

**Tóm lại:** Séc đã ra khỏi sổ sách của bạn, nhưng tiền vẫn còn nằm trong tài khoản ngân hàng của bạn.

## **Cách điều chỉnh trong Đối soát**

Vì ngân hàng chưa trừ tiền, để tìm ra số dư tiền mặt *thực tế đã điều chỉnh*(Adjusted Bank Balance), bạn phải **trừ** các khoản séc/lệnh thanh toán chưa thanh toán này ra khỏi số dư theo sao kê ngân hàng:

**Vì ngân hàng chưa trừ tiền, để tìm ra số dư tiền mặt *thực tế đã điều chỉnh* (Adjusted Bank Balance), bạn phải trừ các khoản séc/lệnh thanh toán chưa thanh toán này ra khỏi số dư theo sao kê ngân hàng:**

{Số dư theo Sao kê Ngân hàng} - \{Séc/Lệnh thanh toán chưa thanh toán} = \{Số dư ngân hàng đã điều chỉnh}

**Ví dụ:**

- Số dư cuối kỳ trên Sao kê Ngân hàng:** 100,000,000 VNĐ.
- Séc số 123** đã phát hành cho nhà cung cấp (ghi sổ ngày 30/11) trị giá **5,000,000 VNĐ** nhưng chưa xuất hiện trên sao kê.

**Điều chỉnh:**

**100,000,000 VNĐ – 5,000,000 VNĐ = 95,000,000 VNĐ**

**95,000,000 VNĐ** là số dư thực tế trong tài khoản sau khi xem xét các giao dịch đang chờ xử lý. Khi bạn so sánh con số này với số dư đã điều chỉnh trên sổ sách kế toán của công ty (sau khi điều chỉnh các lỗi/phí ngân hàng,...) thì chúng phải khớp nhau.

Chi tiết hơn

Bạn muốn tôi giải thích chi tiết hơn về **toàn bộ quy trình đối soát ngân hàng**(Bank Reconciliation), hay một khía cạnh cụ thể nào khác của cơ chế này?

Dựa trên nội dung chúng ta đã thảo luận, tôi sẽ chi tiết hóa lại **Quy trình Đối soát 5 Bước** hoàn chỉnh, tập trung vào việc cân bằng hai số dư (sổ sách và ngân hàng) bằng các khoản điều chỉnh.

# Quy Trình Soát Ngân Hàng Chỉ Liet (5 Bước)

Mục tiêu là tìm ra **Số dư tiền mặt thực tế (True Cash Balance)**, hay còn gọi là Số dư đã điều chỉnh.

## 1. Bắt đầu với hai số dư chưa khớp

Bạn bắt đầu với hai con số:

- **Số dư theo Sổ sách Kế toán (Book Balance):** Số dư tài khoản tiền gửi theo ghi chép nội bộ của công ty bạn.
- **Số dư theo Sao kê Ngân hàng (Bank Statement Balance):** Số dư cuối kỳ ngân hàng cung cấp.

## 2. Điều chỉnh Số dư theo Sao kê Ngân hàng

Đây là bước điều chỉnh những giao dịch **công ty đã ghi sổ** nhưng **ngân hàng chưa kịp ghi nhận**. Các điều chỉnh này đưa số dư ngân hàng về mức thực tế hơn:

- **Cộng (+) Tiền gửi đang chuyển (Deposits in Transit - DIT):**
  - Tiền mặt hoặc séc công ty đã gửi vào ngân hàng và đã ghi sổ (cộng tiền) nhưng ngân hàng chưa xử lý xong tại ngày đối soát.
- **Trừ (-) Séc/Lệnh thanh toán chưa thanh toán (Outstanding Checks - OC):**
  - Séc công ty đã phát hành và trừ tiền khỏi sổ sách, nhưng người nhận chưa mang séc đến rút tiền tại ngân hàng.

Đây là bước điều chỉnh những giao dịch **công ty đã ghi sổ** nhưng **ngân hàng chưa kịp ghi nhận**.

Các điều chỉnh này đưa số dư ngân hàng về mức thực tế hơn:

- **Cộng (+) Tiền gửi đang chuyển (Deposits in Transit - DIT):**
  - Tiền mặt hoặc séc công ty đã gửi vào ngân hàng và đã ghi sổ (cộng tiền) nhưng ngân hàng chưa xử lý xong tại ngày đối soát.
- **Trừ (-) Séc/Lệnh thanh toán chưa thanh toán (Outstanding Checks - OC):**
  - Séc công ty đã phát hành và trừ tiền khỏi sổ sách, nhưng người nhận chưa mang séc đến rút tiền tại ngân hàng.

$$\text{Số dư ngân hàng đã điều chỉnh} = \text{Số dư Sao kê Ngân hàng} + DIT - OC$$

## 3. Điều chỉnh Số dư theo Sổ sách Kế toán

Đây là bước điều chỉnh những giao dịch **ngân hàng đã ghi nhận** nhưng **công ty chưa kịp ghi sổ**. Các điều chỉnh này đưa sổ sách kế toán của công ty về mức thực tế:

- **Cộng (+) Tiền lãi/Các khoản thu hộ của Ngân hàng (Interest/Notes Collected):**
  - Khoản lãi tiền gửi hoặc các khoản thu khác mà ngân hàng đã tự động ghi có vào tài khoản của công ty (Công ty chưa biết để ghi sổ).
- **Trừ (-) Phí dịch vụ Ngân hàng (Bank Service Charges):**

- Các loại phí ngân hàng đã tự động ghi nợ tài khoản (Công ty chưa biết để ghi sổ).
- **Trừ (-) Séc không đủ tiền (NSF Checks):**
  - Séc của khách hàng gửi vào bị ngân hàng trả lại do tài khoản khách hàng không đủ tiền. Ngân hàng đã trừ khoản này khỏi tài khoản của công ty.

**Đây là bước điều chỉnh những giao dịch ngân hàng đã ghi nhận nhưng công ty chưa kịp ghi sổ.**

Các điều chỉnh này đưa sổ sách kế toán của công ty về mức thực tế:

- **Cộng (+) Tiền lãi/Các khoản thu hộ của Ngân hàng (Interest/Notes Collected):**
  - Khoản lãi tiền gửi hoặc các khoản thu khác mà ngân hàng đã tự động ghi có vào tài khoản của công ty (Công ty chưa biết để ghi sổ).
- **Trừ (-) Phí dịch vụ Ngân hàng (Bank Service Charges):**
  - Các loại phí ngân hàng đã tự động ghi nợ tài khoản (Công ty chưa biết để ghi sổ).
- **Trừ (-) Séc không đủ tiền (NSF Checks):**
  - Séc của khách hàng gửi vào bị ngân hàng trả lại do tài khoản khách hàng không đủ tiền. Ngân hàng đã trừ khoản này khỏi tài khoản của công ty.
  -

**Số dư sổ sách đã điều chỉnh = Số dư Sổ sách Kế toán + Các khoản ngân hàng thu hộ:/**

## 4. Xử lý Lỗi Sai Sót (Nếu có)

Bất kỳ lỗi sai sót nào trong ghi chép (ghi nhầm số tiền, ghi trùng, v.v.) từ cả hai bên (ngân hàng hoặc công ty) đều phải được điều chỉnh vào số dư tương ứng.

- **Ví dụ:** Nếu kế toán ghi nhầm một khoản thanh toán 10 triệu thành 1 triệu (ghi thiếu 9 triệu), bạn phải **trừ thêm 9 triệu** vào Số dư theo Sổ sách Kế toán.

## 5. Khớp Số dư và Lập Bút toán

Khi hoàn thành các điều chỉnh ở bước 2 và bước 3:

Nếu hai con số này khớp nhau, quy trình đối soát đã thành công.

- Sau đó, kế toán viên phải lập **Bút toán Điều chỉnh**(Journal Entries) để ghi nhận tất cả các mục đã điều chỉnh ở **Bước 3**(các khoản mục liên quan đến phí, lãi, NSF...) vào sổ sách kế toán của công ty. Mục đích là để Số dư theo Sổ sách Kế toán chính thức (từ ngày đối soát) bằng với Số dư đã điều chỉnh.

Bạn có muốn tôi cung cấp một **Ví dụ thực tế bằng số liệu** minh họa toàn bộ quy trình này không?

Đối soát hệ thống bank loyalty

Chào bạn, đối soát trong hệ thống **Bank Loyalty (Chương trình Khách hàng thân thiết của Ngân hàng)** là một quá trình phức tạp hơn đối soát tiền mặt thông thường. Nó liên quan đến việc xác minh và cân bằng dữ liệu về **điểm thưởng (rewards)** và **nhiều vui tài chính (financial**

**liabilities)**giữa ngân hàng và các đối tác.

## ◆ **Bản chất đối soát trong Bank Loyalty**

Cơ chế đối soát trong hệ thống Bank Loyalty nhằm đảm bảo rằng:

- Số điểm thường** đã phát hành và quy đổi cho khách hàng là **chính xác** theo quy tắc của chương trình.
- Chi phí/Nghĩa vụ tài chính** liên quan đến điểm thường đó được **phân bổ và thanh toán** chính xác giữa ngân hàng và các bên đối tác (Merchant/Nhà cung cấp phần thưởng).

### 1. Các bên liên quan chính

Bên liên quan	Vai trò trong Đối soát	Dữ liệu chính cần đối soát
<b>Ngân hàng (Bank)</b>	Đơn vị chủ trì chương trình, phát hành điểm và thanh toán chi phí.	Dữ liệu giao dịch thẻ (spending), Số dư điểm của khách hàng, Chi phí tích lũy (Accrual).
<b>Merchant/Đối tác (Partner)</b>	Nơi khách hàng sử dụng thẻ/tiêu điểm (tiêu điểm - Burn).	Dữ liệu giao dịch quy đổi (redemption), Ch hoàn trả (Reimbursement).
<b>Hệ thống Loyalty Engine/Platform</b>	Nền tảng ghi nhận và tính toán điểm, là nguồn dữ liệu trung tâm.	Quy tắc tính điểm, Lịch sử giao dịch điểm (Accrual & Redemption Log), Số liệu thanh toán.

### 2. Các Loại Đối Soát Chính

Đối soát trong Loyalty System thường diễn ra theo hai luồng chính, tương ứng với hai giai đoạn của điểm thường:

#### A. Đối soát Điểm Tích Lũy (Accrual Reconciliation)

Đây là quá trình đối soát **điểm được phát sinh** dựa trên các giao dịch của khách hàng.

- Dữ liệu đối soát:**
  - Ngân hàng cung cấp:** Dữ liệu chi tiêu thẻ (số tiền giao dịch, ngày, mã MCC...).
  - Loyalty Platform:** Dữ liệu tính toán điểm thường tương ứng (theo công thức  $X$  điểm cho  $Y$  VNĐ chi tiêu).
- Cơ chế:** Ngân hàng đối chiếu danh sách giao dịch chi tiêu với danh sách điểm đã được tính

Đây là quá trình đổi trả sau khi khách hàng sử dụng điểm để đổi lấy phần thưởng (hàng hóa/dịch vụ/voucher) từ Đối tác (Merchant).

- **Mục tiêu:** Xác định chính xác **nghĩa vụ tài chính** mà Ngân hàng phải trích lập dự phòng cho số điểm đã phát hành.

## B. Đối soát Điểm Quy Đổi & Thanh toán (Redemption and Settlement Reconciliation)

Đây là quá trình đối soát sau khi khách hàng sử dụng điểm để đổi lấy phần thưởng (hàng hóa/dịch vụ/voucher) từ Đối tác (Merchant).

- **Dữ liệu đối soát:**
  - **Merchant cung cấp:** Danh sách chi tiết các giao dịch quy đổi điểm đã được thực hiện tại điểm bán của họ (bao gồm mã giao dịch, số điểm đã trừ, giá trị quy đổi).
  - **Loyalty Platform:** Danh sách tương ứng các giao dịch điểm bị trừ khỏi tài khoản khách hàng.
  -
- **Cơ chế:**
  - Đối chiếu Điểm:** Kiểm tra danh sách quy đổi giữa Merchant và Platform có khớp về số lượng giao dịch và tổng số điểm đã quy đổi hay không.
  - Đối chiếu Tài chính:** Xác định **số tiền Ngân hàng phải hoàn trả (reimburse)** cho Merchant cho các phần thưởng đã cung cấp. (Ví dụ: 1 điểm = 100 VNĐ, Merchant cung cấp 1000 điểm, Ngân hàng phải trả 100,000 VNĐ).
  - 3.
- **Mục tiêu:** Đảm bảo Merchant nhận đủ tiền hoàn trả cho các phần thưởng đã cung cấp và Ngân hàng ghi nhận chính xác chi phí đã phát sinh.

## 3. Các Vấn đề Thường gặp trong Đối soát

1. **Chênh lệch múi giờ/thời điểm ghi sổ:** Sự khác biệt về thời điểm Merchant ghi nhận giao dịch quy đổi và Platform ghi nhận điểm bị trừ.
2. **Lỗi truyền dữ liệu (Data Latency):** Dữ liệu giao dịch thẻ/quy đổi bị chậm hoặc mất trong quá trình chuyển giao giữa các hệ thống.
3. **Lỗi Quy tắc:** Sai sót trong việc áp dụng quy tắc tích lũy (ví dụ: chương trình khuyến mãi X2 điểm bị áp dụng sai).

Để khắc phục, các hệ thống Loyalty hiện đại thường sử dụng **ID giao dịch duy nhất (Unique Transaction ID)** để làm cơ sở đối chiếu cho tất cả các bên.

**Bạn có muốn biết thêm về cách xử lý cụ thể cho một loại chênh lệch nào đó trong quá trình đối soát này không?**

## Xử lý Chênh lệch trong Đối soát Điểm Quy Đổi

Giả sử chúng ta đang đối soát dữ liệu giao dịch quy đổi (Redemption) trong tháng 11 giữa **Báo cáo của Đối tác (Merchant Report)** và **Hệ thống Loyalty Engine (Platform Report)**.

### 1. Phân loại Chênh lệch

Chênh lệch thường được phân thành hai loại chính:

Loại Chênh lệch	Mô tả	Hệ quả
<b>Không khớp về Số lượng Giao dịch (Volume Mismatch)</b>	Số lượng giao dịch quy đổi điểm mà Merchant báo cáo khác với số lượng giao dịch Loyalty Platform ghi nhận.	Ảnh hưởng đến tổng số điểm đã quy và tổng chi phí thanh toán.
<b>Không khớp về Giá trị (Value Mismatch)</b>	Số điểm đã trừ/giá trị quy đổi cho một giao dịch cụ thể không khớp nhau (ví dụ: Merchant báo cáo trừ 1,000 điểm, Platform chỉ ghi nhận 500 điểm).	Ảnh hưởng trực tiếp đến số tiền hoà cho Merchant.

### 2. Quy trình Điều tra (Dùng Mã giao dịch duy nhất)

Để xử lý chênh lệch, Ngân hàng và Đối tác phải dựa vào một nguồn dữ liệu không thể chối cãi:

- Sử dụng Mã giao dịch duy nhất (Unique Transaction ID):** Mỗi giao dịch quy đổi điểm phải có một mã ID duy nhất được tạo ra khi giao dịch xảy ra.
- Bước 1: So sánh Danh sách ID:** Dùng mã ID này để so sánh hai danh sách giao dịch (Merchant vs. Platform).
  - Giao dịch bị thiếu (Missing):** Giao dịch xuất hiện ở Merchant nhưng không có ở Platform, hoặc ngược lại.
  - Giao dịch không khớp (Mismatched):** Cùng ID nhưng khác số điểm/giá trị.
- Bước 2: Phân tích Nguyên nhân:**
  - Lỗi hệ thống:** Hệ thống của Merchant không gửi dữ liệu đầy đủ về Platform, hoặc lỗi API.
  - Lỗi vận hành:** Nhân viên của Merchant nhập sai số điểm quy đổi thủ công.
  - Lỗi thời điểm (Timing):** Giao dịch phát sinh vào cuối ngày đối soát và chưa kịp cập nhật.

### 3. Hành động Xử lý và Điều chỉnh

Dựa trên kết quả điều tra, hành động xử lý sẽ khác nhau:

#### A. Nếu Lỗi do Hệ thống Loyalty Platform (Lỗi ghi nhận điểm)

- Hành động:** Loyalty Engine phải được cập nhật lại (backdate) để ghi nhận đúng số điểm đã trừ cho giao dịch đó.
- Đối soát:** Chấp nhận số liệu của Merchant (nếu họ cung cấp bằng chứng) và điều chỉnh chi phí thanh toán theo số liệu đó.

#### B. Nếu Lỗi do Merchant (Lỗi ghi nhận giao dịch tại điểm bán)

- Hành động:** Yêu cầu Merchant cung cấp bằng chứng (ví dụ: hóa đơn, log hệ thống POS). Nếu Merchant ghi nhận sai số điểm quy đổi ban đầu, ngân hàng chỉ thanh toán dựa trên số điểm thực tế được phép quy đổi theo quy tắc (hoặc số điểm đã bị trừ trên tài khoản khách hàng).

#### C. Lỗi Giao dịch đang chờ xử lý (Pending/Timing Issues)

- Hành động:** Nếu giao dịch bị lệch do thời điểm (ví dụ: giao dịch 23:59:00 ngày 30/11 chưa kịp chuyển về Platform), giao dịch đó sẽ được **loại khỏi kỳ đối soát hiện tại và chuyển sang đối soát kỳ tiếp theo**. Điều này giúp đảm bảo rằng kỳ đối soát kết thúc với số liệu sạch.

