### **SIT Test Approach & Methodology: Payment Module**

#### **1. Test Approach**

The System Integration Testing (SIT) for the Payment Module is a critical phase designed to validate the end-to-end transactional integrity and data flow between the Payment Uploader (PU) and its interconnected systems. Our approach is centered on a risk-based strategy, focusing on the complete business process lifecycle to ensure all components function cohesively as a single, unified system.

**1.1. Objective**

The primary objective of this SIT is to verify that:

* Data is passed accurately and securely between the PU, J-HUB, KONG, OVS-BAS, the Biller Aggregator, ENS, ALPHA, and DWH.
* The system correctly executes the complete, multi-step business workflows as defined in the Payment Flow.csv document.
* The application logic correctly handles a variety of data inputs, business rules, and failure conditions.
* The system meets all specified non-functional requirements for performance, security, and reliability.
* The system is stable and ready for the User Acceptance Testing (UAT) phase.

**1.2. Scope**

| **In-Scope** | **Out-of-Scope** |
| --- | --- |
| **End-to-End Workflow Validation:** Testing the full lifecycle of a payment transaction from Standing Order registration to payment execution, reconciliation, and regulatory reporting. | **Standalone Component Testing:** Unit tests performed by the development team. |
| **API & Interface Validation:** Verifying the data contracts and connectivity for all integrations managed via KONG and J-HUB. | **Performance of External Systems:** The internal performance and logic of the Biller Aggregator or Danamon Bank's systems. Testing is limited to the interface layer. |
| **Data-Driven Scenarios:** Testing with a wide range of biller types, fee structures, and transaction values based on the provided master data. | **Cosmetic GUI Testing:** Minor UI defects (e.g., pixel alignment, color schemes) are considered low priority for SIT. |
| **Error Handling & Exception Paths:** Simulating failures at each integration point to verify the system's error handling and manual repair functionalities. | **Full-scale Stress Testing:** A separate, dedicated performance testing phase will be conducted post-SIT if required. |
| **Role-Based Access Control:** Validating the Maker-Checker workflow and user permissions. |  |
| **Non-Functional Testing:** Validating performance, security, failover, and data retention as defined in this document. |  |

#### **2. Functional Testing Methodology**

Our methodology for functional testing is structured into four distinct, complementary layers to ensure comprehensive test coverage.

##### **2.1. Layer 1: Interface-Driven Integration Testing (The "Connections")**

This foundational layer validates the technical contracts and connectivity at each integration point.

* **Method:**
  + Using API testing tools, we will directly test the REST API endpoints managed by **KONG** and the service calls routed through **J-HUB**.
  + **Positive Testing:** We will send valid requests to each endpoint to confirm a successful (200 OK) response and verify that the data payload is correctly structured as per the technical specifications.
  + **Negative Testing:** We will test for failure conditions by sending requests with invalid data, missing authentication tokens, or incorrect parameters. The expected outcome is a graceful failure with an appropriate HTTP error code (4xx, 5xx) and a clear error message.
* **Key Interfaces to Test:**
  + PU → KONG → Biller Aggregator: Validate API calls for billing inquiry and payment execution.
  + PU → J-HUB → OVS-BAS: Validate service calls for Hold Amount (DD050) and Booking (RK170).
  + PU → ENS: Validate the trigger for sending email notifications.

##### **2.2. Layer 2: Workflow-Based End-to-End Testing (The "Business Process")**

This is the core of SIT, where we validate the system's ability to execute complete business processes by chaining together the verified interfaces. These scenarios are derived directly from the Payment Flow.csv.

* **Method:**
  + We will execute test cases that mirror the four primary payment sub-modules:
    1. **Registration Flow:** A test will follow a standing order from Maker creation, through Biller Aggregator validation, to Approver sign-off, culminating in the successful booking of the registration fee in the OVS-BAS simulator.
    2. **Payment Execution Flow:** This scenario will test the automated, multi-step process: scheduled billing inquiry, successful hold amount, payment confirmation from the biller, final debit from the customer account, and the triggering of a success notification to the ENS simulator.
    3. **Reconciliation Flow:** This test will validate the system's ability to request and process a reconciliation file from the Biller Aggregator simulator and correctly display the results.
    4. **Regulatory Flow:** A dedicated test will verify that the PU correctly sends the required regulatory data packet through KONG to the ALPHA simulator for consumption by REGLA.
* **Validation:** Success is measured by the successful completion of the entire workflow and the verification of correct data states in all involved systems (e.g., transaction status in the PU database, account balance in the OVS-BAS simulator).

##### **2.3. Layer 3: Data-Driven & Scenario-Based Functional Testing (The "Logic")**

This methodology focuses on testing the PU's internal logic and its ability to handle a wide variety of conditions.

* **Method:**
  + **Data Permutation:** We will create a test data matrix using information from the Bill Fee Payment Module.docx and Detail biller SYB.csv to test with:
    - Multiple biller categories (PDAM, Multifinance, BPJS, etc.).
    - Different fee structures (e.g., billers with and without an Administration Fee).
    - Boundary conditions, such as transactions at or just over the IDR 5 mio limit specified for some billers.
  + **Transaction Status Testing:** We will design specific scenarios to force every possible transaction status defined in the FDD, such as:
    - To test **"Balance Short"**: Configure the OVS-BAS simulator to return an "insufficient funds" response.
    - To test **"Paid by another merchant"**: Configure the Biller Aggregator simulator to respond that the bill has already been settled.
    - To test **"Overdue - Failed"**: Let a transaction with a system error pass its due date without being repaired.
  + **Manual Repair Functionality:** Test the user's ability to select a failed transaction and trigger the manual repair process, including the required Checker approval.

