# Test Plan: Project UNITY - Payment Module (SIT & UAT)

**Version:** 1.4

**Publish Date:** 19-Aug-2025

## 1. Introduction

This document provides a detailed testing strategy for the **Payment Module** of Project UNITY. It is an enhancement of the original test plan, incorporating deep insights from all available project documentation, including the Functional Design Document (FDD), Technical Design Documents (TDDs), and detailed interface specifications.

The primary goal is to ensure that all payment-related business requirements are met, that the application is functionally correct and robust, and that it integrates seamlessly with all internal and external systems before production deployment.

## 2. Scope

### 2.1 In-Scope Functionality

This test plan covers all functionalities of the Payment Module accessible to the Jakarta Deposit and Operation teams.

* **Standing Order (SO) Management:**
  + Creation of new SOs (Maker role).
  + Approval and Rejection of new SOs (Approver role).
  + Amendment of existing SOs (Maker/Approver flow).
  + Termination of active SOs (Maker/Approver flow).
* **Automated Payment Execution:**
  + Scheduled job for triggering payments based on due dates.
  + Real-time bill inquiry to the SYB Biller Aggregator.
  + End-to-end payment processing, including financial settlement in OVS.
* **Transaction Monitoring & Repair:**
  + Viewing the status of all payment transactions (e.g., Outstanding, Paid, Failed, Balance Short).
  + Manual intervention (Repair) for failed transactions by authorized users.
* **Reconciliation:**
  + File upload and processing for transaction reconciliation against biller reports.
* **User Roles & Permissions:**
  + Validation of distinct workflows and access controls for "Maker" and "Approver" roles as they relate to payments.
* **Notifications:**
  + Verification that all relevant payment events trigger the correct email notifications via the Email Notification Service (ENS).

### 2.2 Out-of-Scope Functionality

* All functionalities intended for users outside of Jakarta.
* The "Standing Order Collection" feature.
* Any changes or impacts on upstream systems not directly related to the PU application.

## 3. System Integration Landscape

The Payment Module is a highly integrated component. SIT will focus on validating the data flow and transactional integrity across the following key integration points:

* **Payment Uploader (PU) ↔ Data Warehouse (DWH):** For customer data enrichment and validation.
* **Payment Uploader (PU) ↔ SYB Biller Aggregator:** For real-time bill inquiry and payment execution.
* **Payment Uploader (PU) ↔ JHUB ↔ OVS:** For all financial transactions, including fee booking, amount holds, and final settlement.
* **Payment Uploader (PU) ↔ Email Notification Service (ENS):** For sending automated email notifications to customers.

## 4. Test Approach for System Integration Testing (SIT)

The SIT will be executed in three comprehensive phases to ensure a methodical and thorough validation of the Payment Module, covering all aspects of its functionality and integration.

### Phase 1: Interface and API Validation

This phase focuses on testing the direct request/response contracts between the PU and its dependencies to ensure they can communicate correctly before testing complex business flows.

#### **Flow 1: PU ↔ DWH Integration (Data Enrichment & Validation)**

* **Objective:** To verify that the PU can correctly fetch, validate, and utilize customer account data from the DWH.
* **Reference:** DWH API SPEC PAYMENT.xlsx, PU.postman\_collection (1).json

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_DWH\_1.1** | **Happy Path - Active Account:** A user enters a valid and "Active" accountNumber in the PU. | The PU successfully calls the DWH /accountnumber endpoint. The correct accountName and cifNumber are automatically populated and displayed in the PU's read-only fields. |
| **SIT\_DWH\_1.2** | **Negative Path - Invalid Account:** A user enters an accountNumber that does not exist in the DWH. | The DWH API returns an error. The PU displays a user-friendly message (e.g., "Account Number not found") and prevents the user from proceeding. |
| **SIT\_DWH\_1.3** | **Edge Case - Dormant/Inactive Account:** A user enters a valid accountNumber with a status of "Dormant" or "Inactive". | The DWH returns the account details with the corresponding status. The PU displays a clear warning message (e.g., "Account is dormant and cannot be used for transactions") and blocks the transaction. |
| **SIT\_DWH\_1.4** | **API Security - Missing Headers:** An API call is made to the DWH endpoint from the PU without the required user-token or ApiKey. | The DWH gateway should reject the request with a 401 Unauthorized or 403 Forbidden status code. |

