNITY Project**. The purpose of this test plan is to define the scope, approach, resources, and schedule of all testing activities to validate the new **Payment Uploader (PU)** system. It aims to ensure the developed system meets business requirements, technical specifications, and quality standards.

#### **1.2 Scope**

The scope of this plan covers the testing of all functionalities related to the Payment Module of the PU system. This includes Standing Order (SO) management, payment execution, system integrations (SYB, DWH, OVS-BAS), transaction monitoring, reconciliation, user access control, and notifications.

#### **1.3 Objectives**

* **Validate functional correctness** of all features outlined in the Functional Design Document (FDD).
* **Verify API integrity and performance** for all internal microservices and external integrations.
* **Ensure end-to-end data flow integrity** across all connected systems.
* **Confirm system resilience** by testing error handling, abnormal flows, and timeout scenarios.
* **Guarantee security and access control** by validating user roles (Maker, Approver).
* **Certify non-functional requirements**, including performance and disaster recovery.

### 2. Test Items

The primary test item is the **Payment Uploader (PU) Application, Version 1.0**, which includes:

* PU Web Front-End Application
* PU Backend Microservices (User, Product, Payment, Configuration)
* KONG API Gateway configurations for PU services
* Database schemas and objects related to the Payment Module
* Interfaces to SYB, DWH, OVS-BAS, ENS, and ALPHA systems

### 3. Features to be Tested

* **Standing Order (SO) Management:** Creation, Amendment, and Termination of SOs with full Maker-Approver workflow.
* **Payment Execution Module:** End-to-end payment processing for all specified biller categories (BPJS, PDAM, Multifinance, etc.).
* **Financial Calculations:** Accurate calculation of total debit amounts, including dynamic biller charges from SYB and static MUFG fees.
* **API & System Integration:** All internal and external API endpoints, including SYB Inquiry, Payment, Advice, and Reversal.
* **Transaction Monitoring & Repair:** UI functionality for monitoring and manual repair of failed/suspect transactions.
* **Reconciliation:** Automated processing of the daily .RPT file from SYB and UI display of reconciliation status.
* **User Roles & Access Control:** Verification of distinct permissions for Maker and Approver roles.
* **Email Notifications (ENS):** Triggering and content validation for all payment-related email notifications.

### 4. Features not to be Tested

* Performance and load testing of third-party systems (SYB, OVS-BAS).
* Testing of underlying infrastructure (servers, network) not directly related to the PU application.
* Business process changes in departments not directly impacted by the PU system.
* Functionality related to the "Collection" module.

### 5. Approach (Test Strategy)

A multi-layered testing approach will be adopted, progressing from component-level validation to full system integration testing.

* **Levels of Testing:** Unit Testing, API Testing, System Integration Testing (SIT), and User Acceptance Testing (UAT).
* **Types of Testing:** Functional Testing, Data & Database Testing, Error Handling & Negative Testing, Security Testing, Performance Testing, and Disaster Recovery (DR) Testing.

#### **5.1 Pre-SIT Readiness Approach**

Before commencing formal SIT, a series of readiness checks and tests will be performed to ensure the stability of the environment and its core components. This phase is critical to maximize the efficiency of the subsequent end-to-end testing phases.

* **Flow & Scenarios:**
  1. **Environment Health Check:**
     + **Scenario:** Verify that all integrated systems are online and accessible from the PU application server.
     + **Flow:** QA Team → Ping/Telnet → DWH, SYB, OVS-BAS (JHUB), ENS, RabbitMQ
     + **Verification:** Confirm network connectivity to all required ports for each downstream system. Validate that all PU microservices have been successfully deployed and are running.
  2. **Test Data Verification:**
     + **Scenario:** Confirm that the required test data has been set up correctly in all external systems.
     + **Flow:** QA Team → SQL Query/Manual Check → DWH, OVS-BAS, SYB Sandbox
     + **Verification:** Run queries to confirm that test accounts (CUST-001 to CUST-006) exist in DWH/OVS-BAS with the correct statuses and balances. Manually verify with the SYB technical contact that test biller profiles (e.g., unpaid, paid, invalid) are active in their sandbox.
  3. **Basic API "Smoke" Test:**
     + **Scenario:** Perform a simple, positive-path test for each critical API endpoint to ensure it is responsive.
     + **Flow:** Postman → KONG → PU Services → Downstream Systems
     + **Verification:** Execute a single successful POST /accountnumber call to DWH. Execute a single successful POST /inquiry call to SYB. Execute a basic login call via POST /user/auth/login. This is not exhaustive testing but a confirmation that the interfaces are alive.