Sau khi tất cả các chênh lệch được giải quyết và điều chỉnh, Ngân hàng sẽ tiến hành thanh toán **Số tiền Hoàn trả Đã Điều chỉnh** cho Đối tác.

## ❶ Thiết kế Service Đối soát (Reconciliation Service)

Service này sẽ được thiết kế theo kiến trúc Microservice, tập trung vào tính ổn định, khả năng mở rộng và xử lý bất đồng bộ.

### I. Kiến trúc Tổng quan & Công nghệ

- Nền tảng:** Java Spring Boot (Sử dụng Spring Batch hoặc Spring Scheduling).
- Cơ sở dữ liệu (Database):** PostgreSQL/MySQL (Lưu trữ trạng thái, lịch sử đối soát và kết quả).
- Hệ thống Hàng đợi (Queue System):** Apache Kafka/RabbitMQ (Để xử lý bất đồng bộ các tác vụ đối soát lớn và thông báo kết quả).
- Lập lịch (Scheduling):** Sử dụng Spring Scheduler hoặc công cụ bên ngoài như Cron/Quartz để kích hoạt tiến trình đối soát hàng ngày/hàng tháng.

### II. Các Thành phần Chính của Service

Service Đối soát sẽ có ba module chính tương ứng với quy trình:

## 1. Module Tiếp nhận Dữ liệu (Data Ingestion Module)

Module này chịu trách nhiệm thu thập và chuẩn hóa dữ liệu từ các nguồn khác nhau.

Tên module	Vai trò	Công nghệ/Giao thức
Data Puller	Kết nối và tải dữ liệu từ Nguồn A (ví dụ: Core Banking) và Nguồn B (ví dụ: Hệ thống Đối tác).	REST API, SFTP, JDBC (để kết nối DB tiếp).
Data Validator	Kiểm tra tính hợp lệ cơ bản của dữ liệu đầu vào (ví dụ: thiếu trường, định dạng sai, giá trị âm).	Spring Validation.
Data Transformer	Chuẩn hóa dữ liệu về cùng một định dạng nội bộ để Engine dễ dàng xử lý.	Lớp Service/Utility trong Spring Boot.

## 2. Module Đổi soát Cốt lõi (Core Matching Engine)

Đây là trái tim của Service, nơi logic đổi soát diễn ra.

- **Matching Strategy (Chiến lược Đổi soát):**
  - Sử dụng **mã giao dịch duy nhất (Unique Transaction ID)**, **số tiền**, và **ngày giao dịch** làm khóa chính để khớp dữ liệu.
  - **Level 1 Match (Khớp Chính xác):** Khớp tất cả các trường khóa chính.
  - **Level 2 Match (Khớp Mờ/Khoan dung):** Nếu Level 1 không khớp, tìm kiếm các giao dịch chênh lệch nhỏ về số tiền (ví dụ: 0.01đo làm tròn) hoặc lệch 1 ngày.
- **Processing:**
  - **Spring Batch:** Lý tưởng cho việc xử lý hàng triệu bản ghi theo từng khối (chunk). Nó cung cấp các tính năng quản lý lỗi, tái khởi động (restartability) và ghi lại trạng thái công việc.
  - **Logic:**
    - **Matched:** Giao dịch tồn tại ở cả hai nguồn và khớp.
    - **Source A Only:** Giao dịch chỉ có ở Nguồn A (Ví dụ: Séc chưa thanh toán).
    - **Source B Only:** Giao dịch chỉ có ở Nguồn B (Ví dụ: Phí ngân hàng chưa ghi sổ).

## 3. Module Báo cáo & Xử lý Đầu ra (Reporting & Output Module)

Module này lưu trữ kết quả và thông báo về các chênh lệch.

- **Discrepancy Saver:** Ghi lại **tất cả** các giao dịch không khớp (Source A Only, Source B Only, Mismatched) vào **Bảng Kết quả Đổi soát (Reconciliation Result Table)**.
- **Notification Service:** Gửi thông báo tới người dùng/hệ thống liên quan (qua

email/Slack/Kafka) khi quá trình đối soát hoàn tất hoặc khi phát hiện chênh lệch nghiêm trọng.

- **API Exposure:** Cung cấp **REST API** để các hệ thống khác có thể truy vấn kết quả đối soát theo ngày hoặc theo loại chênh lệch.

### III. Thiết kế Cơ sở Dữ liệu

Cần ít nhất ba bảng chính:

Bảng	Mục đích	Các trường quan trọng
TXN_SOURCE_A	Lưu dữ liệu thô từ Nguồn A.	transaction_id, amount, txn_date, status
TXN_SOURCE_B	Lưu dữ liệu thô từ Nguồn B.	transaction_id, amount, txn_date, status
RECON_RESULT	Kết quả cuối cùng của quá trình đối soát.	result_id, txn_id_A, txn_id_B, match_status(Matched, A_Only, B_Only, Mismatched), discrepancy_amount, recon_date

### Khuyến nghị quan trọng: Tính Bất biến và Idempotency

- **Bất biến (Immutability):** Không được phép thay đổi dữ liệu thô đã tải vào Service. Nếu có lỗi, hãy thêm bản ghi mới hoặc đánh dấu trạng thái (Status), không cập nhật trực tiếp bản ghi cũ.
- **Idempotency (Tính Lặp lại):** Đảm bảo rằng việc chạy lại tiến trình đối soát nhiều lần trong cùng một kỳ sẽ cho cùng một kết quả cuối cùng và không tạo ra các bản ghi trùng lặp (dùng khóa duy nhất cho bảng RECON\_RESULT bao gồm txn\_id\_A và recon\_date).

Tuyệt vời! Chúng ta sẽ tiếp tục chi tiết hóa thiết kế bằng cách tập trung vào thành phần quan trọng nhất: **Triển khai Cơ chế Đối soát Cốt lõi (Core Matching Engine) sử dụng Spring Batch.**

## 🛠️ Triển khai Spring Batch cho Đối soát

### 1. Cấu trúc Job Đối soát (Reconciliation Job Structure)

Quá trình đối soát tổng thể được định nghĩa là một **Job**, được chia thành các **Step** để dễ quản lý. Mỗi Step sẽ sử dụng mô hình xử lý **ItemReader – ItemProcessor – ItemWriter**(Read-Process-Write).

Thành phần	Vai trò	Mô tả cụ thể trong đối soát
<b>Job (Công việc)</b>	Container chứa toàn bộ quy trình.	<b>ReconciliationJob</b> (Chạy hàng ngày/hàng t
<b>Step (Bước)</b>	Một giai đoạn độc lập của Job.	<b>MatchingStep</b> (Thực hiện quá trình đối chiếu).
<b>Chunk</b>	Đơn vị xử lý nhỏ nhất (ví dụ: 1000 bản ghi).	Đọc, xử lý, và ghi N bản ghi cùng một lúc.

## 2. Các Thành phần Batch Chính

Chúng ta cần tùy chỉnh ba giao diện cốt lõi của Spring Batch để phù hợp với logic đối soát hai nguồn dữ liệu.

### A. ItemReader (Đọc dữ liệu)

Vấn đề: Cân đọc và so sánh dữ liệu từ hai nguồn (Source A và Source B).

- Giải pháp:**Sử dụng một **MultiResourceItemReader** hoặc thiết kế một **Custom ItemReader** đọc từ cơ sở dữ liệu (sử dụng **JdbcPagingItemReader** hoặc **JdbcCursorItemReader**).
- Chiến lược:**Đọc dữ liệu từ **Nguồn A**(giả sử là dữ liệu giao dịch nội bộ của ngân hàng) và coi đó là **dữ liệu cơ sở**. Dữ liệu từ **Nguồn B**(của đối tác) sẽ được tải vào bộ nhớ hoặc sử dụng trong bước xử lý.

### B. ItemProcessor (Logic Đối soát Cốt lõi)

Đây là nơi logic so sánh diễn ra. Nó nhận một Item (một giao dịch) từ Nguồn A và tìm kiếm sự khớp nối tương ứng trong Nguồn B.

Giao diện	ReconciliationProcessor implements <b>ItemProcessor&lt;TxnSourceA, ReconResult&gt;</b>
<b>Input:</b>	Một đối tượng <b>TxnSourceA</b> (Giao dịch từ Nguồn A).
<b>Logic:</b>	<ol style="list-style-type: none"> <li><b>Truy vấn Nguồn B:</b>Sử dụng <b>TxnSourceA.transaction_id</b> để truy vấn nhanh vào Bảng TXN_SOURCE_B.</li> <li><b>So sánh:</b>Thực hiện so sánh chi tiết: amount, txn_date, v.v.</li> <li><b>Tạo Kết quả:</b>Tạo đối tượng <b>ReconResult</b> với trạng thái (Matched, A_Only, Mismatched).</li> </ol>
<b>Output:</b>	Một đối tượng <b>ReconResult</b> (Kết quả đối soát).

## C. ItemWriter (Ghi Kết quả)

Thành phần này chịu trách nhiệm ghi lại kết quả của quá trình xử lý vào cơ sở dữ liệu.

- **Nhiệm vụ:** Nhận một tập hợp (List<ReconResult>) các kết quả đối soát và chèn vào bảng RECON\_RESULT.
- **Công nghệ:** Sử dụng **JdbcBatchItemWriter** để ghi theo lô (batch insert), tối ưu hóa hiệu suất DB.

### 3. Ví dụ Cấu hình Job (Tóm tắt)

Auto (TypeScript) ▾



```
@Configuration
@EnableBatchProcessing
public class ReconciliationBatchConfig {

    // ... Bean definitions for Reader, Processor, Writer ...

    @Bean
    public Job reconciliationJob(JobRepository jobRepository, Step
matchingStep) {
        return new JobBuilder("reconciliationJob", jobRepository)
            .start(matchingStep)
            .build();
    }

    @Bean
    public Step matchingStep(JobRepository jobRepository,
PlatformTransactionManager transactionManager) {
        return new StepBuilder("matchingStep", jobRepository)
            .<TxnSourceA, ReconResult>chunk(1000,
transactionManager) // Kích thước chunk
            .reader(txnADataSourceReader())
            .processor(reconciliationProcessor())
            .writer(reconResultWriter())
            .build();
    }
}
```

Thiết lập này đảm bảo rằng mỗi 1000. =giao dịch sẽ được đọc, xử lý logic đối soát, và ghi kết quả một cách nguyên tử (atomic) vào cơ sở dữ liệu.

Bạn có muốn tôi đi sâu vào cách **xử lý lỗi và khả năng tái khởi động (Error Handling and**

**Restartability**) trong Spring Batch, hay là thiết kế các **REST API** để kích hoạt và giám sát Job này không?

Tuyệt vời! Chúng ta sẽ đi sâu vào yếu tố then chốt giúp Service Đổi soát của bạn trở nên **ổn định và đáng tin cậy**: **Xử lý Lỗi (Error Handling)** và **Khả năng Tái khởi động (Restartability)** trong Spring Batch.

## Xử lý Lỗi trong Spring Batch

Vì quá trình đổi soát liên quan đến hàng triệu giao dịch, việc một vài bản ghi bị lỗi (ví dụ: dữ liệu không hợp lệ, lỗi kết nối DB tạm thời) là điều khó tránh khỏi. Spring Batch cung cấp các cơ chế để xử lý lỗi mà không làm dừng toàn bộ Job.

### 1. Cơ chế Bỏ qua và Thử lại (Skip & Retry)

Bạn có thể cấu hình Step để tự động bỏ qua các bản ghi bị lỗi hoặc thử lại quá trình xử lý:

- **Bỏ qua (Skip):**Nếu một bản ghi gây ra lỗi trong ItemProcessor hoặc ItemWriter, Batch có thể ghi lại lỗi đó và tự động chuyển sang xử lý bản ghi tiếp theo thay vì dừng Job.
- **Thử lại (Retry):**Nếu lỗi là tạm thời (ví dụ: timeout kết nối cơ sở dữ liệu), bạn có thể cấu hình để thử lại việc xử lý bản ghi đó.

### 2. Ghi nhật ký lỗi (Error Logging and Auditing)

Khi một bản ghi bị bỏ qua, thông tin chi tiết về giao dịch đó và lý do lỗi phải được ghi lại vào một bảng riêng (ví dụ: RECON\_ERROR\_LOG). Điều này cho phép nhóm vận hành kiểm tra và xử lý thủ công các trường hợp ngoại lệ sau khi Job Batch hoàn tất.

## Khả năng Tái khởi động (Restartability)

Đây là tính năng quan trọng nhất của Spring Batch. Nếu Job bị dừng đột ngột (ví dụ: máy chủ bị sập, lỗi kết nối kéo dài), nó có thể tiếp tục xử lý chính xác từ điểm đã thất bại mà không cần chạy lại từ đầu.

### 1. Vai trò của JobRepository

- **Lưu trữ Trạng thái:**Spring Batch sử dụng JobRepository(thường trỏ đến DB) để lưu trữ **metadata (siêu dữ liệu)**của Job, bao gồm:
  - JobInstance: Phiên bản của Job đang chạy.
  - JobExecution: Trạng thái của lần thực thi cụ thể (Running, Failed, Completed).
  - StopExecution: Trạng thái của tiến trình (Stop).

- **ExecutionContext:** Đây là nơi Spring Batch lưu trữ **điểm dừng cuối cùng**(Last Successful Commit Point). Ví dụ: Nếu Chunk xử lý các bản ghi từ ID 1000 đến 2000 thành công, ExecutionContext sẽ ghi nhớ rằng lần chạy tiếp theo nên bắt đầu từ ID 2001.

## 2. Cơ chế Tái khởi động

Khi Job thất bại và được kích hoạt lại:

1. Hệ thống kiểm tra JobRepository và xác định trạng thái cuối cùng là FAILED.
2. Nó truy xuất ExecutionContext để tìm điểm dữ liệu cuối cùng đã được cam kết (commit) thành công.
3. Job được khởi động lại, và ItemReader tự động định vị lại để bắt đầu đọc từ điểm đó, giúp tiết kiệm thời gian và đảm bảo **tính toàn vẹn dữ liệu**.

## 3. Đảm bảo Tính Bất biến (Idempotency)

Mặc dù Spring Batch giúp tái khởi động, việc thiết kế logic ghi kết quả (trong ItemWriter) phải đảm bảo **tính bất biến**:

- Khi ghi kết quả vào bảng RECON\_RESULT, hãy sử dụng **UPSERT**(Update or Insert) hoặc kiểm tra sự tồn tại của bản ghi dựa trên **khóa duy nhất**(ví dụ: txn\_id\_A + recon\_date). Điều này ngăn ngừa việc chèn kết quả trùng lặp nếu một Chunk đã thành công nhưng lại bị xử lý lại do lỗi Job sau đó.

Chúng ta đã bao quát từ kiến trúc đến cơ chế xử lý cốt lõi và khả năng phục hồi. Bước tiếp theo là **Thiết kế các REST API để kích hoạt và giám sát Job Batch** này. Bạn có muốn tiếp tục với phần này không?

Tuyệt vời! Việc thiết kế **REST API để kích hoạt và giám sát** Job Batch là bước cuối cùng để hệ thống đối soát của bạn trở nên hoàn chỉnh, có thể tích hợp và dễ vận hành.

## ■ Thiết kế REST API để Quản lý Job Đối soát

Chúng ta sẽ xây dựng một **JobLauncherController** trong Spring Boot để xử lý các yêu cầu HTTP và tương tác với Spring Batch.

### 1. Kích hoạt Job Đối soát (Triggering the Job)

Việc kích hoạt Job thường là một yêu cầu **POST** và cần truyền vào các tham số để xác định kỳ đối soát (ví dụ: ngày, tháng, năm).