#### **Flow 2: PU ↔ SYB Biller Aggregator Integration (External Communication)**

* **Objective:** To validate that the PU can correctly handle inquiries and payments with the external SYB API for various biller types and properly interpret the responses.
* **Reference:** SYB API SPEC PAYMENT.xlsx, Tech Spec Payment Web Service (JSON) SYB v1.11.pdf

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_SYB\_2.1** | **Happy Path - Multi-Biller Inquiry:** For each major biller category (Multifinance, PDAM, BPJS), perform a bill inquiry using a valid customer number. | The PU correctly parses the different data-n fields from the SYB JSON response to display the accurate Customer Name, Bill Amount, and Admin Biller fee. The "Total Amount" displayed must correctly sum these values plus the Admin MUFG fee. |
| **SIT\_SYB\_2.2** | **Negative Path - Biller Error Response:** Using a mock of the SYB service, simulate various failure response codes (rc) such as "Invalid Customer Number" or "Bill Already Paid." | The PU prevents the user from proceeding with payment and displays a clear, user-friendly error message that corresponds to the specific failure. |
| **SIT\_SYB\_2.3** | **Data Integrity - String Parsing Validation:** For a biller that relies on string parsing (e.g., Multifinance data4), test with mock responses where the spacing or labels are slightly altered. | The PU's parsing logic should be resilient and still extract the correct customer name. If it cannot, it should fail gracefully with a "Could not retrieve customer name" message rather than crashing or displaying partial data. |
| **SIT\_SYB\_2.4** | **Specific Input - BPJS Inquiry:** Perform an inquiry for a BPJS Kesehatan biller, ensuring the additional\_input field for "Jumlah Bulan" (Number of Months) is correctly sent in the request to SYB. | The request to SYB contains the additional\_input parameter with the correct value. The response from SYB is parsed correctly. |

### Phase 2: End-to-End Business Flow Validation

This phase tests the complete, orchestrated business processes from user action to financial settlement.

#### **Flow 3: Standing Order (SO) Lifecycle Management**

* **Objective:** To validate the full lifecycle of creating, approving, amending, and terminating a Standing Order.
* **Reference:** High Level Payment Flow.xlsx, JHUB Mapping files

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_SO\_REG\_3.1** | **Happy Path - Maker/Approver Flow:** 1. A **Maker** creates a new SO. 2. An **Approver** approves it. | 1. A RK170 transaction for the creation\_charge is posted in OVS. 2. The SO status becomes "Active". 3. A "Registration Successful" email is sent via ENS. |
| **SIT\_SO\_REG\_3.2** | **CRITICAL - Data Truncation Test:** 1. A Maker creates an SO using a valid account\_number\_charge that is **25 characters** long. 2. An Approver approves. | The system must reject the transaction because the source data (25 chars) exceeds the JHUB target field length (JHGDACN - 15 chars). **Failure Condition:** The system truncates the account number. This is a **Blocker** defect. |
| **SIT\_SO\_REG\_3.3** | **Negative Path - OVS Rejection (Insufficient Funds):** 1. A Maker creates an SO using an accountNumberCharge with insufficient funds. 2. An Approver approves. | 1. OVS rejects the RK170 transaction. 2. The SO status is updated to "Registration Failed," and the task remains in the Approver's queue. |
| **SIT\_SO\_AMEND\_3.4** | **Happy Path - SO Amendment:** 1. A Maker amends an active SO (e.g., changes the accountNumberCharge). 2. An Approver approves the amendment. | 1. An amendment fee (if applicable) is charged via RK170. 2. The SO details are updated in the PU database. 3. The *next* payment cycle uses the new, amended details. 4. An "Amendment Successful" email is sent. |
| **SIT\_SO\_TERM\_3.5** | **Happy Path - SO Termination:** 1. A Maker terminates an active SO. 2. An Approver approves the termination. | 1. A termination fee (if applicable) is charged. 2. The SO status becomes "Terminated". 3. The SO does **not** generate a payment transaction on its next scheduled due date. |

#### **Flow 4: Automated End-to-End Payment Execution**

* **Objective:** To validate the entire automated payment process, from due date check to final settlement and notification, covering various success and failure paths.
* **Reference:** OVS payment flow.jpg, JHUB Mapping files for DD050, DD100, and RK170.