#### **5.2 System Integration Test (SIT) Approach**

SIT will be broken down into four distinct but overlapping approaches. The testing will progress from foundational interface validation (ITa) to complex, workflow-driven scenarios (ITb).

1. Interface-Driven Integration Testing (ITa)

This phase serves as the formal execution of Pre-SIT validation, focusing on the technical contracts and interactions between the PU application and its direct dependencies.

* **Flow & Scenarios:**
  + **PU ↔ DWH Interface:**
    - **Scenario:** Validate customer account details.
    - **Flow:** PU Product Service → DWH
    - **Variants:** Test with account numbers that are Active, Dormant, Inactive, and Not Found. Verify that the response payload (accountName, accountStatus, cifNumber) matches the DWH API specification for each case.
  + **PU ↔ SYB Interface:**
    - **Scenario:** Validate a biller's customer ID.
    - **Flow:** PU Payment Service → KONG → SYB
    - **Variants:** Test the /inquiry endpoint for multiple biller types (e.g., BPJS which requires additional\_input, and ACC which does not). Verify the SHA-1 signature is correctly generated and accepted. Test for various business responses like rc: "54" (Bill Paid) and rc: "55" (Bill Not Available).
  + **PU ↔ OVS-BAS Interface:**
    - **Scenario:** Place a hold on customer funds before payment.
    - **Flow:** PU Payment Service → JHUB → OVS-BAS
    - **Variants:**
      * **Success:** Send a **DD050** (Hold Amount) instruction for an account with sufficient funds and verify a success response is received from OVS-BAS.
      * **Failure (Funds):** Send a DD050 for an account with insufficient funds and verify the specific failure response code from OVS-BAS.
      * **Failure (Status):** Send a DD050 for an account that is Dormant or Inactive and verify the rejection from OVS-BAS.

2. Workflow-Based End-to-End Testing (ITb)

The goal of this phase is to validate complete business processes from start to finish, ensuring all integrated systems work in concert to achieve a business outcome.

* **Flow & Scenarios:**
  + **Scenario: Successful Standing Order Payment Lifecycle**
    - **Description:** A standard, automated payment for a previously registered Standing Order.
    - **System Flow:** PU Scheduler → SYB (Inquiry) → PU → OVS-BAS (DD050 Hold) → PU → SYB (Payment) → PU → OVS-BAS (RK170 Booking) → PU → ENS (Notification) → PU → ALPHA (Reporting)
    - **Verification:** Confirm that each step in the chain is completed successfully. A **DD050** transaction must be successfully created in OVS-BAS. After SYB confirms payment, verify the DD050 hold is released and a corresponding **RK170** debit transaction is successfully booked. The final transaction status in PU should be "Paid".
  + **Scenario: Full Failure and Recovery Workflow (Timeout)**
    - **Description:** A payment fails due to a network timeout when communicating with the biller, requiring system recovery.
    - **System Flow:** ... → OVS-BAS (DD050 Hold) → PU → SYB (Payment Timeout) → [Transaction marked "Suspect"] → PU Advice Job (Scheduler) → SYB (Advice) → [Status confirmed] → PU → OVS-BAS (Release DD050 or Book RK170)
    - **Verification:** Ensure the transaction is correctly marked "Suspect" and the **DD050** fund hold remains active in OVS-BAS. Verify the automated Advice Job runs and, based on SYB's final status, either initiates an **RK170** booking or sends a reversal instruction to release the **DD050** hold.
  + **Scenario: Hold Release without Booking (Biller Rejection)**
    - **Description:** A payment is rejected by the biller *after* funds have been successfully held in OVS-BAS.
    - **System Flow:** ... → OVS-BAS (DD050 Hold) → PU → SYB (Payment Fails with RC 54) → PU → OVS-BAS (Release DD050 Hold)
    - **Verification:** Confirm that after SYB rejects the payment (e.g., bill already paid), the PU system sends a specific instruction to OVS-BAS to release the **DD050** hold without proceeding to an **RK170** booking. The transaction status in PU should be updated to "PaidByAnotherMerchant".