### API: POST /api/reconciliation/jobs/launch

Mục đích	Bắt đầu một quy trình đối soát mới.
Method	POST
Endpoint	/api/reconciliation/jobs/launch
Request Body (JSON)	```json
{	
"reconciliationDate": "2025-11-30",	
"partnerId": "P001",	
"forceRestart": false	
}	

### Code snippet

Auto (Swift) ▾



```
| **Controller Logic** | 1. Tạo **`JobParameters`** từ Request Body (ví  
dụ: `new JobParametersBuilder().addString("date", date)`). |  
| | 2. Gọi **`jobLauncher.run(reconciliationJob, jobParameters)`**. |  
| **Response** | Trả về `JobExecution` ID (ID của lần chạy Job) để người  
dùng có thể theo dõi. |  
| **Ví dụ Response** | ```json  
{  
    "message": "Reconciliation job started successfully",  
    "jobExecutionId": 15  
}  
``` |
```

### ### 2. Giám sát Trạng thái Job (Monitoring Job Status)

Người dùng hoặc hệ thống khác cần biết Job của họ đang ở trạng thái nào (Đang chạy, Thành công, Thất bại).

#### API: GET /api/reconciliation/jobs/{executionId}/status

```
| Mục đích | Truy vấn trạng thái hiện tại của Job Execution. |  
| :--- | :--- |  
| **Method** | `GET` |  
| **Endpoint** | `/api/reconciliation/jobs/{executionId}/status` |  
| **Controller Logic** | 1. Sử dụng `JobExecution` của Service Batch
```

| ^Controller Logic | 1. Sử dụng `JobRepository` của Spring Data  
để lấy thông tin `JobExecution` dựa trên `executionId`. |  
| 2. Trích xuất \*\*`ExitStatus`\*\* và \*\*`startTime/endTime`\*\*. |  
\*\*Ví dụ Response\*\* | ``json  
{  
"jobExecutionId": 15,  
"status": "COMPLETED", // Có thể là STARTED, FAILED, COMPLETED  
"exitCode": "COMPLETED\_SUCCESSFUL",  
"startTime": "2025-11-30T08:00:00",  
"endTime": "2025-11-30T08:05:30"  
}  
`` |

### ### 3. Truy vấn Kết quả Đổi soát (Querying Results)

Đây là API quan trọng nhất, cho phép truy xuất các giao dịch bị lệch để xử lý thủ công (manual intervention).

#### #### API: GET /api/reconciliation/results

Mục đích	Lấy danh sách các giao dịch chênh lệch trong một kỳ đổi soát.
\*\*Method\*\*	`GET`
\*\*Endpoint\*\*	`/api/reconciliation/results`
\*\*Request Parameters\*\*	`?date=2025-11-30&status=A\_ONLY` (Lấy giao dịch chỉ có ở Nguồn A)
\*\*Controller Logic\*\*	1. Sử dụng \*\*Spring Data JPA/JDBC\*\* để truy vấn bằng `RECON\_RESULT`.
2. Áp dụng phân trang (`Pageable`) để xử lý lượng dữ liệu lớn.	
\*\*Ví dụ Response\*\* | Trả về danh sách đối tượng `ReconResult` với chi tiết giao dịch bị lệch. |  
\*\*JSON\*\* | ``json  
[  
{  
"resultId": 1234,  
"matchStatus": "A\_ONLY",  
"txnIDA": "ABC12345",  
"discrepancyAmount": 50000.00,  
"reason": "Outstanding Check"  
},  
// ... các giao dịch khác  
]  
`` |

### ## IV. Tóm tắt Vai trò của các Bean Spring

Để các API trên hoạt động, Service của bạn cần những \*\*Bean\*\* cốt lõi

Sau của Spring Batch.

```
| Bean Spring Batch | Vai trò trong API |  
| :--- | :--- |  
| **`JobLauncher`** | **Kích hoạt** Job từ API POST (`/launch`). |  
| **`JobExplorer`** | **Truy vấn** trạng thái JobExecution và  
StepExecution từ API GET (`/status`). |  
| **`JobRepository`** | **Lưu trữ** metadata của Job và hỗ trợ  
`JobExplorer`. |
```

Bằng cách này, bạn tách biệt hoàn toàn logic đối soát khỏi lượng lớn (Batch Job) với giao diện tương tác (REST API), tạo nên một kiến trúc mạnh mẽ, dễ bảo trì và khả năng mở rộng cao.

Chào bạn, để hoàn thiện thiết kế Service Đối soát, chúng ta sẽ xem xét các yếu tố về **Quản lý Cấu hình (Configuration)** và **Triển khai/Lập lịch (Deployment/Scheduling)**.

## ⌚ Quản lý Cấu hình (Configuration Management)

Trong môi trường Spring Boot, cấu hình phải linh hoạt để dễ dàng chuyển đổi giữa các môi trường (Dev, Test, Prod).

### 1. File Cấu hình (application.yml hoặc .properties)

Cần quản lý các tham số quan trọng sau:

Cấu hình Cơ sở dữ liệu (Database):

- Cấu hình Spring Batch Metadata:** Cần chỉ định nơi lưu trữ các bảng metadata của Job Repository (thường là cùng DB với dữ liệu nghiệp vụ, hoặc DB riêng nếu cần hiệu năng cao).
- Tham số Nghiệp vụ Đối soát:**
  - reconciliation.batch.chunk-size: Kích thước Chunk(ví dụ: 1000).
  - reconciliation.skip-limit: Giới hạn lỗi bỏ qua (faultTolerant) trước khi dừng Job.
  - reconciliation.match-tolerance: Ngưỡng khoan dung cho việc so sánh số tiền (ví dụ: 0.01USD) nếu áp dụng khớp mờ (fuzzy matching).

### 2. Sử dụng Profiles

Sử dụng Spring Profiles để dễ dàng chuyển đổi cấu hình:

- **application-dev.yml:** Dùng DB local, kích thước chunk nhỏ, bật log debug.
- **application-prod.yml:** Dùng DB production, kích thước chunk lớn, tắt log debug.

## Lập lịch và Triển khai (Scheduling & Deployment)

### 1. Lập lịch Kích hoạt Job (Scheduling)

Việc đối soát thường diễn ra vào ban đêm. Có hai phương pháp lập lịch chính:

- **Spring Scheduler (Internal):** Sử dụng @Scheduled trong Spring Boot.
  - **Ưu điểm:** Đơn giản, tích hợp sẵn.
  - **Nhược điểm:** Phức tạp khi triển khai trên nhiều instance (phải đảm bảo chỉ có một instance chạy Job để tránh trùng lặp - cần dùng cơ chế khóa DB như **ShedLock**).
- Java
- **Công cụ Bên ngoài (External):** Sử dụng **Quartz**, **Apache Airflow**, hoặc **Kubernetes CronJob**.
  - **Ưu điểm:** Khả năng quản lý và theo dõi tập trung tốt hơn, đảm bảo tính duy nhất khi chạy (Single instance guarantee).
  - **Khuyến nghị:** Đối với hệ thống quy mô lớn, nên dùng CronJob hoặc Airflow để gọi API /api/reconciliation/jobs/launch của Service.

### 2. Đóng gói và Triển khai (Packaging and Deployment)

Service cần được đóng gói để triển khai:

- **JAR (Executable JAR):** Cách đóng gói tiêu chuẩn của Spring Boot (mvn package). Dễ dàng chạy trên các máy ảo (VM) truyền thống.
- **Docker Container (Khuyến nghị):** Đóng gói Service và tất cả các phụ thuộc vào một Docker Image.
  - **Ưu điểm:** Đảm bảo môi trường chạy giống hệt nhau trên mọi môi trường (Dev/Test/Prod). Dễ dàng triển khai trên Kubernetes (K8s) hoặc Docker Swarm.
  - **Dockerfile:** Sẽ bao gồm các bước từ việc build ứng dụng Java sang việc tạo một Container Image chạy JAR đó.

Việc thiết lập các cấu hình và quy trình triển khai này giúp Service Đối soát của bạn sẵn sàng hoạt động trong môi trường sản xuất một cách an toàn và tự động.

Tuyệt vời. Sau khi đã có thiết kế kiến trúc, xử lý lỗi và triển khai, bước tiếp theo cực kỳ quan trọng đối với một hệ thống tài chính là **Tối ưu hóa Hiệu năng (Performance Optimization)** và **Chiến lược Kiểm thử (Testing Strategy)**.

Khi dữ liệu lên tới hàng triệu giao dịch mỗi ngày, bạn không thể xử lý tuần tự (sequential) mãi được.

## Tối ưu hóa Hiệu năng (Performance Optimization)

Để xử lý khối lượng dữ liệu lớn (Big Data) trong thời gian ngắn (SLA), chúng ta cần áp dụng các kỹ thuật nâng cao của Spring Batch.

## 1. Phân vùng Dữ liệu (Partitioning)

Thay vì một tiến trình (Single Thread) đọc và xử lý 1 triệu bản ghi từ đầu đến cuối, chúng ta chia dữ liệu thành các **phân nhánh (partitions)** và xử lý chúng **song song (parallel)** trên nhiều luồng.

- **Cơ chế:** Sử dụng Partitioner của Spring Batch để chia dữ liệu dựa trên ID hoặc Ngày.
  - **Ví dụ:** Thread 1 xử lý ID 1 - 10,000; Thread 2 xử lý ID 10,001 - 20,000...
- **Cấu hình:**
  - **Master Step:** Chịu trách nhiệm phân chia công việc.
  - **Slave Step:** Thực hiện công việc xử lý thực tế (Reader → Processor → Writer).
  - **TaskExecutor:** Quản lý Thread Pool (ví dụ: chạy 10 luồng cùng lúc).

## 2. Tối ưu hóa Cơ sở dữ liệu (Database Tuning)

Cổ chai (bottleneck) lớn nhất của đối soát thường nằm ở I/O cơ sở dữ liệu.

- **Batch Insert/Update:** Đảm bảo JDBC driver được cấu hình để gửi lệnh theo lô thay vì từng dòng lệnh đơn lẻ.
  - **Ví dụ (PostgreSQL):** reWriteBatchedInserts=true trong chuỗi kết nối.
- **Indexing (Đánh chỉ mục):** Bắt buộc phải đánh index cho các cột dùng để so sánh (transaction\_id, transaction\_date, amount) trong bảng TXN\_SOURCE\_A và TXN\_SOURCE\_B. Nếu không, bước Processor sẽ cực chậm khi tìm kiếm dữ liệu khớp.
- **Fetch Size:** Cấu hình JdbcCursorItemReader với fetchSize phù hợp (ví dụ: 1000) để giảm số lần gọi network tới DB.

## Chiến lược Kiểm thử (Testing Strategy)

Hệ thống đối soát liên quan trực tiếp đến tiền, nên lỗi logic là không thể chấp nhận được.

### 1. Unit Testing (Kiểm thử Đơn vị)

Tập trung kiểm tra logic nghiệp vụ trong **ReconciliationProcessor**.

- **Mocking:** Tạo dữ liệu giả lập cho TxnSourceA và TxnSourceB.
- **Scenarios (Kịch bản):**
  - Test case 1: Hai giao dịch khớp hoàn toàn → Kết quả MATCHED.
  - Test case 2: Khớp ID nhưng lệch số tiền → Kết quả MISMATCHED+ ghi nhận số tiền lệch.
  - Test case 3: Tìm không thấy trong Source B → Kết quả A\_ONLY.

Auto (Java) ▾