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_PAY\_EXEC\_4.1** | **Happy Path - Full Successful Cycle:** 1. Set up an active SO with a due date for today. 2. Trigger the scheduled job. | **A.** Successful inquiry to SYB. **B.** **DD050** hold transaction posted in OVS. **C.** Successful payment request to SYB. **D.** DD050 hold released and final **DD100** or **RK170** booking posted in OVS. **E.** "Payment Successful" email sent via ENS. **F.** Transaction status is "Paid" in the PU UI. |
| **SIT\_PAY\_EXEC\_4.2** | **Negative Path - Insufficient Balance at Hold:** 1. Set up an active SO where the accountNumberDebit has insufficient funds. 2. Trigger the job. | 1. The SYB inquiry is successful. 2. JHUB/OVS rejects the **DD050** hold request. 3. The transaction status in the PU is updated to "Balance Short." 4. An "Insufficient Balance" email is sent. |
| **SIT\_PAY\_EXEC\_4.3** | **Negative Path - Biller Rejects Payment After Hold:** 1. Set up an active SO with sufficient funds. 2. Trigger the job and confirm the DD050 hold succeeds. 3. Using a mock, configure SYB to return a "Payment Failed" response *after* the hold is placed. | 1. The transaction status in the PU is updated to "Failed." 2. **Crucially**, the system must automatically trigger a reversal to release the **DD050** hold in OVS, ensuring the customer's funds are not locked. 3. A "Payment Failed" notification is sent via ENS. |
| **SIT\_PAY\_EXEC\_4.4** | **Edge Case - Paid by Another Merchant:** 1. Set up an active SO. 2. Using a mock, configure SYB to return a response indicating the bill was already paid. | 1. The transaction status in the PU is updated to "PaidByAnotherMerchant". 2. No financial hold (DD050) or debit (DD100) is attempted against the customer's account. 3. An appropriate notification is sent. |

### Phase 3: Special Cases and Non-Functional Validation

This final phase covers other critical aspects of the application's quality.

#### **Flow 5: Transaction Repair and Reconciliation**

* **Objective:** To validate the manual intervention processes for failed transactions and reconciliation.
* **Reference:** Process and People.xlsx - Impact Analysis.csv, Payment mockup design.docx

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_REPAIR\_5.1** | **Happy Path - Transaction Repair:** 1. A transaction is in a "Balance Short" status. 2. A Maker initiates a "Repair" action. 3. An Approver approves the repair. | 1. The transaction is re-submitted into the payment execution flow. 2. If funds are now sufficient, the payment proceeds successfully (as per SIT\_PAY\_EXEC\_4.1). 3. A "Repair Successful" email is sent. |
| **SIT\_RECON\_5.2** | **Happy Path - Reconciliation File Upload:** 1. The Operations team uploads a valid reconciliation report from a biller. | 1. The PU successfully parses the file. 2. The status of the corresponding transactions in the PU is updated to "Matched" or "Reconciled". 3. A summary report of the upload (e.g., X records matched, Y mismatched) is displayed. |

#### **Flow 6: Role-Based Access Control (RBAC) and Security**

* **Objective:** To ensure the distinct permissions and workflows for "Maker" and "Approver" roles are strictly enforced at both the UI and API levels.
* **Reference:** Payment mockup design.docx, PU.postman\_collection (1).json