3. Data-Driven and Scenario-Based Functional Testing

This approach uses specific data inputs to trigger different logic paths and validate complex business rules within the integrated system.

* **Flow & Scenarios:**
  + **Scenario: Biller-Specific Fee Calculation**
    - **Description:** Test the system's ability to apply different fee structures based on the selected biller.
    - **Data Input:** Create two Standing Orders: one for **Biznet Home** and one for **MPM**.
    - **System Flow:** PU Scheduler → SYB (Inquiry) → PU (Fee Calculation) → OVS-BAS (Hold)
    - **Verification:** When the Biznet Home transaction is processed, verify the total hold amount is (SYB Bill) + 2,270 + 5,000. When the MPM transaction is processed, verify the total hold amount is (SYB Bill) + 2,500.
  + **Scenario: Biller-Specific Data Parsing**
    - **Description:** Test the system's ability to correctly parse complex, multi-part bill amounts from the SYB response.
    - **Data Input:** Use a test customer\_no for **PDAM Surabaya** where the SYB sandbox is configured to return a bill with TAGIHAN: 100000, TUNGGAKAN&DENDA: 15000, and RETRIBUSI: 5000.
    - **System Flow:** PU Scheduler → SYB (Inquiry) → PU (Data Parsing & Calculation)
    - **Verification:** Verify that the "Bill Amount" stored and displayed in the PU system is correctly calculated as **120,000** before any MUFG fees are added.

4. Role-Based Testing

This approach focuses on verifying the Maker-Checker principle and ensuring users can only perform actions permitted by their assigned roles.

* **Flow & Scenarios:**
  + **Scenario: Segregation of Duties - Creation vs. Approval**
    - **Description:** A Maker should not be able to approve their own requests.
    - **System Flow:** Maker Login → PU User Service (Get Permissions) → UI Action → PU Backend (Validate Action)
    - **Verification:**
      * **Variant 1:** A user with only the paymentMaker role creates a new SO. Verify that the "Approve" and "Reject" buttons are not visible or are disabled for them on the SO details screen.
      * **Variant 2:** A user with only the paymentChecker role logs in. Verify that the "Add Standing Order" button is not visible or is disabled.
  + **Scenario: Access Control on Task List**
    - **Description:** A user's task list should only display items relevant to their role.
    - **System Flow:** User Login → PU Payment Service (Query Task List)
    - **Verification:** A paymentMaker logs in and sees an SO that was rejected by an approver (status "Rejected"). A paymentChecker logs in and sees an SO that was submitted by a maker (status "Pending Approval").

5. Asynchronous Workflow & Message Broker Testing

This approach validates the system's use of RabbitMQ for asynchronous task processing, ensuring resilience and scalability as per the Technical Design Document.

* **Flow & Scenarios:**
  + **Scenario: Message Publication and Consumption**
    - **Description:** Verify that a payment task is correctly queued and processed by a worker service.
    - **System Flow:** PU Scheduler (Producer) → RabbitMQ (Queue) → Payment Worker (Consumer) → Downstream Systems
    - **Verification:**
      1. Trigger the payment execution job for a single transaction.
      2. Using RabbitMQ's management console, confirm that one message is published to the payment\_execution\_queue.
      3. Confirm that a Payment Worker service consumes the message and the queue becomes empty.
      4. Verify the transaction is processed end-to-end and its status is updated in the PU database.
  + **Scenario: Resilience - Worker Service Failure**
    - **Description:** Test the system's ability to recover from a worker crash without losing a transaction.
    - **System Flow:** PU Scheduler → RabbitMQ → Payment Worker (CRASH) → RabbitMQ (Re-queue) → New Payment Worker
    - **Verification:**
      1. Publish a payment message to the queue.
      2. Allow a worker to consume the message, but then terminate the worker process *before* it can complete the transaction and send an acknowledgement.
      3. Verify in the RabbitMQ console that the message is not lost and is re-queued.
      4. Confirm that another available worker service eventually picks up and successfully processes the same message.
  + **Scenario: Downstream Task Queuing (Notifications)**
    - **Description:** Verify that a completed payment correctly triggers a downstream asynchronous task, such as an email notification.
    - **System Flow:** Payment Worker → RabbitMQ (email\_queue) → ENS Worker
    - **Verification:** After a Payment Worker successfully processes a transaction, verify that it publishes a new message to the email\_queue. Confirm that an ENS worker consumes this message and the final email is sent.