```

    @Test
    void testMatchingTransaction() {
        // Arrange
        TxnSourceA txnA = new TxnSourceA("ID1", 100.00);
        TxnSourceB txnB = new TxnSourceB("ID1", 100.00);
        // Act
        ReconResult result = processor.process(txnA, txnB);
        // Assert
        assertEquals(MatchStatus.MATCHED, result.getStatus());
    }

```

## 2. Integration Testing (Kiểm thử Tích hợp)

Kiểm tra toàn bộ luồng Batch (Reader → Processor → Writer) tương tác với cơ sở dữ liệu thực tế như thế nào.

- **Công cụ:** Sử dụng **@SpringBatchTest** và **Testcontainers**.
- **Testcontainers:** Tự động khởi chạy một Docker Container chứa PostgreSQL thật trong quá trình chạy test. Điều này đảm bảo các câu lệnh SQL và logic giao dịch hoạt động đúng như môi trường thật (điều mà H2 database in-memory đôi khi không phản ánh đúng).

## 3. Performance Testing (Kiểm thử Tải)

Trước khi đưa lên Production, bạn cần biết giới hạn của hệ thống.

- **Tạo dữ liệu giả (Data Seeding):** Script tạo 1 triệu dòng dữ liệu vào 2 bảng nguồn.
- **Đo lường:** Chạy Job và đo thời gian hoàn thành. Quan sát CPU/RAM usage.
- **Chaos Engineering:** Thủ tắt ngẫu nhiên Database hoặc kill Process khi Job đang chạy để kiểm tra xem tính năng **Restartability** (Tái khởi động) có hoạt động đúng không (không bị mất dữ liệu hay trùng lặp).

## ■ Giám sát & Cảnh báo (Monitoring & Alerting)

Cuối cùng, khi hệ thống đã chạy, bạn cần "mắt thần" để theo dõi nó.

- **Spring Boot Actuator:** Cung cấp các metrics về sức khỏe hệ thống.
- **Micrometer + Prometheus + Grafana:**
  - Đẩy metrics từ Spring Batch (số bản ghi đọc/giây, thời gian xử lý trung bình, số lỗi).
  - Vẽ biểu đồ dashboard để đội vận hành theo dõi tiến độ Job theo thời gian thực.
- **Alerting:** Gửi cảnh báo qua Slack/Email/SMS ngay lập tức nếu:
  - Job kết thúc với trạng thái FAILED.
  - Tỷ lệ giao dịch lỗi (Discrepancy Rate) vượt quá ngưỡng an toàn (ví dụ: > 5% giao dịch bị lỗi).

Tới đây, bạn đã có một bản thiết kế toàn diện từ **Kiến trúc → Code logic → API → Hiệu năng** → **Kiểm thử → Giám sát**.

Chào bạn, đối soát thanh toán thẻ (Card Reconciliation) là một trong những nghiệp vụ phức tạp và quan trọng nhất trong ngân hàng, vì nó liên quan đến nhiều bên tham gia và các cấu trúc phí rắc rối.

Dưới đây là giải thích chi tiết về cơ chế này theo hai góc độ: **Ngân hàng Phát hành (Issuer)** và **Ngân hàng Thanh toán (Acquirer)**.

## 1. Mô hình tổng quan: Ai tham gia?

Trước khi đi vào đối soát, bạn cần hình dung dòng chảy của một giao dịch thẻ (Transaction Flow).

Trong mô hình này có 4 bên chính (4-Party Model):

- Chủ thẻ (Cardholder):** Người quét thẻ.
- Merchant (Đơn vị chấp nhận thẻ):** Cửa hàng, web TMĐT.
- Acquirer (Ngân hàng thanh toán):** Ngân hàng cung cấp máy POS/cổng thanh toán cho Merchant.
- Issuer (Ngân hàng phát hành):** Ngân hàng cấp thẻ cho Chủ thẻ.
- Card Scheme (Tổ chức thẻ):** Visa, Mastercard, JCB, Napas (đóng vai trò trung gian chuyển mạch và bù trừ).

**Đối soát thẻ chính là việc đảm bảo tiền đi từ Issuer → Scheme → Acquirer → Merchant là chính xác đến từng xu.**

## 2. Chi tiết Đối soát theo Vai trò Ngân hàng

### A. Đối với Ngân hàng Phát hành (Issuing Reconciliation)

*Vai trò: Ngân hàng của người mua. Bạn trừ tiền tài khoản khách hàng và phải trả tiền đó cho Visa/Mastercard.*

**Quy trình đối soát 3 bên (3-Way Reconciliation):**

Hệ thống phải khớp dữ liệu từ 3 nguồn:

- Hệ thống Switch (Chuyển mạch thẻ):** Ghi nhận hành động quét thẻ (Authorization logs).
- Hệ thống Core Banking:** Ghi nhận việc trừ tiền/phong tỏa tiền trong tài khoản khách hàng.
- Báo cáo từ Tổ chức thẻ (Scheme Incoming Files):** File quyết toán (Settlement File) mà Visa/Mastercard gửi về (ví dụ: file TC33, IPM).

### Logic đối soát:

- **Bước 1: So sánh Switch vs Core Banking.** Đảm bảo mọi giao dịch quẹt thẻ thành công đều đã trừ tiền khách hàng. (Phát hiện lỗi hệ thống nội bộ).
- **Bước 2: So sánh Core Banking vs Scheme File.**
  - Ngân hàng kiểm tra xem số tiền Visa/Mastercard đòi (trong file quyết toán) có khớp với số tiền đã trừ của khách hàng không.
  - **Vấn đề phức tạp:**
    - **Phí chuyển đổi ngoại tệ:** Khách quẹt 100 USD, Core trừ 2.500.000 VNĐ. File Visa báo nợ 101 USD (bao gồm phí). Hệ thống phải tính toán lại tỷ giá để xem khớp không.
    - **Giao dịch Offline:** Các giao dịch trên máy bay hoặc vùng sâu vùng xa (xử lý trễ) gửi về sau vài ngày.

## B. Đối với Ngân hàng Thanh toán (Acquiring Reconciliation)

*Vai trò: Ngân hàng của người bán. Nhận tiền từ Visa/Mastercard và phải trả cho Merchant.*

### Quy trình:

Ngân hàng Acquiring phải đối soát giữa:

1. **Dữ liệu từ POS/Cổng thanh toán:** Danh sách các giao dịch Merchant đã thực hiện (đã kết toán - Batch Settlement).
2. **Báo cáo từ Tổ chức thẻ (Scheme Outgoing Files):** Xác nhận số tiền Visa/Mastercard sẽ chuyển về cho Ngân hàng Acquiring.
3. **Hệ thống tính phí (Merchant Management System):** Tính toán phí MDR (Merchant Discount Rate) mà ngân hàng thu của Merchant.

### Logic đối soát:

- **Mục tiêu:** Đảm bảo Visa trả đủ tiền để Ngân hàng trả cho Merchant.
- **Vấn đề phức tạp:**
  - **Phí Interchange (Interchange Fee):** Visa sẽ không trả đủ 100% số tiền giao dịch. Họ sẽ trừ phí Interchange (trả cho Issuer) và phí Scheme.
  - **Ví dụ:** Khách quẹt 1 triệu. Visa chỉ chuyển về cho Acquirer 980k. Acquirer trả cho Merchant 985k (chịu lỗ 5k nếu tính phí sai) hoặc trả 970k (lời 10k). Hệ thống đối soát phải tách bóc (breakdown) các loại phí này cực kỳ chi tiết.

## 3. Các loại Dữ liệu và File Đối soát Đặc thù

Trong đối soát thẻ, dữ liệu không chỉ là file Excel đơn giản mà là các chuẩn tin điện quốc tế:

- **Dữ liệu Giao dịch (Transaction Logs):** Chuẩn ISO 8583.
- **File từ Tổ chức thẻ (Scheme Files):**
  - **Visa:** File VSS (Visa Settlement Service) - chứa thông tin bù trừ (Clearing) và quyết toán (Settlement).

- **Mastercard:** File IPM (Integrated Product Message).
- **Dữ liệu ATM:**
  - File nhật ký điện tử (Electronic Journal - EJ) từ máy ATM.
  - File đếm tiền vật lý (Physical Cash Count) từ đội tiếp quỹ.

## 4. Các Trường hợp Chênh lệch (Exceptions) Thường gặp

Hệ thống đối soát thẻ phải xử lý tự động các tình huống "đau đầu" sau:

1. **Chargeback (Tra soát/Hoàn trả):** Khách hàng khiếu nại giao dịch giả mạo. Visa sẽ tự động rút tiền từ Acquirer trả lại cho Issuer. Hệ thống đối soát phải nhận diện mã giao dịch này là "Reversal" chứ không phải giao dịch mới.
2. **Lệch phí (Fee Mismatch):** Phí ước tính trên Core Banking khác với phí thực tế Visa thu (do tỷ giá thay đổi giữa thời điểm quẹt và thời điểm quyết toán - thường lệch nhau 1-2 ngày).
3. **Giao dịch treo (Pending):** Khách quẹt thẻ, tiền bị giữ (Hold) nhưng Merchant không gửi lệnh kết toán (Settlement) trong ngày. Tiền treo trên Core nhưng không có trong báo cáo Visa.
4. **ATM thừa/thiếu quỹ:**
  - Khách rút 500k, máy nhả tiền nhưng không ghi nợ tài khoản (Lỗi Core).
  - Khách rút 500k, tài khoản bị trừ nhưng máy không nhả tiền (Lỗi phần cứng ATM). Đối soát phải khớp giữa Log máy ATM, Camera, và Core Banking để hoàn tiền (Refund).

Hệ thống đối soát thẻ (Card Recon) về bản chất là một **cỗ máy tính phí và so khớp khổng lồ**. Nó không chỉ so sánh số tổng (Total Amount) mà còn phải "mổ xé" từng giao dịch để kiểm tra:

- Phí Interchange có đúng không?
- Phí xử lý giao dịch quốc tế có đúng không?
- Tỷ giá áp dụng ngày hôm đó có khớp không?

Nếu không có hệ thống này, ngân hàng sẽ bị "chảy máu" dòng tiền do chênh lệch phí và rủi ro gian lận rất lớn.

Chào bạn, đây là một chủ đề rất chuyên sâu. Nếu như đối soát thẻ (Card Recon) xử lý hàng triệu giao dịch nhỏ lẻ (high volume, low value), thì **Đối soát Hệ thống Thanh toán (Payment Reconciliation)** lại xử lý các giao dịch có **giá trị rất lớn** và độ rủi ro cực cao (high value, critical impact).

Chỉ một lỗi sai sót trong hệ thống này (ví dụ: chuyển nhầm 100 tỷ VNĐ) có thể gây hậu quả nghiêm trọng về thanh khoản và uy tín.

Dưới đây là chi tiết cơ chế đối soát cho hai hệ thống xương sống của ngân hàng: **RTGS/CITAD** (Trong nước) và **SWIFT** (Quốc tế).

## 1. Đối soát RTGS/CITAD (Hệ thống Thanh toán Điện tử Liên ngân hàng)

**CITAD** (viết tắt của Trung tâm Tin học và Thông tin Ngân hàng) thường được dùng để chỉ hệ thống **IBPS** (Inter-bank Payment System). Đây là nơi các ngân hàng chuyển tiền cho nhau thông qua Ngân hàng Nhà nước (NHNN).

### A. Luồng giao dịch (Transaction Flow)

Để hiểu cách đối soát, ta cần nhìn luồng đi của một lệnh chuyển tiền:

- Core Banking:** Giao dịch viên tạo lệnh chuyển tiền đi → Core ghi nợ tài khoản khách hàng -> Trạng thái **DEBITED**.
- Payment Gateway (Cổng thanh toán nội bộ):** Hệ thống trung gian lấy lệnh từ Core, đóng gói thành điện tử chuẩn CITAD → Gửi đi NHNN → Trạng thái **SENT**.
- Hệ thống NHNN (SBV System):** Nhận điện tử, kiểm tra hạn mức thanh toán của ngân hàng -> Ghi nợ tài khoản ngân hàng gửi, ghi có ngân hàng nhận → Trả về trạng thái **SETTLED**.

### B. Cơ chế Đối soát 3 Chiều (3-Way Reconciliation)

Khác với thẻ, đối soát CITAD thường diễn ra **trong ngày** (Intra-day) và **cuối ngày** (End-of-day) giữa 3 nguồn dữ liệu:

Nguồn Dữ liệu	Dữ liệu chính	Vai trò
<b>1. Core Banking</b>	Số tiền, Tài khoản KH, Mã giao dịch (Ref ID).	Xác nhận tiền đã ra khỏi tài khoản khách hàng.
<b>2. Payment Hub (Gateway)</b>	Trạng thái gửi tin, ACK/NAK từ NHNN.	Xác nhận lệnh đã được gửi đi thành công mặt kỹ thuật.
<b>3. Báo cáo NHNN (CITAD Report)</b>	Bảng kê kết quả thanh toán (Cuối ngày).	<b>Nguồn sự thật (Single Source of Truth).</b> Xác nhận tiền thực sự đã được NHNN xử lý.

### C. Các tình huống lệch (Exceptions)

- Tiền đi nhưng không đến (Pending/Timeout):** Core đã trừ tiền, Gateway đã gửi, nhưng

NHNN chưa phản hồi (do nghẽn mạng).

- **Xử lý:** Hệ thống đối soát phải đánh dấu **PENDING**. Cuối ngày nếu không thấy trong báo cáo NHNN → Phải hoàn tiền (Reversal) cho khách.
- **NHNN từ chối (Rejected) nhưng Core đã trừ:** Lệnh sai định dạng hoặc Ngân hàng hết hạn mức thanh khoản, NHNN trả lại (Reject). Nhưng Core chưa cập nhật trạng thái này.
  - **Xử lý:** Đối soát phát hiện trạng thái lệch → Tự động hạch toán hoàn tiền vào tài khoản khách hàng.

## 2. Đối soát SWIFT (Thanh toán Quốc tế)

Đối soát SWIFT phức tạp hơn nhiều vì nó không phải là "chuyển tiền trực tiếp" mà là "gửi tin nhắn cam kết trả tiền" qua nhiều ngân hàng trung gian (Correspondent Banks).

### A. Luồng giao dịch và Tin nhắn SWIFT

- **MT103:** Điện chuyển tiền cho khách hàng.
- **MT202:** Điện chuyển tiền giữa các ngân hàng (Cover payment).
- **ACK/NAK:** SWIFT gửi lại xác nhận đã nhận tin (ACK) hoặc từ chối (NAK).

### B. Quy trình Đối soát SWIFT

Quá trình này chia làm 2 tầng:

#### Tầng 1: Đối soát Tin nhắn (Message Reconciliation)

Mục tiêu: Đảm bảo tin nhắn đã gửi đi thành công.

- **So sánh:** Dữ liệu từ **Hệ thống thanh toán quốc tế (T24/Trade Finance)** vs. **SWIFT Alliance Access (SAA - Cổng kết nối SWIFT)**.
- **Kiểm tra:** Mọi điện MT103 gửi đi phải nhận được **ACK** từ mạng lưới SWIFT. Nếu nhận **NAK** (Negative Acknowledgement), nghĩa là điện bị sai định dạng và chưa được gửi đi → Giao dịch thất bại.

#### Tầng 2: Đối soát Tài khoản Nostro (Nostro Reconciliation)

Mục tiêu: Đảm bảo tiền thực sự đã di chuyển. Đây là phần quan trọng nhất.

- Ngân hàng A gửi lệnh MT103 cho Ngân hàng B qua Ngân hàng trung gian C (JPMorgan).
- Ngân hàng A hạch toán giảm số dư tại sổ cái (Ledger) của tài khoản Nostro mở tại JPMorgan.
- **Ngày T+1:** JPMorgan gửi về một điện **MT950 (Sao kê tài khoản)** báo cáo các giao dịch đã phát sinh.

#### Cơ chế khống:

1. Hệ thống Smart Recon sẽ đọc file **MT950** (từ JPMorgan).
2. So sánh từng dòng trong MT950 với sổ cái kế toán nội bộ của Ngân hàng.

- 3. Khớp:** Số tiền, Loại tiền (Currency), Ngày hiệu lực (Value Date), Số tham chiếu (Reference Number).

### C. Vấn đề "Đau đầu" nhất: Float & Charges

Trong SWIFT, số tiền gửi đi và số tiền bên kia nhận được thường không khớp nhau do các ngân hàng trung gian "cắt phí" trên đường đi (nếu chọn phí SHA/BEN).

- **Ví dụ:** Gửi 10,000 USD.
- Trung gian 1 trừ 20 USD.
- Trung gian 2 trừ 15 USD.
- Trên báo cáo MT950 có thể chỉ hiện ghi nợ 10,000 USD (còn phí thu riêng) hoặc ghi nợ 10,035 USD.
- **Logic đối soát:** Phải cực kỳ linh hoạt để nhận diện đâu là số tiền gốc (Principal) và đâu là phí (Charges) dựa trên mã code trong điện MT950 (ví dụ code **CHGS** ).

### Tóm tắt sự khác biệt

Đặc điểm	Đối soát CITAD/RTGS	Đối soát SWIFT
Phạm vi	Trong nước (Domestic).	Quốc tế (Cross-border).
Thời gian	Real-time hoặc cuối ngày (T).	Thường là T+1 (khi nhận được sao kê MT950).
Đối tượng	NHNN (Ngân hàng Nhà nước).	Ngân hàng đại lý (Correspondent Banks).
Độ phức tạp	Trung bình (Dữ liệu cấu trúc rõ ràng).	Rất cao (Liên quan đến tỷ giá, phí trung gian, múi giờ).
Rủi ro	Thanh khoản (Liquidity).	Tỷ giá (FX Risk) và rửa tiền (AML).

Hệ thống đối soát thanh toán (Payment Recon) đóng vai trò như "chốt chặn cuối cùng" để đảm bảo tiền của ngân hàng không bị thất thoát ra ngoài biên giới hoặc bị kẹt trong hệ thống liên ngân hàng.

**Payment System Reconciliation Service** (đặc biệt cho SWIFT và CITAD/RTGS) khác biệt hoàn toàn với đối soát Loyalty hay Tiền mặt thông thường.

Đặc thù của nó là: **Đa định dạng** (MT files, XML MX, Excel, DB rows), **Đa tiền tệ**, và **Logic phức tạp** (người chuyển trả, người nhận trả, chia đôi).

Dưới đây là bản thiết kế chi tiết kiến trúc và kỹ thuật cho hệ thống này trên nền tảng **Java Spring Boot**.

## I. Kiến trúc Tổng thể (High-Level Architecture)

Hệ thống nên được chia thành 4 lớp xử lý chính theo mô hình **ETL (Extract - Transform - Load)** kết hợp **Event-Driven**.

### 1. Ingestion Layer (Lớp Tiếp nhận)

- **Core Banking Adapter:** Lấy dữ liệu sổ cái (General Ledger) qua DB Link hoặc Kafka.
- **SWIFT Adapter:** Lắng nghe thư mục an toàn (Secure FTP) để đọc file MT940/MT950 (Statement) và MT103/202 (Message).
- **CITAD Adapter:** Kết nối API hoặc đọc file báo cáo cuối ngày từ hệ thống IBPS.

### 2. Normalization Layer (Lớp Chuẩn hóa - Quan trọng nhất)

Đây là "bộ não" chuyển đổi các định dạng điện tử của ngân hàng về một chuẩn chung.

- **Input:** MT950 text file, CITAD Excel report, JSON từ Core.
- **Process:** Parsing (phân tích cú pháp).
- **Output:** **StandardTransaction** (Object chuẩn nội bộ).

### 3. Matching Engine (Lớp Đối chiếu)

Thực hiện so khớp dựa trên Rule-set.

### 4. Exception & Settlement (Lớp Xử lý Lỗi & Hạch toán)

Giao diện cho con người xử lý sai lệch và module tự động hạch toán (Auto-posting) nếu cần.

## II. Chi tiết Kỹ thuật & Công nghệ

### 1. Data Model: Chuẩn hóa dữ liệu (Standard Transaction)

Dù nguồn là SWIFT hay CITAD, mọi giao dịch đều phải được convert về entity này để đối soát.

Auto (TypeScript) ▾



```
@Entity
@Table(name = "std_transaction")
public class StandardTransaction {
    @Id
    private String id; // Internal UUID
    private String sourceSystem; // "CORE", "SWIFT", "CITAD"
    private String originalRawData; // Lưu lại chuỗi gốc MT950 dòng 61
    // để tra cứu
    // Các trường khóa để đối soát
    private String transactionRef; // Ref number (VD: FT230910001)
    private String relatedRef; // Dùng cho trường hợp SWIFT (Field
```

```

21)
    private BigDecimal amount;
    private String currency;
    private LocalDate valueDate;      // Ngày hiệu lực (Rất quan trọng
    trong SWIFT)
    private String drCrType;         // DEBIT hoặc CREDIT
    private String accountNo;        // Số tài khoản Nostro hoặc TK trung
    gian

    private String reconciliationStatus; // UNMATCHED, MATCHED, SUGGESTED
}

```

## 2. Xử lý SWIFT MT950/MT940 (Parser)

Điện MT950 rất khó parse thủ công vì nó là chuỗi text không cấu trúc rõ ràng.

- **Thư viện khuyến nghị: Prowide Core** (Open source Java library for SWIFT). Thư viện này giúp bạn tách bóc các trường (Tag) trong điện SWIFT dễ dàng.

Ví dụ xử lý Tag :61: (Statement Line) trong MT950:

Tag :61: chứa ngày, nợ/có, số tiền và mã tham chiếu trong một dòng duy nhất.

Dòng mẫu: :61:2311291129D1000,00NTRFNONREF//FT12345678

Java

Auto (Java) ▾