##### **2.4. Layer 4: Role-Based Security Testing (The "Users")**

This testing ensures the application is secure and that the Maker-Checker workflow is enforced correctly.

* **Method:**
  + **Permission Matrix Testing:** We will create a matrix of user roles (Maker, Checker) vs. application functions. Test cases will verify that users can only perform actions for which they are authorized. Key scenarios include:
    - A Maker cannot approve their own or any other Maker's submission.
    - A read-only user cannot see action buttons like "Add", "Edit", or "Approve".
  + **Negative Authorization:** We will attempt to access API endpoints directly with a token belonging to a user who does not have the required permissions and verify that the request is denied.

#### **3. Non-Functional Testing (NFT) Approach**

Non-Functional Testing focuses on *how well* the system operates under various conditions. The objective is to validate key attributes such as performance, security, and resilience against the requirements specified in the FDD and TDD.

##### **B1: Infrastructure Verification**

* **Objective:** To verify that the underlying technical environment is configured correctly as per the architecture defined in the TDD (Section 4).
* **Scenarios:**
  + **Scenario B1.1 (Container Orchestration Check):** Confirm that all PU microservices are deployed as Docker containers and are successfully managed by the Red Hat OpenShift platform.
  + **Scenario B1.2 (Database & Cache Connectivity):** Execute a basic transaction (e.g., log in) to verify that the application can successfully establish connections to the PostgreSQL database and the Redis cache.
  + **Scenario B1.3 (Message Broker Health Check):** Send a test message via the RabbitMQ/IBM ACE message broker to confirm that asynchronous communication channels are operational.
  + **Scenario B1.4 (Network Path & Firewall Validation):** In coordination with IT Operations, verify that network paths and firewall rules between the PU, KONG, J-HUB, and other integrated systems are open and allow for successful communication.

##### **B2: Capacity and Performance Verification**

* **Objective:** To verify that the PU system's performance is acceptable under expected load conditions and that it remains stable over extended periods.
* **Scenarios:**
  + **Scenario B2.1 (Load Testing):**
    - **Workload:** Simulate a realistic load based on the transaction volume projections in the FDD (starting at ~4,860 transactions/month). The test will concentrate this load within the primary business operation hours (08:00 – 16:00 WIB) to simulate peak activity.
    - **Validation:** Key API response times must remain within acceptable, pre-defined thresholds. System resource utilization (CPU, Memory) on the OpenShift cluster should not exceed 80%.
  + **Scenario B2.2 (Endurance / Soak Testing):**
    - **Workload:** Run a sustained, moderate load against the system for an extended duration (e.g., 8 hours) that mimics the daily operation window.
    - **Validation:** The system must show no degradation in response times or evidence of memory leaks over the test period.

##### **B3: Operations / Capability Verification**

* **Objective:** To ensure the system provides the necessary capabilities for security, data management, and operational support as specified in the TDD.
* **Scenarios:**
  + **Scenario B3.1 (Security Capability - Authentication):** Verify that all secured APIs managed by KONG reject requests lacking a valid JWT token. Test token expiration scenarios.
  + **Scenario B3.2 (Security Capability - Encryption):** Use network tools to confirm that all external traffic is encrypted using TLS/HTTPS.
  + **Scenario B3.3 (Data Management Capability - Archiving):** Create test transactions, manually age them in the database beyond the **6-month threshold**, trigger the archiving job, and verify the records are moved correctly from the live database to the archive database.
  + **Scenario B3.4 (Monitoring & Logging Capability - Log Validation):** Execute successful and failed transactions and check the centralized logging system (ELK Stack or similar) to verify that events are logged with the correct severity and detail, and that no sensitive data is exposed.
  + **Scenario B3.5 (Monitoring & Logging Capability - Alerting):** During the Load Test (B2), coordinate with IT Operations to confirm that pre-configured performance alerts in Prometheus/Grafana are triggered when thresholds are breached.

##### **B4: High Availability (HA) Verification**

* **Objective:** To validate the system's resilience and ability to withstand single-component failures within the primary data center, as per the HA design implicit in the TDD's containerized architecture.
* **Scenarios:**
  + **Scenario B4.1 (Application Instance Failover):**
    - **Test:** While running a light, constant load, manually terminate one of the active PU application containers on the OpenShift cluster.
    - **Validation:** The OpenShift Load Balancer should seamlessly redirect traffic to the remaining healthy instances with minimal to no failed requests. OpenShift should also automatically restart the terminated container to restore full redundancy.
  + **Scenario B4.2 (Supporting Service Node Failover):**
    - **Test:** If supporting services like Redis or PostgreSQL are deployed in a clustered HA configuration, simulate a failure of the primary node in coordination with IT Operations.
    - **Validation:** The system should automatically failover to a secondary/standby node without significant application downtime or data loss.

##### **B5: DRC (Disaster Recovery Center) Verification**

* **Objective:** To validate the system's ability to recover from a complete data center outage and meet the Recovery Time Objective (RTO) and Recovery Point Objective (RPO) defined in the FDD (Section 12.1.2).
* **Scenarios:**
  + **Scenario B5.1 (Full DC Failover and Fallback):**
    - **Test:** Execute the precise failover and fallback process flows documented in the FDD (Section 14.2.2). This involves simulating a full outage of the primary Production Data Centre (Prod DC).
    - **Validation:**
      1. The system becomes fully operational on the Disaster Recovery (DR) cluster within the specified RTO of **6 hours**.
      2. Data integrity is maintained, confirming no significant data loss (meeting the RPO).
      3. The documented fallback procedure to return operations to the restored Prod DC is successful.