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **SIT\_RBAC\_6.1** | **Maker UI Validation:** Log in as a "Maker". Navigate to the Standing Order and Transaction task lists. | The user can see buttons for "Add," "Amend," and "Terminate." The "Approve" and "Reject" buttons must **not** be visible. |
| **SIT\_RBAC\_6.2** | **Approver UI Validation:** Log in as an "Approver". Navigate to the task lists. | The user can see "Approve" and "Reject" buttons for pending tasks. The "Add Standing Order" button should not be visible (confirming segregation of duties). |
| **SIT\_RBAC\_6.3** | **API Security - Maker Cannot Approve:** 1. Log in as a "Maker" and obtain a user token. 2. Using an API tool (e.g., Postman), attempt to call the /approve endpoint for a pending SO, using the Maker's token. | The API call must fail with a 403 Forbidden status code. The SO's status must remain "Pending Approval". |
| **SIT\_RBAC\_6.4** | **API Security - Unauthorized Access:** Attempt to access any PU page or API endpoint *without* a valid login/token. | The system should redirect to the login page (for UI) or return a 401 Unauthorized error (for API). |
| **SIT\_RBAC\_6.5** | **Workflow - Self-Approval Prevention:** 1. A user who has both "Maker" and "Approver" rights (if such a role exists) creates a new SO. 2. The same user attempts to approve their own submission. | The system should prevent the action and display an error message, "You cannot approve your own request," enforcing the four-eyes principle. |

## 5. Non-Functional Testing (NFT) Approach

NFT will be conducted in parallel with SIT to ensure the application meets requirements for performance, reliability, and operational readiness.

### B1. Infrastructure Verification

* **Objective:** To verify that all application components are deployed correctly and that connectivity between them is established as per the architecture diagrams.
* **Flow:** A series of health checks and connectivity tests performed by the QA and infrastructure teams prior to the start of SIT execution.

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **NFT\_INFRA\_1.1** | **Component Connectivity:** From the PU application server, test network connectivity (e.g., using telnet or curl) to the required endpoints for JHUB (via KONG), DWH, RabbitMQ, and Redis. | All connections are established successfully on the designated ports without timeouts or refusal errors. |
| **NFT\_INFRA\_1.2** | **Firewall Verification:** Confirm with the network team that firewall rules are in place to allow traffic between all required system components as defined in the TDD. | A network trace or firewall rule review confirms that traffic is not being blocked. |
| **NFT\_INFRA\_1.3** | **Service Health Checks:** Access the health check endpoint (e.g., /health) for each microservice (Payment, Product, etc.). | Each service returns a 200 OK status, indicating it is running and connected to its database. |

### B2. Capacity and Performance Verification

* **Objective:** To measure the system's responsiveness, stability, and resource usage under various load conditions and ensure it meets defined performance benchmarks.
* **Flow:** Use a load testing tool (e.g., JMeter) to simulate concurrent users and batch processes executing key business flows.
* **Prerequisites:**
  + The test database must be pre-loaded with a realistic volume of data: at least **20,000 active Standing Order records** across various billers.
  + Non-functional requirements must be defined (e.g., "Average API response time under 2 seconds").
* **Monitoring:** During all tests, the following will be monitored:
  + Server CPU and Memory Utilization (PU, JHUB, Database).
  + API Response Times (average and 95th percentile).
  + API Error Rates (%).
  + Database connection pool usage and query performance.
  + RabbitMQ message queue depth and consumer processing rates.

#### **Flow 1: Online Transaction Performance (User Activity Simulation)**

| **Scenario ID** | **Scenario Description** | **Key Performance Indicators (KPIs)** |
| --- | --- | --- |
| **NFT\_PERF\_2.1** | **Load Test:** Simulate a gradual ramp-up to **100 concurrent users** over 15 minutes. The user mix will be: 60% viewing transaction lists, 20% creating new SOs, 15% performing bill inquiries, and 5% approving SOs. | - Average API response time < 2 seconds. <br> - Error rate < 0.1%. <br> - Server CPU utilization < 75%. |
| **NFT\_PERF\_2.2** | **Endurance (Soak) Test:** Run a sustained, moderate load of **50 concurrent users** for **4 hours**, executing the same transaction mix as the Load Test. | - No memory leaks (stable memory usage over time). <br> - No degradation in response time over the 4-hour period. |
| **NFT\_PERF\_2.3** | **Spike Test:** While running a baseline load of 25 users, suddenly inject an additional **150 users** for 5 minutes to simulate a peak business event (e.g., start of the month). | - The system must not crash or become unresponsive. <br> - Response times should return to baseline levels within 10 minutes after the spike ends. |