```

// Sử dụng Prowide Core để parse
public StandardTransaction parseSwiftMT950Line(Field61 field61) {
    StandardTransaction txn = new StandardTransaction();
    txn.setValueDate(field61.getValueDate()); // Lấy ngày 29/11/23
    txn.setDrCrType(field61.getDCMark());     // Lấy 'D' (Debit)
    txn.setAmount(field61.getAmount());        // Lấy 1000.00
    txn.setTransactionRef(field61.getReference()); // Lấy FT12345678
    txn.setSourceSystem("SWIFT");
    return txn;
}

```

## 3. Thuật toán Matching (Matching Engine)

Đối với thanh toán quốc tế, chúng ta không thể chỉ so khớp "Bằng nhau tuyệt đối" (Exact Match).

### Chiến lược Matching (Java Spring Batch/Service):

- **Pass 1: Exact Match (Khớp 100%)**
  - Điều kiện: `Core.Ref = Swift.Ref AND Core.Ccy = Swift.Ccy AND Core.Amount = Swift.Amount`.
  - Độ chính xác: Cao nhất. Tự động duyệt.
- **Pass 2: Tolerance Match (Khớp lệch phí)**
  - Trong SWIFT, số tiền nhận được có thể bị trừ phí (ví dụ \$10,000 còn \$9,980).
  - Điều kiện: `Core.Ref = Swift.Ref AND Abs(Core.Amount - Swift.Amount) ≤ FEE_THRESHOLD` (Ví dụ: \$50).
  - Hành động: Đánh dấu là `MATCHED_WITH_DIFF`. Hệ thống tự động hạch toán phần chênh lệch (\$20) vào tài khoản "Chi phí thanh toán quốc tế".
- **Pass 3: Fuzzy Reference Match (Khớp sai mã tham chiếu)**
  - Đôi khi mã Ref bị cắt bớt hoặc thêm tiền tố/hậu tố khi đi qua các ngân hàng trung gian.
  - Điều kiện: `Core.Amount = Swift.Amount AND Core.ValueDate = Swift.ValueDate AND LevenshteinDistance(Core.Ref, Swift.Ref) < 3`.
  - Hành động: Đánh dấu `SUGGESTED` (Gợi ý). Cần con người xác nhận (Manual Confirm).

## 4. Xử lý Vấn đề "Ngày hiệu lực" (Value Date)

Vấn đề: Core Banking hạch toán ngày T (30/11), nhưng tiền thực tế về tài khoản Nostro vào ngày T+1 (01/12).

Giải pháp:

- Khi query dữ liệu để đối soát, **không query chính xác theo ngày**.
- Query theo **Cửa sổ trượt (Sliding Window)**: Lấy dữ liệu Core ngày T và đối chiếu với dữ liệu SWIFT từ `T-1` đến `T+3`.

## III. Thiết kế Cơ sở dữ liệu (Database Schema)

Cần tối ưu cho việc truy vấn lịch sử và kiểm toán (Audit).

Bảng	Chức năng	Index Quan trọng
<b>PAYMENT_RAW_MSG</b>	Lưu toàn bộ nội dung file gốc (Clob/Text) để làm bằng chứng pháp lý.	<code>msg_id</code> , <code>received_date</code>
<b>RECON_LEDGER</b>	Dữ liệu từ Core Banking (Sổ cái).	<code>ref_no</code> , <code>amount</code> , <code>value_date</code>

<b>RECON_STATEMENT</b>	Dữ liệu từ SWIFT/CITAD (Sổ phụ).	<code>ref_no, amount, value_date</code>
<b>MATCH_RESULT</b>	Kết nối ID giữa Ledger và Statement.	<code>ledger_id, stmt_id, match_type</code>
<b>EXCEPTION_QUEUE</b>	Danh sách các giao dịch không khớp cần xử lý.	<code>status</code> (OPEN, INVESTIGATING, CLOSED)

## IV. Quy trình Xử lý Ngoại lệ (Exception Handling Workflow)

Đây là phần UI/UX dành cho nhân viên vận hành (Back-office users).

1. **Màn hình Dashboard:** Hiển thị số lượng giao dịch Unmatched theo tuổi nợ (Aging): 1 ngày, >3 ngày, >7 ngày (Rủi ro cao).
2. **Công cụ Điều tra:**
  - Cho phép search giao dịch bên Core và bên SWIFT để khớp thủ công (Manual Match).
  - Tính năng "**Force Match**": Người dùng chọn 1 giao dịch Core và 1 giao dịch SWIFT, hệ thống sẽ khớp cưỡng bức và yêu cầu nhập lý do chênh lệch.
3. **Tự động Hạch toán (Auto-Adjustment):**
  - Nếu chênh lệch được xác định là phí ngân hàng nước ngoài thu → Gọi API Core Banking để hạch toán bút toán chi phí (Debit Expense / Credit Nostro).

## V. Công nghệ đề xuất (Tech Stack)

- **Framework:** Java 17/21, Spring Boot 3.x.
- **Batch Processing:** Spring Batch (Xử lý file MT950/CITAD cuối ngày).
- **Integration:** Apache Camel (Cực kỳ mạnh mẽ để định tuyến file, kết nối FTP, MQ, và parse định dạng).
- **Parsing:** Prowide Core (cho SWIFT), Apache POI (cho Excel CITAD).
- **Database:** Oracle (thường dùng trong bank) hoặc PostgreSQL.
- **Message Queue:** Kafka/RabbitMQ (để nhận real-time status từ Payment Hub).

Bạn có muốn tôi đi sâu vào chi tiết code mẫu cấu hình **Apache Camel** để tự động đọc và parse file SWIFT MT940/950 không?

Troyet von Duy là hướng dẫn cách khai triển cách sao chép Apache Camel kết hợp với thư viện

**Prowide Core** (thư viện chuẩn công nghiệp để xử lý SWIFT) để tự động hóa việc đọc và phân tích file điện MT950 (**Statement**).

## 🛠 Cài đặt Môi trường

Trước hết, bạn cần thêm các dependency sau vào file `pom.xml` của dự án Spring Boot:

XML

Auto (HTML, XML) ▾



```
<dependencies>
    <dependency>
        <groupId>org.apache.camel.springboot</groupId>
        <artifactId>camel-spring-boot-starter</artifactId>
        <version>4.0.0</version> </dependency>

    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-file</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-ftp</artifactId>
    </dependency>

    <dependency>
        <groupId>com.prowidesoftware</groupId>
        <artifactId>pw-swift-core</artifactId>
        <version>SRU2023-10.1.5</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

Như đã thảo luận, chúng ta cần một object chuẩn để hứng dữ liệu sau khi parse.

Java

Auto (TypeScript) ▾



```
import lombok.Data;
import java.math.BigDecimal;
import java.time.LocalDate;

@Data
public class StandardTransaction {
    private String sourceSystem = "SWIFT";
    private String swiftMessageType; // VD: "950"
    private String accountNo; // Tag :25: (Số tài khoản Nostro)
    private String transactionRef; // Tag :61: Reference
    private String relatedRef; // Tag :61: Supplementary Details
    private BigDecimal amount; // Tag :61: Amount
    private String currency; // Từ Tag :25: hoặc cấu hình
    private String drCrType; // "D" hoặc "C"
    private LocalDate valueDate; // Tag :61: Date
    private String rawData; // Nội dung gốc dòng 61 (để trace
lỗi)
    private String entryDetail; // Tag :86: Thông tin bổ sung (Rất
quan trọng)
}
```

## 2. Thiết kế Camel Route (Luồng xử lý)

Class này định nghĩa luồng đi của dữ liệu: **Đọc File → Parse → Lưu DB.**

Auto (Scala) ▾



```
import org.apache.camel.builder.RouteBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
```

```
@Component
public class SwiftIngestionRoute extends RouteBuilder {

    @Autowired
    private SwiftMt950Processor swiftMt950Processor;

    @Autowired
    private TransactionService transactionService;

    @Override
    public void configure() throws Exception {
        // 1. Đọc file từ thư mục input (hoặc SFTP server)
        from("file:data/swift/inbox?
delay=5000&move=.done&readLock=changed")
            .routeId("SwiftReaderRoute")
            .log("Đang xử lý file SWIFT: ${header.CamelFileName}")

        // 2. Xử lý lỗi (Try-Catch)
        .doTry()
            // 3. Gọi Processor để parse nội dung file
            .process(swiftMt950Processor)

        // 4. Kết quả sau khi parse (List<StandardTransaction>
được gửi đi lưu
            .bean(transactionService, "saveBatchTransactions")
            .log("Đã lưu thành công các giao dịch từ file:
${header.CamelFileName}")
            .doCatch(Exception.class)
                .log("Lỗi xử lý file SWIFT: ${exception.message}")
                .to("file:data/swift/error") // Di chuyển file lỗi ra
chỗ khác
            .end();
    }
}
```

### 3. Implement Processor (Logic Parse SWIFT Cốt lõi)

Đây là nơi "ma thuật" xảy ra. Chúng ta sử dụng **Provide Core** để biến chuỗi text vô nghĩa thành dữ liệu có cấu trúc.

Auto (Java) ▾



```
import com.prowidesoftware.swift.model.field.Field61;
import com.prowidesoftware.swift.model.field.Field86;
import com.prowidesoftware.swift.model.mt.mt9xx.MT950;
import com.prowidesoftware.swift.model.mt.AbstractMT;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

@Component
public class SwiftMt950Processor implements Processor {

    @Override
    public void process(Exchange exchange) throws Exception {
        // 1. Lấy nội dung file dưới dạng String
        String body = exchange.getIn().getBody(String.class);

        List<StandardTransaction> parsedTransactions = new ArrayList<>();

        // 2. Tự động nhận diện và parse file SWIFT
        // Provide hỗ trợ parse cả file có nhiều message (RJE format)
        // nhưng ở đây ta giả sử 1 message/file
        AbstractMT abstractMT = AbstractMT.parse(body);

        // 3. Kiểm tra xem có phải MT950 không
        if (abstractMT.isMT950()) {
            MT950 mt950 = (MT950) abstractMT;

            // Lấy thông tin Header chung
            String accountNo = "";
            if (mt950.getField25() != null) {
                accountNo = mt950.getField25().getValue(); // Tag :25:
                Số tài khoản
            }

            // 4. Lặp qua danh sách các giao dịch (Loop 1 trong cấu trúc
            // SWIFT)
            // MT950 chứa danh sách Field 61 (Statement Line)
            for (Field61 field61 : mt950.getField61()) {
                StandardTransaction txn = new StandardTransaction();

                txn.setSwiftMessageType("950");
                txn.setAccountNo(accountNo);

                // Parse các trường quan trọng từ dòng :61:
                // Cấu trúc :61:2311301130D1000,00NTRFNONREF//REF123456
            }
        }
    }
}
```

```

txn.setValueDate(com.prowidesoftware.swift.utils.Lib.getCalendar(field61.
getValueDate()).toInstant().atZone(java.time.ZoneId.systemDefault()).toLo
calDate());
        txn.setDrCrType(field61.getDCMark()); // "D" hoặc "C"
hoặc "RC" (Reversal)
        txn.setAmount(field61.getAmount()); // Tự động convert
"1000,00" thành BigDecimal
        txn.setTransactionRef(field61.getReference()); //
"NONREF" hoặc Ref thực tế
        txn.setRelatedRef(field61.getSupplementaryDetails()); //
"REF123456" - Thường mã Ref thật nằm ở đây
        txn.setRawData(field61.getValue());

        // 5. Xử lý Tag :86: (Information to Account Owner) -
Nếu có
        // Tag 86 thường đi ngay sau Tag 61 để mô tả chi tiết
giao dịch đó
        // Provide không tự map 1-1, ta cần logic để tìm Tag 86
tương ứng (thường là tag tiếp theo trong sequence)
        // *Lưu ý: Logic tìm tag 86 trong thực tế phức tạp hơn,
cần duyệt tuần tự các tag.*

        parsedTransactions.add(txn);
    }
} else {
    throw new IllegalArgumentException("File không phải định
dạng MT950 hợp lệ");
}

// 6. Đưa danh sách đã parse vào Body để bước tiếp theo xử lý
exchange.getIn().setBody(parsedTransactions);
}
}

```

## 4. Xử lý Nâng cao: Mapping Field :86:

Trong SWIFT MT950, dòng **:61:** chứa thông tin tài chính (tiền, ngày), nhưng dòng **:86:** đi kèm ngay sau nó mới chứa **nội dung chuyển tiền** (VD: "Invoice Payment 123 from Company A"). Đối soát viên rất cần trường này.

Để map chính xác **:61:** với **:86:** tương ứng, bạn nên dùng cách duyệt tuần tự thay vì **mt950.getField61()**:

Auto (C#) ▾



```

// Logic cài tiến trong Processor để map Field 61 và 86
List<Tag> tags = mt950.getSwiftMessage().getBlock4().getTags();
StandardTransaction currentTxn = null;

for (Tag tag : tags) {
    if (tag.getName().equals("61")) {
        // Nếu gặp thẻ 61 mới, lưu thẻ cũ (nếu có) và tạo giao dịch mới
        if (currentTxn != null) {
            parsedTransactions.add(currentTxn);
        }
        currentTxn = new StandardTransaction();
        Field61 f61 = (Field61) tag.asField();
        // ... (Map dữ liệu f61 vào currentTxn như trên) ...
    }
    else if (tag.getName().equals("86") && currentTxn != null) {
        // Nếu gặp thẻ 86 và đang có giao dịch mở, gán nó vào giao dịch
        // đó
        Field86 f86 = (Field86) tag.asField();
        currentTxn.setEntryDetail(f86.getValue());
    }
}
// Add giao dịch cuối cùng
if (currentTxn != null) {
    parsedTransactions.add(currentTxn);
}

```

## 5. Tại sao thiết kế này hiệu quả?

- Event-Driven:** Apache Camel tự động phát hiện khi có file mới rơi vào thư mục hoặc SFTP. Không cần chạy Cron Job thủ công để check file.
- Robust Parsing:** Sử dụng Prowide Core giúp tránh được hàng trăm lỗi parsing thủ công (ví dụ: xử lý dấu phẩy thay vì dấu chấm thập phân trong file SWIFT **1000,00**, xử lý năm nhuận, định dạng ngày **YYMMDD** ).
- Tách biệt (Decoupling):** Luồng đọc file tách biệt với logic parse và logic lưu DB. Bạn có thể dễ dàng thay đổi nguồn đọc từ File sang Kafka mà không sửa logic parse.

Bạn có muốn tôi giải thích thêm về cách cấu hình kết nối **SFTP an toàn (có SSH Key)** trong Camel để lấy file từ ngân hàng nước ngoài không?

Tuyệt vời. Việc kết nối SFTP với xác thực bằng **SSH Key** (thay vì mật khẩu thông thường) là tiêu chuẩn bắt buộc trong ngành ngân hàng để đảm bảo bảo mật khi truyền tải các file nhạy cảm như MT940/MT950.

Dưới đây là hướng dẫn chi tiết cách cấu hình **Apache Camel** để kết nối an toàn tới máy chủ của đối tác (ngân hàng nước ngoài).

## 1. Cơ chế hoạt động (Concept)

Trong mô hình này:

- Bank A (Bạn):** Tạo ra một cặp khóa (Key Pair).
  - Private Key:** Bạn giữ bí mật tuyệt đối trên server của mình.
  - Public Key:** Bạn gửi cho ngân hàng đối tác (Bank B) để họ cài đặt lên server SFTP của họ (Authorized Keys).
- Kết nối:** Khi Camel kết nối, nó dùng Private Key để chứng minh danh tính.

## 2. Cấu hình Dependency

Đảm bảo bạn đã có `camel-ftp` trong `pom.xml`. Thư viện này sử dụng JSch (Java Secure Channel) bên dưới để thực hiện giao thức SSH.

XML

Auto (HTML, XML) ▾



```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-ftp</artifactId>
</dependency>
```

## 3. Cấu hình trong `application.yml`

Để tránh hard-code thông tin nhạy cảm trong Java, chúng ta đưa cấu hình ra file properties.

YAML

Auto (Bash) ▾



```

bank:
sftp:
  host: sftp.jpmorgan.com
  port: 22
  username: my-bank-user
  # Đường dẫn tuyệt đối tới file Private Key trên server deploy
  private-key-path: /app/secrets/keys/id_rsa_prod.pem
  # Mật khẩu bảo vệ Private Key (nếu có)
  key-passphrase: ${SFTP_KEY_PASS}
  # Thư mục chứa file SWIFT trên server đối tác
  remote-dir: /outgoing/mt950
  # Thư mục local để lưu file tải về
  local-dir: data/swift/inbox
  # File chứa fingerprint của server đối tác (Chống Man-in-the-Middle)
  known-hosts-file: /app/secrets/keys/known_hosts

```

## 4. Xây dựng Route Camel (Java Code)

Đây là đoạn code Route hoàn chỉnh xử lý các yêu cầu khắt khe của ngân hàng:

Java

Auto (Scala) ▾



```

import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class SecureSftpRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        // Xây dựng chuỗi URI kết nối SFTP
        // Cú pháp: sftp://host:port/directory?options
        String sftpUri = new StringBuilder("sftp://{{bank.sftp.host}}:" +
        {{bank.sftp.port}}/{{bank.sftp.remote-dir}}") +
            .append("?username={{bank.sftp.username}}")
    }
}

```

```
// --- CẤU HÌNH BẢO MẬT QUAN TRỌNG ---  
  
// 1. Chỉ định Private Key  
.append("&privateKeyFile={{bank.sftp.private-key-path}}")  
.append("&privateKeyPassphrase={{bank.sftp.key-  
passphrase}}")  
  
// 2. Bảo mật Host Key (Chống giả mạo Server)  
// 'yes': Bắt buộc server đích phải có trong file  
known_hosts  
.append("&strictHostKeyChecking=yes")  
.append("&knownHostsFile={{bank.sftp.known-hosts-file}}")  
  
// --- CẤU HÌNH VẬN HÀNH ---  
  
// 3. Binary transfer: Bắt buộc để không làm hỏng  
encoding file  
.append("&binary=true")  
  
// 4. Hành động sau khi tải xong:  
// Move file vào thư mục 'backup' trên server đối tác để  
tránh tải lại  
  
.append("&move=.backup/${date:now:yyyyMMdd}/${file:name}")  
// Hoặc xóa luôn: .append("&delete=true")  
  
// 5. Reconnect: Tự động thử lại nếu mất mạng  
.append("&maximumReconnectAttempts=5")  
.append("&reconnectDelay=5000")  
  
// 6. Polling: Quét 5 phút/lần  
.append("&delay=300000")  
  
.toString();  
  
// Định nghĩa luồng  
from(sftpUri)  
.routeId("SecureSftpDownloadRoute")  
.log("Đã kết nối SFTP thành công. Đang tải file:  
${header.CamelFileName}")  
  
// Lưu file xuống ổ cứng local  
.to("file:{{bank.sftp.local-dir}}")  
  
// Sau khi lưu xong, gửi tín hiệu để Route xử lý SWIFT (đã  
viết ở bước trước) hoạt động  
.log("Đã lưu file local. Sẵn sàng parse.");  
}  
}
```

## 5. Các Vấn đề "Sống còn" trong Thực tế (Troubleshooting)

Khi triển khai thật với ngân hàng nước ngoài, bạn sẽ gặp 3 vấn đề sau, hãy lưu ý kỹ:

### A. Định dạng Private Key (PEM vs PPK)

- **Vấn đề:** Apache Camel (JSch) thường yêu cầu định dạng **OpenSSH** hoặc **PEM** (bắt đầu bằng `-----BEGIN RSA PRIVATE KEY-----`).
- **Lỗi:** Nếu bạn dùng file `.ppk` (của PuTTY), Camel sẽ báo lỗi `invalid privatekey`.
- **Giải pháp:** Dùng `ssh-keygen` hoặc PuTTYgen để convert sang định dạng PEM.

### B. Vấn đề `known_hosts` (Man-in-the-Middle)

- **Vấn đề:** Ngân hàng cực kỳ sợ bạn kết nối nhầm vào một server giả mạo (Phishing Server).
- **Giải pháp:**
  1. Lần đầu tiên, yêu cầu đội hạ tầng dùng lệnh `ssh-keyscan -H sftp.jpmorgan.com >> known_hosts` trên server production.
  2. Cấu hình `strictHostKeyChecking=yes`.
  3. Tuyệt đối **không** dùng `strictHostKeyChecking=no` trên môi trường Production (dù nó tiện cho Dev).