### 6. Non-Functional Test (NFT) Approach

This section outlines the strategy for verifying the system's non-functional requirements, focusing on its operational readiness, performance, and resilience.

**B1. Infrastructure Verification**

* **Objective:** To confirm that the deployed application and its underlying infrastructure are configured correctly and all communication pathways are open.
* **Flow & Scenarios:**
  + **Scenario: Component & Connectivity Health Check**
    - **Flow:** QA Team/Ops → Server CLI / Monitoring Dashboard → All System Components
    - **Verification:**
      * **Component Status:** Log into the application servers and confirm that all PU microservice processes (User, Product, Payment, Workers) are running. Verify the status of the RabbitMQ and Redis services.
      * **Network Connectivity:** From the PU application server, execute ping and telnet commands to the required hostnames and ports for all external dependencies: DWH database, SYB API endpoint, OVS-BAS (JHUB), and ENS.
      * **Configuration Validation:** Review key application configuration files to ensure database connection strings, API endpoints, and queue names are pointing to the correct SIT environments.

**B2. Capacity and Performance Verification**

* **Objective:** To measure the system's responsiveness, stability, and resource usage under a defined load, ensuring it meets performance benchmarks.
* **Flow & Scenarios:**
  + **Scenario: Load Test (Peak Hour Simulation)**
    - **Flow:** Load Generation Tool (e.g., JMeter) → KONG API Gateway → PU Services → Database/RabbitMQ
    - **Verification:** Simulate 50 concurrent users performing a mix of operations (SO creation, approval, transaction searches) for a sustained period of 1 hour. Monitor and verify that API response times remain below the 2-second threshold and that server CPU and memory usage stay below 75%.
  + **Scenario: Stress Test (Breakpoint Identification)**
    - **Flow:** Load Generation Tool → KONG API Gateway → PU Services
    - **Verification:** Gradually increase the number of virtual users beyond the expected peak load until response times degrade significantly or errors appear. Identify the first component to fail (e.g., database connection pool, a specific microservice CPU) to understand the system's bottleneck.
  + **Scenario: Batch Throughput Test**
    - **Flow:** Test Script → PU Database → PU Scheduler → RabbitMQ → Payment Workers
    - **Verification:** Pre-load the database with 1,000 pending transactions. Trigger the payment execution job and measure the time it takes for all messages to be published to RabbitMQ and subsequently processed by the workers. Calculate the average transactions-per-minute throughput.

**B3. Operations and Capability Verification**

* **Objective:** To ensure the system can be effectively monitored, managed, and supported by the operations team.
* **Flow & Scenarios:**
  + **Scenario: Monitoring and Alerting Verification**
    - **Flow:** Simulate Failure → PU Application → Monitoring System (e.g., Splunk/Dynatrace) → Ops Team
    - **Verification:** Intentionally stop the RabbitMQ service. Verify that a critical alert is automatically triggered in the central monitoring system within 5 minutes, notifying the operations team of the service disruption.
  + **Scenario: Log Validation**
    - **Flow:** User Action (UI) → PU Application → Log Files
    - **Verification:** Perform a single, complete end-to-end payment transaction. Subsequently, inspect the application logs for all relevant microservices. Verify that the entire journey of the transaction is logged with a unique correlation ID, detailing each major step (Inquiry, Hold, Payment, Booking) and the outcome.
  + **Scenario: Backup and Restore Drill**
    - **Flow:** Ops Team → Database Server → Backup Storage → Database Server
    - **Verification:** The operations team will perform a manual backup of the PU database. A test record will then be deleted. The restore procedure will be executed, and the QA team will verify that the deleted record is present again and the application is fully functional.