#### **Flow 2: Batch Transaction Performance (Automated Payment Job Simulation)**

| **Scenario ID** | **Scenario Description** | **Key Performance Indicators (KPIs)** |
| --- | --- | --- |
| **NFT\_BATCH\_2.4** | **Volume Test:** Trigger the automated payment job to process **10,000 due transactions** at once. No other user activity is simulated. | - Total job completion time. <br> - Transactions processed per minute. <br> - RabbitMQ queue for payment processing should not grow uncontrollably and should return to zero after the job completes. |
| **NFT\_BATCH\_2.5** | **Concurrency Test (Stress Test):** While the payment job is processing **5,000 transactions**, simulate **50 concurrent users** performing online activities (as per NFT\_PERF\_2.1). | - The batch job's completion time should not increase by more than 20% compared to the Volume Test baseline. <br> - Online user response times should not degrade by more than 30% from their baseline. <br> - No database deadlocks should occur. |

### B3. Operation / Compatibility Verification

* **Objective:** To ensure the application can be operated, monitored, and supported effectively by the Operations team and is compatible with required client software.
* **Flow:** Testers will execute scenarios based on documented operational procedures.

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **NFT\_OPER\_3.1** | **Log Verification:** An operator performs a test transaction and then checks the application logs on the server. | The transaction flow is clearly logged with appropriate INFO/ERROR levels. The log entries contain a unique transaction ID for easy tracing. |
| **NFT\_OPER\_3.2** | **Audit Trail Verification:** An Approver rejects a Standing Order. | A record is created in the audit database table with the Approver's username, the action taken ("Reject"), a timestamp, and the ID of the SO. |
| **NFT\_COMP\_3.3** | **Browser Compatibility:** Execute a smoke test of the "Add Standing Order" and "Approve Standing Order" user journeys. | The application is fully functional and renders correctly on the latest versions of Google Chrome, Microsoft Edge, and Mozilla Firefox. |

### B4. High Availability (HA) Verification

* **Objective:** To verify the system's resilience and automatic failover capabilities within a single data center.
* **Flow:** While a low-level performance test is running, the infrastructure team will simulate the failure of a single component in the cluster.

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **NFT\_HA\_4.1** | **Application Server Failover:** While a load of 20 concurrent users is running, one of the PU application server nodes is shut down. | The load balancer correctly redirects all traffic to the remaining active nodes. Users may experience a single failed request, but subsequent requests should succeed. The overall service remains available. |
| **NFT\_HA\_4.2** | **Database Failover:** The primary database instance is manually failed over to the standby instance. | The application may become briefly unresponsive but should automatically reconnect to the new primary database within 1-2 minutes. No data corruption occurs. |

### B5. Disaster Recovery (DRC) Verification

* **Objective:** To validate the documented disaster recovery plan by simulating a full data center failure and failback.
* **Flow:** A planned, coordinated event involving all relevant teams to execute the failover and failback procedures as outlined in the Functional Design Document.docx (Section 14).

| **Scenario ID** | **Scenario Description** | **Expected Result** |
| --- | --- | --- |
| **NFT\_DRC\_5.1** | **DR Failover:** 1. The Production (PROD) data center is declared unavailable. 2. The DR failover procedure is initiated. | 1. The application becomes available on the DR site within the defined Recovery Time Objective (RTO). 2. Data loss is within the defined Recovery Point Objective (RPO). 3. The system is fully functional for all key business flows on the DR site. |
| **NFT\_DRC\_5.2** | **DR Failback:** 1. The PROD data center is restored. 2. The failback procedure is initiated to return services to the PROD site. | 1. Traffic is successfully redirected back to the PROD site. 2. All data that was processed on the DR site is successfully replicated back to the PROD database before the switchover. 3. The DR site is successfully returned to a standby state. |

## 6. Test Data Requirements

To effectively execute the scenarios above, the following test data must be prepared in the SIT environment prior to the start of testing.