### C. Proxy (Firewall ngân hàng)

- **Vấn đề:** Server ứng dụng của bạn thường nằm trong mạng nội bộ và không được ra Internet trực tiếp. Nó phải đi qua một **HTTP Proxy** hoặc **SOCKS Proxy**.
- **Cấu hình Camel:** Bạn cần thêm tham số proxy vào chuỗi URI:  
Java

### D. File đang được ghi (File Locking)

- **Vấn đề:** Ngân hàng đối tác đang ghi file `MT950_today.txt` (ghi được 50%), Camel nhảy

vao tai ve → File bị lỗi (corrupted).

- **Giải pháp:** Sử dụng `readLock` :  
Java

Với cấu hình này, hệ thống của bạn sẽ tự động đăng nhập an toàn, tải file về, backup file trên server đối tác và đẩy vào quy trình Parse SWIFT mà chúng ta đã thiết kế trước đó.

## 🏗 Kiến trúc Schema Tổng Quan

Hệ thống được chia thành 3 nhóm bảng chính:

- Ingestion & Data (Dữ liệu đầu vào):** Nơi lưu trữ mọi giao dịch từ mọi nguồn.
- Configuration (Cấu hình):** Định nghĩa các quy tắc khớp (Rule Engine).
- Result & Exception (Kết quả & Ngoại lệ):** Lưu trữ kết quả khớp và trạng thái xử lý lỗi.

### 1. Bảng Dữ liệu Cốt lõi: `UNIFIED_TRANSACTION`

Đây là bảng quan trọng nhất. Tất cả dữ liệu (SWIFT MT940, ATM Log, Core Banking, Excel) sau khi parse đều đổ vào đây.

Bảng này sử dụng chiến lược **Partitioning** (Phân vùng) theo `transaction_date` để đảm bảo hiệu năng khi dữ liệu lên tới hàng tỷ bản ghi.

SQL

Auto (SQL) ▾



```
CREATE TABLE unified_transaction (
    id                      BIGSERIAL PRIMARY KEY,
    -- 1. Định danh nguồn dữ liệu
    batch_id                BIGINT NOT NULL,           -- ID của lần import
    file/batch
    source_system            VARCHAR(50) NOT NULL,      -- VD: "CORE", "VISA",
    "SWIFT", "ATM_HN"
    recon_type               VARCHAR(50) NOT NULL,      -- VD: "ATM_RECON",
    "NOSTRO_RECON"
    -- 2. Các trường Cốt lõi (Core Fields - dùng để tính toán)
    transaction_date         TIMESTAMP NOT NULL,        -- Ngày giao dịch
    value_date               DATE,                      -- Ngày hiệu lực (Quan
```

*trọng cho SWIFT)*

```

amount           DECIMAL(19, 4) NOT NULL, -- Số tiền
currency        VARCHAR(3) NOT NULL,      -- VD: VND, USD
dr_cr_type     CHAR(1) NOT NULL,       -- 'D' (Debit) hoặc
'C' (Credit)

-- 3. Các trường Khóa để So khớp (Matching Keys)
-- Thay vì tạo cột riêng (card_no, swift_ref...), ta dùng các cột
Key tổng quát
primary_key_val  VARCHAR(255),          -- Khóa chính: Ref No,
Trace No ...
secondary_key_val VARCHAR(255),         -- Khóa phụ: Card
Number, Account No ...

-- 4. Dữ liệu Linh hoạt (The Magic Column)
-- Lưu toàn bộ các trường đặc thù của nghiệp vụ vào đây dưới dạng
JSON
-- VD SWIFT: {"imad": "...", "sender_bic": "..."}
-- VD ATM: {"terminal_id": "ATM01", "cassette_count": 5}
metadata        JSONB,                 

-- 5. Trạng thái (Lifecycle)
recon_status    VARCHAR(20) DEFAULT 'UNMATCHED', -- UNMATCHED,
MATCHED, SUGGESTED
match_group_id   BIGINT,                 -- Link tới bảng kết
quả khớp

created_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) PARTITION BY RANGE (transaction_date);

```

## Tại sao thiết kế này hiệu quả?

- **SWIFT:** `primary_key_val` lưu số Ref (Tag 20), `metadata` lưu thông tin phí, ngân hàng trung gian.
- **ATM:** `primary_key_val` lưu số Trace Number (Seq No), `metadata` lưu mã lỗi phần cứng.
- **Thẻ:** `primary_key_val` lưu Auth Code, `secondary_key_val` lưu Masked Card Number (4 số cuối).

## 2. Bảng Kết quả Khớp: `MATCH_GROUP` & `MATCH_ITEM`

Đối soát không chỉ là 1-1. Nó có thể là **1-N** (1 lệnh chuyển lương tổng khớp với 1000 lệnh ghi có

SQL

Auto (SQL) ▾



```
-- Bảng gom nhóm các giao dịch đã khớp với nhau
CREATE TABLE match_group (
    id                  BIGSERIAL PRIMARY KEY,
    recon_job_id       VARCHAR(50),          -- Mã Job đối soát
    match_date         TIMESTAMP,            -- Ngày chạy khớp
    match_rule_id      VARCHAR(50),          -- Khớp theo luật nào? (VD:
    "Rule_Exact_Match")

    total_debit_amt   DECIMAL(19,4),        -- Tổng tiền bên Nợ
    total_credit_amt  DECIMAL(19,4),        -- Tổng tiền bên Có
    variance_amt       DECIMAL(19,4),        -- Số tiền lệch (nếu cho phép
    lệch phí)

    status             VARCHAR(20)           -- APPROVED, PENDING REVIEW
);

-- Bảng liên kết (Junction Table) giữa Transaction và Group
-- Một transaction chỉ thuộc về 1 Group tại 1 thời điểm
CREATE TABLE match_item (
    group_id           BIGINT REFERENCES match_group(id),
    transaction_id     BIGINT,                -- Link tới
    unified_transaction
    txn_source         VARCHAR(50)           -- Để truy vấn nhanh (Nguồn A
    hay B)
);
```

### 3. Bảng Cấu hình & Quy tắc: RECON\_CONFIG

Để hệ thống linh hoạt ("Dynamic"), ta không hard-code logic khớp trong Java (`if ... else`), mà lưu quy tắc trong Database hoặc file cấu hình.

SQL

Auto (SQL) ▾



```

CREATE TABLE recon_rule_config (
    id                  VARCHAR(50) PRIMARY KEY,   -- VD:
    "SWIFT_NOSTRO_USD"
    description        VARCHAR(255),
    -- Định nghĩa dung sai (Tolerance)
    tolerance_amount   DECIMAL(10,2) DEFAULT 0,   -- Cho phép lệch bao
    nhiêu?
    tolerance_days     INT DEFAULT 0,           -- Cho phép lệch ngày
    T+ mấy?

    -- Định nghĩa biểu thức khớp (Advanced)
    -- VD: "T1.amount = T2.amount AND T1.ref = T2.ref"
    match_logic_script TEXT,
    active             BOOLEAN DEFAULT TRUE
);

```

## 4. Bảng Xử lý Ngoại lệ: EXCEPTION\_CASE

Dành cho nhân viên vận hành (Back-office) xử lý các giao dịch không khớp.

SQL

Auto (SQL) ▾



```

CREATE TABLE exception_case (
    id                  BIGSERIAL PRIMARY KEY,
    transaction_id     BIGINT,           -- Link tới giao dịch bị lỗi
    exception_type     VARCHAR(50),      -- VD: MISSING_IN_CORE,
    AMOUNT_MISMATCH
    priority            VARCHAR(10),       -- HIGH, MEDIUM, LOW
    assigned_user       VARCHAR(50),      -- Người đang xử lý
    investigation_note TEXT,           -- Ghi chú điều tra
    status              VARCHAR(20),       -- OPEN, INVESTIGATING,
    RESOLVED, WRITTEN_OFF
    resolution_action   VARCHAR(50),      -- VD: MANUAL_MATCH, REFUND,
    -----

```

~~CHANGE\_UFT~~

updated_at	TIMESTAMP
);	

## 💡 Ví dụ Dữ liệu Thực tế (Data Visualization)

Hãy xem cách bảng **UNIFIED\_TRANSACTION** lưu trữ dữ liệu hỗn hợp:

ID	Source	Type	Amt	Key_Val (Ref)	Metadata (JSONB)
1	CORE	SWIFT	1000	FT12345	{"sender": "VNCBVNVX", "charge": "OUR"}
2	SWIFT_MT950	SWIFT	995	FT12345	{"msg_type": "950", "intermediary_fee": 5}
3	ATM_LOG	ATM	500	SEQ_9988	{"error_code": "Hardware_Dispatch_Fail", "cassette": 2}
4	VISA_FILE	CARD	25.5	AUTH_6677	{"merchant_id": "MCDONALDS", "mcc": "5814"}

## Chiến lược Indexing (Chỉ mục)

Để tìm kiếm nhanh trên cột JSONB, PostgreSQL hỗ trợ **GIN Index**:

SQL

Auto (SQL) ▾



```
-- Index cho các trường cơ bản
CREATE INDEX idx_txn_date ON unified_transaction (transaction_date);
CREATE INDEX idx_primary_key ON unified_transaction (primary_key_val);

-- Index đặc biệt cho JSONB (Tìm kiếm nhanh trong Metadata)
-- Ví dụ: Tìm tất cả giao dịch ATM có lỗi phần cứng
CREATE INDEX idx_metadata ON unified_transaction USING GIN (metadata);
```

## Triển khai trên Java Spring Boot

Khi map bảng này vào Java Entity, bạn sử dụng thư viện `hibernate-types` để map cột JSONB vào `Map<String, Object>` hoặc `JsonNode`.

Java

Auto (Kotlin) ▾



```
@Entity
@Table(name = "unified_transaction")
@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class) // Hỗ trợ
JSONB
public class UnifiedTransaction {

    @Id
    private Long id;

    private BigDecimal amount;

    @Column(name = "primary_key_val")
    private String refNo;

    @Type(type = "jsonb")
    @Column(columnDefinition = "jsonb")
    private Map<String, Object> metadata; // Linh hoạt lưu mọi thứ

    ...
}
```

Với thiết kế này, Service của bạn có thể mở rộng để đối soát bất kỳ nghiệp vụ mới nào (Ví dụ: Đối soát Tiền điện tử, Chứng khoán) chỉ bằng cách cấu hình Rule mới mà **không cần sửa đổi cấu trúc Database (No Schema Change)**.

Chào bạn, chuyển từ đối soát cuối ngày (Batch) sang **Real-time Reconciliation (Đối soát thời gian thực)** là bước nhảy vọt về công nghệ. Nó giúp ngân hàng phát hiện sự cố (ví dụ: Core trừ tiền nhưng Ví điện tử chưa nhận) chỉ trong vài giây thay vì đợi đến sáng hôm sau.

Với **Apache Kafka** và **Kafka Streams (KStreams)**, chúng ta sẽ xây dựng một hệ thống xử lý luồng sự kiện (Event Streaming) mạnh mẽ.

Dưới đây là thiết kế chi tiết.

---

## III 1. Kiến trúc Tổng thể (High-Level Architecture)

Trong mô hình này, chúng ta không "kéo" (pull) dữ liệu từ Database nữa. Thay vào đó, các hệ thống nguồn sẽ "đẩy" (push) sự kiện (Event) vào Kafka ngay khi giao dịch phát sinh.

Các thành phần chính:

### 1. Input Topics (Đầu vào):

- **topic-core-banking** : Chứa sự kiện trừ tiền từ Core.
- **topic-payment-gateway** : Chứa sự kiện giao dịch thành công từ Cổng thanh toán.

### 2. Stream Processing Engine (Bộ xử lý):

Sử dụng thư viện **Kafka Streams** (nhúng trong Spring Boot Microservice) để thực hiện logic so khớp (Join).

### 3. State Store (Bộ nhớ tạm):

Sử dụng **RocksDB** (tích hợp sẵn trong KStreams) để lưu trạng thái giao dịch đang chờ khớp (ví dụ: Core đến trước, Gateway chưa đến).

### 4. Output Topics (Đầu ra):

- **topic-recon-matched** : Giao dịch khớp → Đẩy về Dashboard/Analytics.
- **topic-recon-mismatch** : Giao dịch lệch/thiếu → Đẩy cảnh báo (Alert) hoặc lưu vào DB để xử lý sau.

---

## ❖ 2. Thiết kế Topology (Luồng xử lý dữ liệu)

Logic đối soát trong Kafka thực chất là bài toán **Stream-Stream Join** (Khớp hai dòng dữ liệu đang chảy).

### Bước 1: Chuẩn hóa & Tái phân vùng (Rekeying)

Dữ liệu từ Core và Gateway có thể dùng Key khác nhau. Để khớp được, chúng phải có cùng

## Partition Key.

- Core Event (Key = TransactionID): **T123**
- Gateway Event (Key = OrderID): **ORDER\_T123**
- Hành động:** Map lại (Remap) sao cho cả 2 stream đều sử dụng **TransactionID** làm Key.

## Bước 2: Windowing (Cửa sổ thời gian)

Trong thực tế, 2 sự kiện không bao giờ đến cùng lúc. Core có thể xong lúc 10:00:00, nhưng Gateway 10:00:05 mới trả về.

- Giải pháp:** Sử dụng **Sliding Window** (Cửa sổ trượt) hoặc **Tumbling Window**.
- Quy tắc:** "Chờ 2 sự kiện khớp Key xuất hiện trong khoảng thời gian **5 phút**. Nếu quá 5 phút mà chỉ có 1 bên → Báo lỗi."

## Bước 3: Joining (Khớp lệnh)

Thực hiện **Outer Join** giữa 2 Stream.

- Nếu có cả 2 (Left + Right) → Kiểm tra số tiền → Output **MATCHED** .
- Nếu chỉ có Left (Core) sau khi hết Window → Output **MISSING\_IN\_GATEWAY** .
- Nếu chỉ có Right (Gateway) sau khi hết Window → Output **MISSING\_IN\_CORE** .

## 3. Triển khai Code (Spring Cloud Stream & Kafka Streams)

Dưới đây là ví dụ code Java sử dụng API Kafka Streams DSL.

### Cấu hình Dependency

XML

Auto (HTML, XML) ▾



```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
</dependency>
<dependency>
```

```
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-stream-binder-kafka-streams</artifactId>
</dependency>
```

## Logic Stream Join

Java

Auto (Dart) ▾



```
@Bean
public java.util.function.BiConsumer<KStream<String, CoreTxn>,
KStream<String, GatewayTxn>> process() {
    return (coreStream, gatewayStream) → {

        // 1. Định nghĩa giới hạn thời gian (Window 5 phút)
        JoinWindows joinWindow =
        JoinWindows.ofTimeDifferenceWithNoGrace(Duration.ofMinutes(5));

        // 2. Định nghĩa Serde (Serializer/Deserializer) cho JSON
        StreamJoined<String, CoreTxn, GatewayTxn> joinParams =
            StreamJoined.with(Serdes.String(), new JsonSerde<>
        (CoreTxn.class), new JsonSerde<>(GatewayTxn.class));

        // 3. Thực hiện OUTER JOIN
        KStream<String, ReconResult> joinedStream = coreStream.outerJoin(
            gatewayStream,
            (core, gateway) → {
                // Logic so sánh diễn ra ở đây
                ReconResult result = new ReconResult();

                if (core ≠ null && gateway ≠ null) {
                    // Cả 2 đều tồn tại → So sánh số tiền
                    if (core.getAmount().compareTo(gateway.getAmount()) =
                        0) {
                        result.setStatus("MATCHED");
                    } else {
                        result.setStatus("MISMATCH_AMOUNT");
                    }
                }
                result.setDiff(core.getAmount().subtract(gateway.getAmount()));
            }
        } else if (core ≠ null) {
            // Chỉ có Core (Gateway chưa đến hoặc mất)
            result.setStatus("DENIED_GATEWAY");
        }
    }
}
```

```

        result.setStatus("PENDING_GATEWAY"),
    } else {
        // Chỉ có Gateway
        result.setStatus("PENDING_CORE");
    }
    return result;
},
joinWindow,
joinParams
);

// 4. Phân loại và đẩy ra Topic đích
joinedStream.filter((k, v) → "MATCHED".equals(v.getStatus()))
    .to("topic-recon-matched");

joinedStream.filter((k, v) → !"MATCHED".equals(v.getStatus()))
    .to("topic-recon-pending");
};

}
}

```

## 4. Vấn đề nan giải: Xử lý Timeout (Sự kiện bị thiếu)

Đoạn code trên có một vấn đề: **OuterJoin** trong Kafka Streams sẽ phát ra sự kiện ngay lập tức khi có 1 bên đến (**PENDING**). Nhưng làm sao biết khi nào thì **Hết thời gian chờ (Timeout)** để kết luận là "Giao dịch lỗi"?

Có 2 cách xử lý nâng cao:

### Cách A: Sử dụng Punctuator (Schedule check) - Khó nhưng Chính xác

Thay vì dùng Join đơn thuần, bạn nạp dữ liệu vào **State Store** và đặt một "đồng hồ báo thức" (Punctuator).