**B4. High Availability (HA) Verification**

* **Objective:** To validate the system's ability to withstand a single component failure within the production data center without causing a user-facing outage.
* **Flow & Scenarios:**
  + **Scenario: Application Node Failure**
    - **Flow:** Live Traffic → Production Cluster (Node 1 & 2) → [Terminate Node 1 Process] → Load Balancer → Production Cluster (Node 2)
    - **Verification:** While running a low-level load test, terminate one of the active Payment Service instances. Verify that the load balancer correctly detects the failure and redirects all traffic to the remaining healthy node. The load test should show no failed transactions, only a brief increase in latency.
  + **Scenario: Database Failover**
    - **Flow:** PU Application → Primary DB → [Trigger Manual Failover] → Standby DB
    - **Verification:** Trigger a manual failover of the primary database instance to its standby replica within the same data center. Verify that the PU application services automatically detect the change, reconnect to the new primary database, and resume normal operations within the expected recovery time (typically < 1 minute).

**B5. Disaster Recovery Center (DRC) Verification**

* **Objective:** To validate the full disaster recovery plan, ensuring the system can be failed over to the secondary data center (DRC) and brought back online within the defined Recovery Time Objective (RTO).
* **Flow & Scenarios:**
  + **Scenario: Full Failover Drill**
    - **Flow:** Production DC (Active) → [Simulate DC Outage] → DR DC (Activate)
    - **Verification:**
      1. The application is stopped in the production DC.
      2. The replicated database in the DR DC is promoted to become the primary.
      3. All PU application microservices are started in the DR DC.
      4. The DNS/load balancer is switched to point traffic to the DR DC.
      5. A basic smoke test (login, view SO) is performed to confirm the system is operational. The total time taken is measured against the RTO.
  + **Scenario: Failback Drill**
    - **Flow:** DR DC (Active) → [Production DC Restored] → Production DC (Activate)
    - **Verification:** Once the production DC is declared "restored," the failover process is reversed. Database replication from DR back to Production is established and synchronized. The application is then failed back to the primary data center, and its operational status is confirmed.

### 7. Item Pass/Fail Criteria

* **Pass:** A test case is considered "Passed" if the actual result matches the expected result defined in the test case documentation.
* **Fail:** A test case is considered "Failed" if the actual result deviates from the expected result. All failed test cases will be logged as defects with appropriate severity.
* **Entry Criteria:**
  + Successful deployment of the PU application to the SIT environment.
  + All required test environments are available and populated with test data.
  + The development team has completed and signed off on unit testing.
* **Exit Criteria:**
  + 100% of planned test cases have been executed.
  + 95% of test cases have passed.
  + No open "Blocker" or "Critical" severity defects.
  + All UAT test cases have been signed off by business users.

### 8. Suspension Criteria and Resumption Requirements

* **Suspension Criteria:**
  + A "Blocker" defect is found that prevents further testing of a major functional area.
  + The primary test environment (PU SIT) becomes unavailable for more than one business day.
  + A critical external dependency (e.g., SYB Sandbox) is unavailable for more than two business days without a viable mock service.
* **Resumption Requirements:**
  + The blocking defect has been resolved, and a new build has been deployed.
  + The test environment or critical dependency has been restored.
  + A regression test of the affected area is completed successfully.

### 9. Test Deliverables

| **Deliverable** | **Description** | **Owner** |
| --- | --- | --- |
| **Test Plan** | This document. | QA Lead |
| **Test Cases** | Detailed step-by-step instructions for executing tests. | QA Team |
| **Test Execution Log** | Record of all executed test cases and their outcomes (Pass/Fail). | QA Team |
| **Defect Reports** | Detailed reports for each failed test case, logged in the project's defect tracking system. | QA Team |
| **Test Summary Report** | A final report summarizing all testing activities, outcomes, and the overall quality of the system. | QA Lead |

### 10. Testing Tasks (Execution Plan & Key Scenarios)

Testing will be conducted in phases, focusing on mitigating the highest-risk areas first.

#### **Phase 1: API & Core Logic Validation (QA Team)**

* **API-TC-01 (SYB Auth):** Send valid and invalid SHA-1 signatures to all SYB endpoints.
* **API-TC-02 (DWH Account Check):** Call the /accountnumber endpoint with valid, invalid, and dormant accounts.
* **CALC-TC-01 (Complex Fee - PDAM Surabaya):** Verify the final debit amount for a biller with multiple bill components and fees.
* **CALC-TC-02 (Simple Fee - MPM):** Verify the final debit amount for a biller with a simple fee structure.