* **Customer Accounts (OVS):**
  + At least 5 active CUA accounts with sufficient funds.
  + At least 2 active CUA accounts with zero or insufficient funds.
  + At least 1 "Dormant" account.
  + At least 1 "Inactive" account.
* **Biller Data (SYB):**
  + A list of valid, reusable test customer numbers for each in-scope biller category (Multifinance, PDAM, BPJS, etc.).
* **User Accounts (PU):**
  + At least two user accounts with only the "Maker" role.
  + At least two user accounts with only the "Approver" role.
  + (If applicable) One user account with both "Maker" and "Approver" roles.

## 7. Risks & Mitigation (Payment Module Specific)

This section supplements the general risks outlined in the main Test Plan.

| **Risk ID** | **Risk Description** | **Likelihood** | **Impact** | **Mitigation Strategy** |
| --- | --- | --- | --- | --- |
| **RISK-PAY-01** | **Data Truncation:** Mismatched field lengths between PU and JHUB cause critical data (e.g., account numbers) to be silently corrupted. | High | Critical | Execute targeted SIT scenarios (e.g., **SIT\_SO\_REG\_3.2**) with boundary value analysis. All data truncation issues will be raised as **Blocker** defects. |
| **RISK-PAY-02** | **Incorrect Fee Calculation:** Ambiguous logic for calculating the Total Amount across different billers leads to incorrect customer debits. | Medium | Critical | Execute test scenarios for every biller, validating the Total Amount against the formula: Bill Amount + Admin Biller + Admin MUFG. |
| **RISK-PAY-03** | **Fragile SYB Integration:** Changes in the SYB API's free-form data-n fields break the PU's parsing logic, causing payment inquiries to fail. | Medium | High | Execute scenario **SIT\_SYB\_2.3** to test the resilience of the parsing logic. Propose a long-term enhancement to the SYB API for structured data. |
| **RISK-PAY-04** | **Improper Access Control:** A user is able to perform actions outside of their designated role (e.g., a Maker approving a payment). | Medium | High | Execute the dedicated RBAC test scenarios in **Flow 6** to validate both UI and API-level security controls. |

## **8. Automation Test Approach**

### **8.1 Strategy and Scope**

The automation strategy for Project UNITY focuses on creating a robust regression suite that provides rapid feedback on the health of the application's core functionalities. The primary goals are to:

* Accelerate regression testing cycles.
* Ensure the integrity of critical API contracts between microservices.
* Enable continuous testing by integrating the automated suite into the CI/CD pipeline.

#### **Scope of Automation**

* **In-Scope for Automation:**
  + **API-Level Testing:** All major API endpoints exposed by the PU microservices will be automated. This is the highest priority, as it provides the most stable and efficient way to test business logic and integration points.
  + **Core End-to-End (E2E) Scenarios:** A select number of high-value, "happy path" business flows will be automated at the UI level to serve as a smoke test.
  + **Regression Suite:** All automated tests will be consolidated into a regression suite that is run frequently.
* **Out-of-Scope for Automation (Manual Testing):**
  + **UI/UX and Visual Testing:** Validating layouts, colors, and overall user experience is best suited for manual testing.
  + **Exploratory Testing:** Creative, unscripted testing to find edge-case defects will be performed manually.
  + **Complex Negative Scenarios:** One-off, intricate negative test cases that require significant setup will be tested manually.