- Mỗi phút, quét State Store một lần.
- Nếu thấy giao dịch nào có Timestamp cũ hơn 5 phút mà vẫn chưa khớp → Đẩy ra topic **topic-recon-timeout** → Gửi Alert cho vận hành.

### Cách B: Stream-Table Duality (Hybrid) - Dễ hơn

- 1. Stream:** Kafka xử lý việc khớp **MATCHED** nhanh nhất có thể.
- 2. Table (Database):** Những giao dịch **PENDING** (chưa khớp) được đẩy xuống Database (PostgreSQL).
- 3. Scheduled Job:** Một Job nhỏ chạy trên DB mỗi 5 phút:  
SQL

Những dòng này chính là giao dịch bị lỗi/treo cần xử lý.

## ⚡ 5. Ưu và Nhược điểm của Real-time Recon

Đặc điểm	Batch Reconciliation (Truyền thống)	Real-time Reconciliation (Kafka)
<b>Độ trễ</b>	T+1 (Hôm sau mới biết).	T+0 (Biết ngay sau vài giây).
<b>Độ phức tạp</b>	Trung bình.	Rất cao (State management, Time windows).
<b>Xử lý sửa sai</b>	Dễ (Sửa file rồi chạy lại Batch).	Khó (Dữ liệu đã trôi qua, phải đẩy event bù trừ).
<b>Hệ tầng</b>	Database SQL mạnh.	Cụm Kafka, Zookeeper, RocksDB.
<b>Mục đích</b>	<b>Quyết toán tài chính</b> (Chính xác tuyệt đối).	<b>Giám sát vận hành</b> (Phát hiện lỗi nhanh).

### 💡 Lời khuyên cho Kiến trúc sư

Đừng bỏ Batch hoàn toàn. Hãy dùng mô hình **Hybrid (Lai)**:

- 1. Kafka (Real-time):** Dùng để phát hiện lỗi "nóng" (ví dụ: Hệ thống đối tác bị sập, tỷ lệ lỗi tăng vọt) để IT/Vận hành can thiệp ngay.
- 2. Batch (End-of-Day):** Vẫn chạy vào cuối ngày để chốt sổ sách kế toán chính thức (Final Ledger), đảm bảo không sót bất kỳ giao dịch nào mà Kafka có thể bị trôi (due to retention policy).

Đây là sơ đồ và thiết kế chi tiết cho **Mô hình Hybrid (Lai)** – sự kết hợp giữa **Real-time** (tốc độ) và **Batch** (chính xác). Trong giới kỹ thuật dữ liệu, mô hình này còn được gọi là **Lambda Architecture**.

Mô hình này giúp ngân hàng đạt được hai mục tiêu cùng lúc:

- Vận hành:** Biết ngay hệ thống đang gặp lỗi gì (trong vòng vài giây).
- Tài chính:** Chốt sổ sách chính xác tuyệt đối vào cuối ngày.

## III Sơ đồ Kiến trúc Hybrid (Lambda Architecture)

Hệ thống được chia thành 2 luồng xử lý chạy song song:

### 1. Luồng Nóng (Speed Layer - Real-time)

- Công nghệ:** Apache Kafka, Kafka Streams (hoặc Flink), Redis.
- Mục tiêu:** Giám sát & Cảnh báo (Monitoring & Alerting).
- Hoạt động:**
  - Khi giao dịch phát sinh, nó đi qua Kafka.
  - Stream Processor khớp sơ bộ (ví dụ: khớp ID, số tiền).
  - Nếu lệch → Gửi cảnh báo lên Dashboard (Grafana) hoặc Telegram/Slack cho đội vận hành.
  - Lưu ý:* Kết quả ở đây là "tạm tính" (Intraday status), chưa dùng để hạch toán vào sổ cái.

### 2. Luồng Lạnh (Batch Layer - End of Day)

- Công nghệ:** Spring Batch, PostgreSQL/Oracle, Hadoop/S3.
- Mục tiêu:** Quyết toán Tài chính (Financial Settlement).
- Hoạt động:**
  - Cuối ngày, toàn bộ dữ liệu từ Kafka được đổ xuống Database/Data Lake (qua Kafka Connect).
  - Spring Batch chạy job tổng hợp toàn bộ giao dịch trong ngày + các giao dịch treo của ngày hôm trước.
  - Thực hiện các logic phức tạp (N-to-N matching, tính phí bậc thang, xử lý chênh lệch tỷ giá).
  - Kết quả ở đây là "chính thức" (Final status) → Ghi vào Core Banking.

## ❖ Thiết kế Chi tiết Dòng dữ liệu (Data Flow)

Để tránh việc phải xây dựng 2 hệ thống ingest dữ liệu riêng biệt (rất tốn kém), chúng ta sử dụng **Kafka làm trung tâm trung chuyển**.

## Bước 1: Ingestion (Thu thập chung)

Tất cả các nguồn (Core Banking, Payment Gateway, ATM) đều đẩy dữ liệu vào **Kafka Topics** ngay lập tức.

- Đây là "Source of Truth" (Nguồn sự thật) đầu tiên.

## Bước 2: Tách Luồng (Forking)

Tại đây, dữ liệu sẽ làm 2 nhánh:

### Nhánh A: Xử lý Nóng (Real-time Stream)

- Kafka Streams App:** Đọc topic `txn-events`.
- Logic:** Thực hiện `Windowed Join` (trong 5 phút).
- Output:**
  - Nếu khớp: Đếm số lượng để hiển thị lên Dashboard (VD: "Đã xử lý 1 triệu giao dịch, tổng 500 tỷ").
  - Nếu lệch (Timeout): Bắn Alert ngay lập tức "Cảnh báo: Tỷ lệ lỗi Gateway A đang tăng vọt lên 15%".
  - Tác dụng:** Giúp IT phát hiện server đối tác bị sập để chuyển luồng (Routing) sang đối tác khác ngay lập tức, giảm thiểu thiệt hại.

### Nhánh B: Lưu trữ & Xử lý Lạnh (Batch Storage)

- Kafka Connect (Sink Connector):** Tự động "hút" dữ liệu từ Kafka Topic và đổ vào bảng `RAW_TRANSACTIONS` trong Database (PostgreSQL/Oracle) hoặc lưu file xuống S3 (Data Lake).
- Spring Batch Job (23:00 hàng ngày):**
  - Đọc dữ liệu từ bảng `RAW_TRANSACTIONS`.
  - Đọc thêm File đối soát (MT950/Excel) do đối tác gửi về lúc cuối ngày (File này thường không có realtime).
  - Chạy thuật toán đối soát chính xác 100% (bao gồm cả việc khớp lại những giao dịch mà Real-time có thể đã báo sai do độ trễ mạng).
  - Sinh ra **Bút toán điều chỉnh (Adjustment Entries)** và gửi vào Core Banking.

## Bảng so sánh vai trò

Đặc điểm	Luồng Nóng (Kafka Real-time)	Luồng Lạnh (Spring Batch)
Dữ liệu đầu vào	Sự kiện đơn lẻ (Events), API Logs.	File tổng hợp (Settlement Files), DB Dump
Độ trễ	Giây (Seconds).	Ngày (T+1).
Độ chính xác	95-99% (Có thể sai do mạng lag, data đến muộn).	100% (Dữ liệu tinh, đầy đủ).
Mục đích chính	<b>Operational Health:</b> Hệ thống có sống không? Có bị tấn công không?	<b>Financial Integrity:</b> Tiền có khớp không? Sách có cân không?
Hành động	Gửi Cảnh báo (Alert), chặn giao dịch rủi ro.	Hạch toán (Accounting), hoàn tiền (Refund), thu phí.

## Tại sao phải cần cả hai? (Case Study thực tế)

Hãy tưởng tượng kịch bản: **Cáp quang biển bị đứt lúc 10h sáng.**

### 1. Nếu chỉ có Batch:

- Giao dịch quốc tế bị lỗi hàng loạt. Ngân hàng không biết gì cả.
- Đến 23h đêm chạy Batch mới thấy "Chết rồi, lệch 5000 giao dịch!". Lúc này đã quá muộn, khách hàng đã khiếu nại khắp nơi.

### 2. Nếu chỉ có Real-time:

- Hệ thống báo lỗi lúc 10h05. IT chuyển line dự phòng. Ôn.
- Tuy nhiên, Real-time rất khó xử lý các loại phí phức tạp (ví dụ: phí bậc thang - giao dịch thứ 1000 mới được giảm giá). Nếu dùng số liệu Real-time để hạch toán kê toán, số liệu sẽ bị sai lệch lỗ lãi.

### 3. Mô hình Hybrid:

- **10h05 (Real-time):** Báo động đỏ → IT khắc phục → Giữ uy tín ngân hàng.
- **23h00 (Batch):** Tính toán lại toàn bộ phí, tỷ giá chính xác của ngày hôm đó → Đảm bảo không mất tiền của ngân hàng.

Đây chính là mô hình tiêu chuẩn vàng (Gold Standard) cho các hệ thống ngân hàng số (Digital Banking) hiện đại.

## Ý nghĩa chính của đoạn mô tả

Bạn đang nói với team rằng: **Tại sao phải đối soát (reconciliation) trong ngành thanh toán?**

Và bạn gợi ý một cách diễn đạt dễ hiểu hơn:

👉 **Ai đang chi tiền cho ai?**

Hay: **Tiền do bên nào ứng ra – và ai phải trả lại cho ai?**

## Ví dụ ATM: Vì sao cần đối soát?

✓ Khách A của ngân hàng ABC rút tiền tại ATM của ngân hàng XYZ

→ ATM XYZ lấy tiền mặt ra khỏi két và đưa cho khách.

Điều đó nghĩa là:

- XYZ đã ứng tiền mặt cho khách ABC.
- Vì vậy ABC phải trả lại số tiền đó cho XYZ trong quá trình quyết toán liên ngân hàng.

Bạn hỏi thêm:

👉 “Ngân hàng XYZ có thể lấy tiền trực tiếp từ ngân hàng ABC không?”

→ Không, ngân hàng không tự cẩn trừ trực tiếp.

→ Họ phải dựa vào **ghi nhận giao dịch** từ nhiều hệ thống → sau đó **đối soát & quyết toán** qua hệ thống bù trừ (Napas, Visa, MasterCard...).

## Vậy đối soát để làm gì?

💡 Để đảm bảo rằng **tất cả các hệ thống đều ghi nhận đúng cùng một giao dịch**, cụ thể là:

- Khách thật sự rút bao nhiêu?
- ATM có đưa tiền không?
- Switch có ghi log không?
- Core Banking của ngân hàng phát hành có trừ tiền không?
- Mạng thẻ (Visa/Rupay/NFS...) có ghi nhận giao dịch?

Nếu có bất kỳ nơi nào **thiếu hoặc sai** → sẽ tranh chấp → cần điều tra.

## Giải thích 2-Way, 3-Way, 4-Way Reconciliation

Dưới đây là giải thích theo hướng **dễ hiểu – bám nghiệp vụ ATM/thẻ**.

## ✓ 2-Way Reconciliation

Đối soát giữa **hai hệ thống**:

1. Payment processing switch (Switch)
2. Core banking system

☞ Dùng để kiểm tra:

- Switch nói "có giao dịch rút tiền",
- Core Banking có ghi trừ tiền khách tương ứng không?

Nếu không khớp → nghi lỗi hoặc thiếu log → phải điều tra.

## ✓ 3-Way Reconciliation

Thêm một lớp đối chiếu thứ ba:

1. Switch
2. Core Banking
3. Card network files (NFS, Rupay, Visa, Mastercard...)

☞ Mục tiêu:

Đảm bảo rằng **ngân hàng → switch → mạng thẻ** đều thống nhất về:

- Số tiền
- Thời gian
- Terminal ID
- Mã giao dịch

Nếu mạng thẻ báo "giao dịch thành công", nhưng ngân hàng phát hành không trừ tiền → ngân hàng phải **trả tiền lại** cho ngân hàng chấp nhận (acquirer).

## ✓ 4-Way Reconciliation

Đối soát với **4 nguồn dữ liệu**:

1. EJ/Transaction log từ ATM/POS
2. Switch
3. Core Banking
4. Card network files

☞ EJ (Electronic Journal) = nhật ký từ chính máy ATM/POS → log do máy tạo, gồm:

- Số tiền thực sự đã nhả
- Ảnh/record của giao dịch
- Mã lỗi

**Đây là bằng chứng mạnh nhất.**

→ Dùng trong trường hợp có tranh chấp, ví dụ:

- ATM mất tiền nhưng không ghi log
- Khách không nhận tiền nhưng tài khoản bị trừ
- Switch hoặc Core Banking ghi nhận sai

## Những dữ liệu quan trọng để đối chiếu

Các hệ thống phải khớp về:

- a. **Terminal ID** (mã máy ATM/POS)
- b. **Thời gian giao dịch**
- c. **Số tiền giao dịch**

Trong điều tra sâu hơn sẽ cần:

- Số thẻ (đã masked)
- Số tài khoản (masked)

## Tóm lại (cách nói ngắn gọn cho team)

**Đối soát = kiểm tra để biết “ai nợ ai bao nhiêu tiền”.**

Ví dụ:

- ATM của ngân hàng XYZ đã đưa tiền mặt cho khách ABC.
- Vì vậy ABC phải chuyển tiền lại cho XYZ trong quá trình quyết toán.
- Muốn làm được điều đó → cần đối chiếu dữ liệu từ 2, 3 hoặc 4 hệ thống để chắc chắn rằng giao dịch là thật, số tiền đúng, và không thiếu log.

Nếu bạn muốn, mình có thể vẽ lại sơ đồ, hoặc **viết lại bản giải thích ngắn gọn cho nhân viên mới**, hoặc **giải thích theo kiểu kể chuyện** cho dễ nhớ.

Dưới đây là **bản thiết kế hoàn chỉnh cho một “Reconciliation Service”** trong hệ thống thanh toán/ATM/POS — theo chuẩn ngân hàng hiện đại.

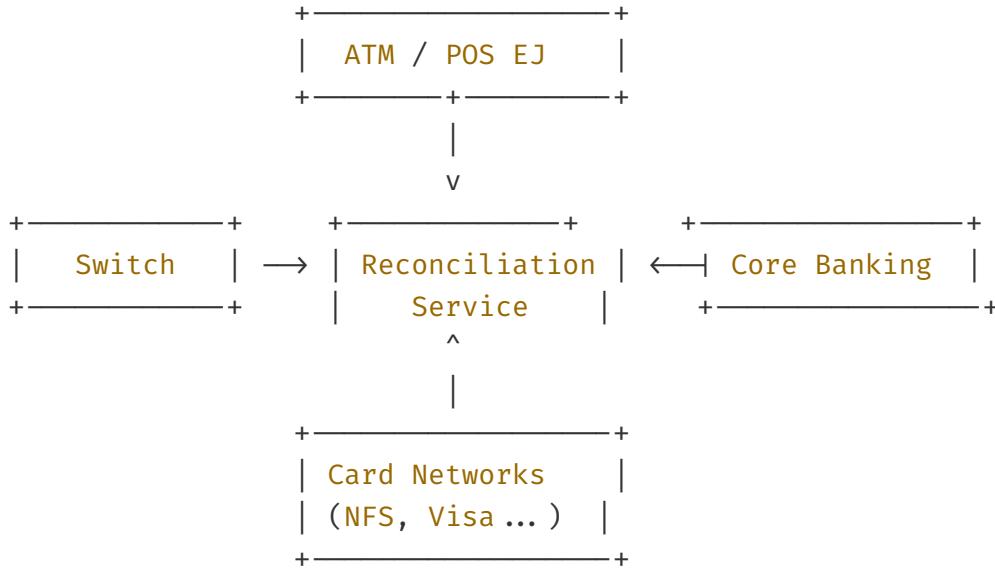
Thiết kế được chia theo **kiến trúc, luồng dữ liệu, module, DB schema, API, batch, alert, và mở rộng tương lai**.

## 1. Mục tiêu của Reconciliation Service

- Đối chiếu giao dịch từ nhiều nguồn (Switch, Core Banking, Network, ATM EJ).
- Xác định giao dịch **khớp / lệch / thiếu / trùng**.
- Sinh **báo cáo settlement** cho mỗi ngày.
- Hỗ trợ **tranh chấp (dispute)**.
- Tự động gửi **alert khi có lỗi**.