#### **Phase 2: End-to-End Functional Testing (QA & Business Users)**

* **E2E-TC-01 (Happy Path - Standing Order):** Full lifecycle test: Maker creates SO -> Approver approves -> System executes payment on due date.
* **E2E-TC-02 (Failure - Insufficient Funds):** Execute a payment for a test account with zero balance.
* **E2E-TC-03 (Failure - Biller Timeout):** Simulate a timeout from the SYB /payment API and verify the "Suspect" and "Advice" recovery flow.
* **E2E-TC-04 (UI - SO Amendment):** A Maker amends an existing SO; verify it requires re-approval.

#### **Phase 3: UAT & Non-Functional Testing (Business Users & QA Team)**

* **UAT-TC-01:** Business users execute a predefined set of scripts covering daily operational tasks.
* **PERF-TC-01:** Simulate 50 concurrent users performing tasks while a payment batch is running.
* **DR-TC-01:** Execute the documented disaster recovery failover procedure.

### 11. Environmental Needs

| **System** | **Environment** | **Purpose** |
| --- | --- | --- |
| **Payment Uploader (PU)** | QA/SIT | Primary testing environment for the new application. |
| **SYB Biller Aggregator** | UAT/Sandbox | Endpoint for simulating biller inquiries and payments. |
| **OVS-BAS** | UAT/Test | Core banking simulator for processing fund holds and bookings. |
| **DWH** | Test | Database for testing account number validation. |
| **ENS** | Test | Email service for verifying notification delivery. |

### 12. Test Data Preparation

A dedicated test data set will be created and managed to support all SIT and UAT scenarios. No production data will be used.

#### **12.1 OVS-BAS & DWH Test Data**

A minimum of 10 customer profiles will be created in the OVS-BAS and DWH test environments. These profiles will cover the following variations:

| **Profile ID** | **Account Status** | **Balance** | **Notes** |
| --- | --- | --- | --- |
| CUST-001 | Active | Sufficient | For happy path testing. |
| CUST-002 | Active | Insufficient | For testing "Balance Short" scenarios (e.g., balance < total bill). |
| CUST-003 | Active | Exact Amount | For testing scenarios where balance exactly matches the total bill. |
| CUST-004 | Dormant | Sufficient | For testing validation that prevents transactions from dormant accounts. |
| CUST-005 | Inactive | Sufficient | For testing validation that prevents transactions from inactive/closed accounts. |
| CUST-006 | Active | High | For performance testing scenarios. |

#### **12.2 SYB Biller Aggregator Sandbox Data**

The SYB sandbox environment must be configured with a set of testable customer\_no (Billing IDs) for each in-scope biller product. This data must cover the following states:

| **Biller Product** | **Scenario** | **Required customer\_no** |
| --- | --- | --- |
| **BPJS Kesehatan** | Unpaid Bill | A valid No VA/BPJS with an outstanding bill. |
|  | Already Paid Bill | A valid No VA/BPJS for a bill that has been paid. |
|  | Invalid/Non-existent | A No VA/BPJS that does not exist in SYB's system. |
| **PDAM Surabaya** | Unpaid Bill | A valid No Pelanggan with an outstanding bill. |
|  | Multi-component Bill | A No Pelanggan whose bill includes TAGIHAN, TUNGGAKAN&DENDA, and RETRIBUSI. |
|  | Invalid/Non-existent | An invalid No Pelanggan. |
| **ACC (Multifinance)** | Unpaid Bill | A valid No Kredit with an outstanding bill. |
|  | Already Paid Bill | A valid No Kredit for a paid bill. |
| **All Other Billers** | (Repeat as above) | At least one valid (unpaid), one invalid, and one already-paid customer\_no for each product. |

#### **12.3 Data Management Responsibilities**

* **QA Team:** Responsible for defining the test data requirements and providing the list of required profiles and scenarios to the respective teams.
* **OVS-BAS/DWH Admin Team:** Responsible for creating and maintaining the customer account profiles in the test environments.
* **SYB Technical Contact:** Responsible for configuring the biller sandbox with the required test customer numbers and their corresponding bill statuses.