### **8.2 Tools and Framework**

* **API Testing:** **Postman** (for test case creation) and **Newman** (for command-line execution). The existing PU.postman\_collection (1).json will be used as the foundational asset.
* **UI Testing:** **Selenium** with a **Java/TestNG** framework. This provides a robust and widely-supported platform for browser automation.
* **CI/CD Integration:** **Jenkins**. The automated test suites (both API and UI) will be configured as a Jenkins job that can be triggered automatically on a nightly schedule or after a new build is deployed to the SIT environment.

### **8.3 Automation Flows and Scenarios**

#### **Flow 1: API Regression Suite (Postman/Newman)**

* **Objective:** To build a comprehensive API test suite that validates the contracts, logic, and integration of all PU microservices.

| **Scenario ID** | **Scenario Description** | **Automation Steps** |
| --- | --- | --- |
| **AUTO\_API\_1.1** | **Authentication and Authorization:** Verify the login process and token-based security for all protected endpoints. | 1. Automate the /user/auth/login request. 2. Extract the userToken from the response and store it as a collection variable. 3. Ensure all subsequent requests include this token in the Authorization header. 4. Include a test that attempts to access a protected endpoint with an invalid token and asserts a 401 Unauthorized response. |
| **AUTO\_API\_1.2** | **DWH Data Integrity:** Automate the validation of the DWH integration for account enrichment. | 1. Create a data-driven test using a CSV file with various account numbers (Active, Dormant, Invalid). 2. Call the /product/check-account-number/{accountNumber} endpoint. 3. Assert that the response body contains the correct accountName, cifNumber, and accountStatus for each valid account, and a proper error for invalid ones. |
| **AUTO\_API\_1.3** | **SYB Biller Inquiry Validation:** Automate the bill inquiry flow for all major biller types. | 1. Using a mock of the SYB service, create distinct mock responses for each biller (ACC, PDAM, BPJS, etc.). 2. Automate a POST request to the PU's /externals/check-billing endpoint for each biller type. 3. Assert that the PU's response correctly calculates the totalBill and extracts the customerName based on the specific parsing logic for that biller. |
| **AUTO\_API\_1.4** | **Data Truncation Negative Test:** Create a specific API test to validate the system's defense against data truncation. | 1. Send a POST request to the /StandingOrder endpoint to create a new SO. 2. In the request body, use a 25-character string for the accountNumberCharge. 3. **Assert** that the API response is a 400 Bad Request or similar error, explicitly stating that the field length is invalid. This directly tests **RISK-PAY-01**. |

#### **Flow 2: UI Smoke Test Suite (Selenium)**

* **Objective:** To create a small set of high-value UI tests that verify critical end-to-end business flows are functional after a new deployment.

| **Scenario ID** | **Scenario Description** | **Automation Steps** |
| --- | --- | --- |
| **AUTO\_UI\_2.1** | **Happy Path - SO Creation & Approval:** Automate the complete Maker/Approver workflow for a new Standing Order. | 1. **(Maker)** Login as MAKER\_01. 2. Navigate to the Standing Order page. 3. Click "Add Standing Order". 4. Fill in all mandatory fields with valid data and submit. 5. Logout. 6. **(Approver)** Login as APPROVER\_01. 7. Navigate to the Task List. 8. Find the newly created SO and click to view details. 9. Click the "Approve" button and confirm. 10. **Assert** that the SO status changes to "Active". |
| **AUTO\_UI\_2.2** | **Happy Path - Transaction Search:** Automate the process of searching for a specific transaction. | 1. Login to the PU. 2. Navigate to the Transaction Monitoring page. 3. Use the filter controls to search by a known Billing ID or Transaction ID. 4. **Assert** that the results table updates and contains only the expected transaction record. |

### **8.4 Execution and Reporting**

* **Execution Strategy:** The full automation suite (API and UI) will be executed as a **nightly regression run** against the SIT environment. A smaller subset (the API suite) will be configured to run on-demand after every new build is deployed.
* **Reporting:** Test results will be automatically published from Jenkins. The report will include:
  + A dashboard summary of Pass/Fail percentages.
  + Execution time for each suite.
  + Detailed logs and screenshots for any failed UI test steps.
  + Detailed request/response logs for any failed API assertions.