### 13. Responsibilities

| **Role** | **Responsibilities** |
| --- | --- |
| **Project Manager** | Overall project oversight, resource allocation, and issue resolution. |
| **Development Team** | Unit testing, defect fixing, and application deployment to the test environment. |
| **QA Team** | Test planning, test case creation, test execution (API, SIT, NFR), defect logging, and reporting. |
| **Business Users** | UAT execution, validation of business requirements, and final sign-off. |

### 14. Staffing and Training Needs

* The existing QA team is staffed to perform the required testing.
* Business users participating in UAT will require a 2-hour training session on the PU application's features and the UAT process.

### 15. Schedule

*A detailed schedule will be maintained in the master Project Plan document. High-level milestones are as follows:*

* **SIT Phase 1 (API Testing):** [Start Date] - [End Date]
* **SIT Phase 2 (E2E Testing):** [Start Date] - [End Date]
* **UAT & NFR Testing:** [Start Date] - [End Date]

### 16. Risks and Contingencies

| **Risk** | **Likelihood** | **Impact** | **Mitigation Strategy** |
| --- | --- | --- | --- |
| **SYB Sandbox Unavailability** | Medium | High | Establish a clear support contact with SYB. Develop a set of mock API responses (stubs) for use if the live sandbox is down. |
| **Inaccurate Test Data** | High | High | Dedicate a resource to manage test data creation and maintenance across all systems (SYB, OVS-BAS, DWH) before SIT begins. |
| **Complex Biller Logic Defects** | High | Medium | Prioritize testing of billers with complex calculation logic. Create specific, data-driven test cases for each component. |
| **Timeout/Error Handling Failure** | Medium | High | Conduct rigorous negative testing by simulating API timeouts and error codes from SYB to ensure recovery flows are robust. |

### 17. Automation Test Strategy

This section outlines the approach for automating test cases to improve efficiency, increase coverage, and enable a robust regression testing practice.

#### **17.1 Objectives of Automation**

* **Regression Suite:** To create a stable suite of automated tests that can be run against every new build to ensure existing functionality is not broken.
* **Efficiency:** To automate repetitive and time-consuming test cases, such as the creation of Standing Orders for multiple biller types, freeing up manual testers to focus on exploratory and usability testing.
* **Continuous Feedback:** To integrate automated tests into a CI/CD pipeline to provide rapid feedback to the development team on the health of the application.

#### **17.2 Scope of Automation**

* **In-Scope for Automation:**
  + **API Layer (High Priority):** All critical API endpoints defined in the PU.postman\_collection.json. This includes all interactions with SYB, DWH, and internal PU services. This is the most stable and valuable layer to automate.
  + **Core UI Workflows (Medium Priority):** The "happy path" user journeys that are critical to the business, including:
    - Creating a new Standing Order.
    - Approving a Standing Order.
    - Searching and viewing a transaction in the monitoring screen.
  + **Critical End-to-End Scenarios:** A limited set of high-value end-to-end tests that validate the full payment flow, focusing on the happy path and the insufficient funds scenario.
* **Out-of-Scope for Automation:**
  + Complex UI interactions that are brittle and have a low return on investment (e.g., dynamic table sorting, complex filter combinations).
  + One-time setup or configuration tasks.
  + Features with a highly volatile UI that is expected to change frequently.

#### **17.3 Tools and Framework**

* **API Automation:** **Postman** with **Newman** will be used to execute the API test collections from the command line, allowing for integration into CI/CD pipelines.
* **UI Automation:** **Cypress** is recommended for its modern architecture and ease of use in testing web applications. The framework will be built using the **Page Object Model (POM)** design pattern to ensure maintainability and reusability of code.
* **CI/CD Integration:** **Jenkins** will be used to schedule and trigger the execution of the automated test suites.
* **Reporting:** The **Allure Report** framework will be integrated to generate detailed, readable, and shareable test execution reports.

#### **17.4 Automation Execution Plan**

* **Smoke Suite:** A small subset of critical API and UI tests (<< 5 minutes execution time) that will be run automatically on every new deployment to the QA/SIT environment. A failure in the smoke suite will reject the build and prevent further testing.
* **Regression Suite:** The full suite of automated API and UI tests will be scheduled to run nightly against the SIT environment. The results will be automatically published and emailed to the project team each morning.
* **On-Demand Execution:** QA team members will have the ability to run specific automation suites manually against their test environments as needed.

### 18. Approvals

*Sign-off on this document indicates approval of the testing strategy and plan for the UNITY Project Payment Module.*

**QA Lead:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Project Manager:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Business User Lead:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_