## 2. Kiến trúc tổng thể (High-level Architecture)

Auto (Haskell) ▾



Reconciliation Service gồm 4 module chính:

1. **Data Ingestion Layer** – nạp file/log từ các nguồn
2. **Matching Engine** – logic đối soát
3. **Settlement Engine** – tính toán tiền nợ/tiền trả
4. **Reporting + Dashboard**

## 3. Các thành phần chi tiết

### 3.1 Data Ingestion (Input Collector)

Hệ thống nhận dữ liệu từ:

Nguồn	Dạng file	Tần suất
Switch	CSV/JSON	real-time + end-of-day
Core Banking	DB pull / file	end-of-day
Network (NFS, Visa...)	settlement file	daily
ATM/POS EJ	text/log file	end-of-day

Module thực hiện:

- Validate format
- Convert về **Unified Transaction Model**
- Gán **source\_type** = switch/core/network/ej

## 3.2 Unified Transaction Model (UTM)

Dữ liệu quy về một dạng chuẩn:

Auto ▾



```
{  
    txn_id,  
    rrn,  
    stan,  
    card_number_masked,  
    account_number_masked,  
    amount,  
    currency,  
    txn_type,  
    txn_time,  
    terminal_id,  
    acquirer_id,  
    issuer_id,  
    source_type,  
    raw_data  
}
```

## 3.3 Matching Engine (Logic đối soát)

Hỗ trợ:

### ✓ 2-Way

Switch ↔ Core

### ✓ 3-Way

Switch ↔ Core ↔ Network

### ✓ 4-Way

EJ ↔ Switch ↔ Core ↔ Network

### 🔍 Matching Rules:

Dùng 3 tiêu chí chính:

## 1. Terminal ID (exact match)

## 2. Transaction Amount

## 3. Transaction Time ± 5 phút

Additional:

- RRN hoặc STAN
- PAN masked
- Trace ID

## 🔍 4 trạng thái đối soát:

1. **Matched** (khớp đủ các nguồn)
2. **Missing** (một nguồn không có)
3. **Mismatch** (có nhưng không khớp số tiền/thời gian)
4. **Duplicate** (1 giao dịch xuất hiện 2 lần cùng nguồn)

## ⌚ Matching Flow

Auto (Visual Basic .NET) ▾



Step 1: Group giao dịch theo RRN/STAN

Step 2: Ghép EJ → Switch

Step 3: Ghép Switch → Core Banking

Step 4: Ghép Switch → Network

Step 5: Mark trạng thái đối soát

Step 6: Ghi log & audit trail

## 3.4 Settlement Engine

Dùng kết quả đối soát để tính:

- Ngân hàng **được thu** bao nhiêu
- Ngân hàng **phải trả** bao nhiêu
- Phí networks (interchange)

Ví dụ ATM:

Auto (Java) ▾



Nếu ATM XYZ nhả tiền cho khách ABC:

- XYZ ghi nhận “dispensed cash”
- ABC ghi nhận “debit”
- **Network calc** = XYZ receivable, ABC payable

Output:

- settlement\_report
- posting instruction (nếu cần ghi sổ)

## ● 3.5 Reporting & Dashboard

- Dashboard "Reconciliation Result Overview"
- Báo cáo theo từng loại:
  - ATM
  - POS
  - ECOM
  - International card (Visa/MasterCard)
- Export Excel/PDF
- API cho BI (PowerBI/Tableau)

## ■ 4. Database Schema (đủ để implement)

### Table: transactions\_raw

Dữ liệu ingest từ tất cả nguồn.

Field	Description
id	PK
rrn	reference number
stan	system trace
source_type	switch/core/network/ej
amount	số tiền
txn_time	thời gian
terminal_id	ATM/POS
raw_payload	json full

## Table: reconciled\_transactions

Field	Description
recon_id	PK
rrn	group
switch_id	FK transactions_raw
core_id	FK
network_id	FK
ej_id	FK
status	matched / mismatch / missing
remarks	ghi chú
recon_date	ngày đối soát

---

## Table: settlement\_summary

Field	Description
bank_code	acquirer/issuer
total_txn	số lượng
total_amount	số tiền
net_amount	phải trả/nhận
date	value date

---

## 5. API design (REST)

### 1. Upload file

Auto ▾



POST /recon/upload

## 2. Trigger reconciliation

Auto ▾



POST /recon/run?date=2025-03-01

## 3. Get result summary

Auto ▾



GET /recon/result?date=2025-03-01

## 4. Search detail

Auto ▾



GET /recon/txn?rrn=123456789

## 5. Export settlement

Auto ▾



GET /recon/settlement-report?date=2025-03-01

## 6. Batch jobs (Scheduler)

- 00:30: pull Core Banking log
- 01:00: ingest Switch file
- 02:00: ingest Network settlement

- 03:00: ingest ATM EJ
- 04:00: run matching engine
- 05:00: generate settlement file
- 05:30: send email/alert Slack

## 7. Alerting & Monitoring

- Missing file: notify Slack/Email
- Mismatch > X%: trigger warning
- ATM EJ not received: high risk dispute
- Network file delay: settlement risk

## 8. Mở rộng tương lai

- Real-time reconciliation (hot recon)
- Machine-learning fraud detection
- Support QR, UPI, NAPAS247
- Auto dispute filing

## Bạn muốn mình làm gì tiếp?

Tốt — mình sẽ **mở rộng & thiết kế thêm một “Payment Reconciliation Service”** (đối soát thanh toán: merchant payments, e-commerce, wallet top-up, payout, QR/QRIS, v.v.) — tích hợp với hệ Reconciliation Service ATM bạn đã có. Mình đưa ra thiết kế kỹ thuật, luồng dữ liệu, DB schema, API, luật đối soát, rules settlement, exception handling, monitoring, bảo mật và roadmap triển khai.

Mình trình bày bằng tiếng Việt, có phần mẫu JSON/SQL để dev bắt tay vào triển khai ngay.

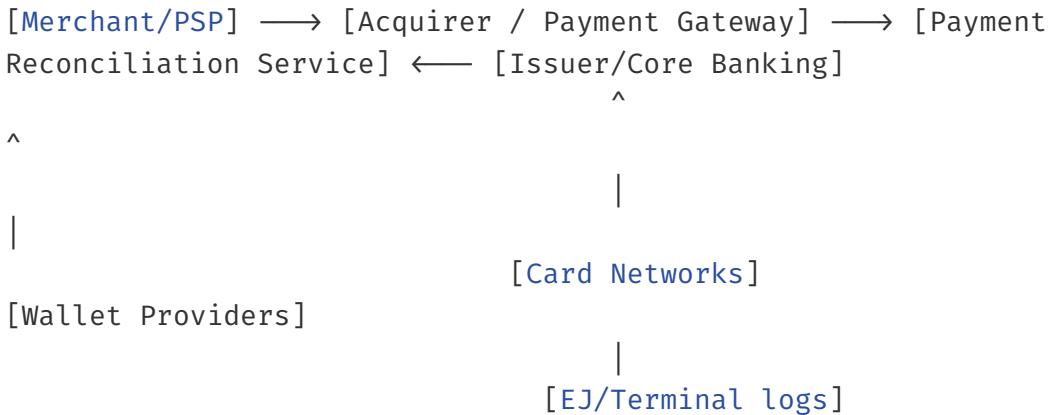
### 1 Mục tiêu (Scope)

- Đối soát giao dịch thanh toán (online/offline) giữa: **Acquirer (merchant bank), Payment Gateway/Switch, Issuer (card bank/wallet), PSP/Payment Network (Visa, Mastercard, local), Merchant systems.**
- Hỗ trợ nhiều kênh: Card, e-wallet, QR, Bank Transfer, Direct Debit, Refunds, Payouts.
- Phân loại trạng thái: **matched / missing / mismatch / pending / refunded / disputed.**
- Sinh báo cáo settlement (per merchant / per acquirer) và file cho clearing/settlement.
- Tự động reconcile real-time (hot) cho giao dịch X phút và batch EOD.

### 2 High-level Architecture (mở rộng)



Auto (C#) ▾



## Components:

- **Ingestion Layer (real-time & batch)**
- **Normalization (Unified Payment Model)**
- **Matching Engine (rules engine)**
- **Business Rules / Settlement Engine**
- **Dispute Management Module**
- **Reporting & Exporter (CSV, NACHA, ISO20022, host format)**
- **API Gateway + Admin UI / Dashboard**
- **Audit & Audit Trail store**
- **Alerting & Monitoring**

## 3 Unified Payment Model (UPM)

Chuẩn hoá mọi nguồn về một object:

Auto (Bash) ▾



```
{
  "txn_id": "uuid",
  "source": "gateway|acquirer|issuer|network|merchant|ej",
  "source_txn_id": "string",
  "rrn": "string",
  "stan": "string",
  "pan_masked": "xxxx-xxxx-xxxx-1234",
  "merchant_id": "M1234",
  "merchant_name": "ACME",
  "terminal_id": "T001",
  "amount": 10000,
  "currency": "VND",
  "fee": 200,
  "net_amount": 9800,
  "tx_type": "purchase|refund|tokenization|batch"
}
```

```

    "status":"success|failed|pending",
    "txn_time":"2025-11-30T08:12:34Z",
    "settlement_date":"2025-11-30",
    "raw_payload":{ ... }
}

```

## 4 Luồng dữ liệu & matching flow (chi tiết)

### A. Ingestion

- **Real-time:** Webhooks / MQ (Kafka) from Gateway, PSP, Issuer.
- **Batch:** SFTP file pulls (CSV, ISO8583 dump), daily network settlement files.
- Normalize và persist vào `transactions_raw`.

### B. Matching Engine

1. **Group by (primary keys):** rrn, stan, source\_txn\_id, pan\_masked, merchant\_id, txn\_date.
2. **Try 1: Exact match** on {rrn OR stan OR source\_txn\_id} AND amount AND merchant\_id → mark `matched`.
3. **Try 2: Fuzzy match** if exact fails:
  - a. amount  $\pm$  tolerance (configurable %) AND txn\_time  $\pm$  X minutes AND same merchant/terminal
  - b. pan\_masked matches / same currency
4. **Try 3: Cross-type match** for refunds/payouts (link by original\_txn\_id)
5. **If still not matched** → mark `missing` (missing\_source list) and escalate to investigation queue.

### C. Settlement

- **For each matched group compute:**
  - Gross amount per acquirer/issuer
  - Fees (interchange, acquirer fee, gateway fee) — rules engine configurable
  - Net payable per merchant/acquirer
- Produce per-merchant settlement file and per-acquirer summary.
- Generate posting instructions for GL (Core Banking).

## 5 Trạng thái & business rules

- **MATCHED**: tất cả nguồn yêu cầu đều có record và amount khớp.
- **PENDING**: record từ một bên chưa đến (chờ N minutes/hours).
- **MISMATCH**: same tx id/rrn nhưng amount khác hoặc currency mismatch.

- **MISSING**: ĐÃI CÔNG LẠI KHÔNG CÓ RECORD SAU TÍM → LẤU TICKET.
- **REFUNDED**: refund logged & linked to original txn.
- **DISPUTED**: merchant/issuer opened dispute → link tới dispute workflow.

### Rules examples:

- **Tolerance**: Card transaction tolerance = 0.00% (strict); Wallet topup tolerance = 1% (configurable)
- **Time window**: Real-time match window = ±2 minutes; EOD batch window = same day
- **Duplicate detection**: same pan\_masked + amount + terminal\_id + within 30s ⇒ duplicate

## 6 DB Schema (core tables) — mẫu SQL

Auto (SQL) ▾



```
-- transactions_raw
CREATE TABLE transactions_raw (
    id UUID PRIMARY KEY,
    source VARCHAR(50),
    source_txn_id VARCHAR(128),
    rrn VARCHAR(32),
    stan VARCHAR(32),
    pan_masked VARCHAR(32),
    merchant_id VARCHAR(64),
    terminal_id VARCHAR(64),
    amount NUMERIC(18,2),
    currency VARCHAR(3),
    txn_type VARCHAR(32),
    status VARCHAR(32),
    txn_time TIMESTAMP,
    settlement_date DATE,
    raw_payload JSONB,
    created_at TIMESTAMP DEFAULT now()
);
```

```
-- reconciled_groups
CREATE TABLE reconciled_groups (
    recon_id UUID PRIMARY KEY,
    group_key VARCHAR(128),
    status VARCHAR(32),
    total_amount NUMERIC(18,2),
    created_at TIMESTAMP DEFAULT now(),
    updated_at TIMESTAMP DEFAULT now()
);
```

```
-- reconciled_items (links)
CREATE TABLE reconciled_items (
    id UUID PRIMARY KEY,
```

```

recon_id UUID REFERENCES reconciled_groups(recon_id),
raw_txn_id UUID REFERENCES transactions_raw(id),
role VARCHAR(32), -- 'acquirer'/'issuer'/'network'/'merchant_ej'
created_at TIMESTAMP DEFAULT now()
);

-- settlement_summary
CREATE TABLE settlement_summary (
    id UUID PRIMARY KEY,
    recon_id UUID,
    party_id VARCHAR(64),
    party_type VARCHAR(32), -- merchant/acquirer/issuer
    gross_amount NUMERIC(18,2),
    total_fee NUMERIC(18,2),
    net_amount NUMERIC(18,2),
    settlement_date DATE,
    created_at TIMESTAMP DEFAULT now()
);

```

## 7 API Design (mẫu)

- `POST /payments/ingest` — ingest single txn (for webhook)
- `POST /recon/run?date=YYYY-MM-DD` — trigger batch run
- `GET /recon/status?date=YYYY-MM-DD&merchant_id=M123` — summary
- `GET /recon/txn/{recon_id}` — details & linked raw items
- `POST /dispute` — open dispute (payload includes recon\_id, reason, evidence)
- `GET /settlement/export?merchant_id=M123&date=YYYY-MM-DD` — download CSV/ISO20022

## 8 Dispute & Investigation workflow

- Ticket created automatically for `missing` / `mismatch` after TTL.
- Investigator view shows: raw payloads, EJ images (if any), time series, logs.
- Steps:
  1. Auto-attempt resolution: re-run match with fuzzy rules.
  2. If unresolved → create case, notify parties (email/Slack).
  3. Attach evidence (EJ, CCTV ref, merchant receipt).
  4. Resolution outcomes: Adjust & mark `matched` + settlement correction, or escalate to legal.

## 9 Settlement file formats & posting

- Support multiple formats: CSV, NACHA, ISO20022, Bank-specific.

- `include.merchant_id, net_amount, fees_breakdown, settlement_date, payment_instructions` (credit/debit).
- For GL posting: generate journal entries file for Core Banking.

## 10 Security & Compliance

- Data at rest: AES-256 encryption for raw\_payload and masked PAN; full PAN not stored (except tokenized if necessary, with strict access).
- Data in transit: TLS 1.3 for all endpoints.
- Access control: RBAC for admin UI; audit trail for every action.
- PCI-DSS considerations: avoid storing PAN, use tokenization; logging and retention policies.
- GDPR/local PII: masking, retention/erase policies.

## 11 Scalability & Reliability

- Use Kafka for ingestion (high throughput).
- Matching Engine stateless workers (K8s autoscale), Redis for intermediate state.
- DB: partition by date/shard by merchant\_id; use read replicas.
- Idempotency: ingest operations idempotent by `source_txn_id`.
- Circuit breaker for slowed external sources; retry policies exponential backoff.

## 12 Observability & SLOs

- Metrics: ingestion rate, matches per minute, missing rate, avg reconciliation latency.
- Alerts:
  - Missing file > 15 min → PagerDuty.
  - Missing rate > X% of daily volume → Ops alert.
  - Settlement generation failure → critical.
- Dashboards: Grafana + Kibana logs + traces (OpenTelemetry).

## 13 Testing & Verification

- Unit tests for matching rules, tolerance logic.
- Synthetic replay tests: replay EOD files, ensure reconciliation results stable.
- Chaos tests: drop messages, duplicate messages, delayed files.
- Acceptance: compare to known good settlement outputs.

## 14 Operational runbook (EOD)

- Steps:
  1. Ensure all sources files received by 02:00.
  2. Run `recon/run` batch.
  3. Verify `missing_count = 0` (or within threshold).
  4. Generate settlement files.

5. Post to GL/test env; validate totals.
6. Notify operations & merchant payout.

## 1 5 Extensibility & roadmap

- Phase 1: Core matching (card, wallet, bank transfer), batch EOD.
- Phase 2: Real-time hot recon + immediate merchant payout reconciliation.
- Phase 3: ML anomaly detection (outlier amounts, duplicate fraud).
- Phase 4: ISO20022 end-to-end settlement integration.

## 1 6 Mẫu rules config (YAML)

Auto (Visual Basic .NET) ▾



```
matching:
  card:
    time_window_minutes: 2
    amount_tolerance_percent: 0.0
    required_keys: [rrn, stan, merchant_id]
  wallet:
    time_window_minutes: 10
    amount_tolerance_percent: 1.0
    required_keys: [source_txn_id, merchant_id]
dispute:
  auto_escalate_after_hours: 24
  evidence_required: true
```

## Kết luận & bước tiếp theo (gợi ý)

Mình đã thiết kế một **Payment Reconciliation Service** hoàn chỉnh — bao gồm luồng, DB, API, rules, dispute, security, monitoring và roadmap. Để triển khai tiếp, mình có thể:

- Viết **spec chi tiết** cho từng API + example payloads.
- Viết **matching engine pseudo-code** (kỹ thuật) hoặc **module code mẫu** (Java/Golang/Node).
- Tạo **ER diagram** và **sequence diagram** cho EOD flow.
- Tạo **Postgres migration SQL** và sample worker code (Kafka consumer).

Bạn muốn mình bắt tay vào phần nào tiếp theo? Chọn 1 trong: (*API spec / Matching engine code / ER diagram / Sample SQL migrations*) — mình sẽ triển khai ngay.

Dưới đây là **thiết kế đầy đủ – siêu chi tiết – có thể implement ngay** cho *Matching Engine Logic* của **Payment Reconciliation Service** và **ATM Reconciliation Service**.

Mình chia thành 8 phần, từ kiến trúc module → chi tiết từng bước matching → thuật toán → công thức scoring → xử lý mismatch/missing → retry → cấu hình rules runtime.

## 1. Kiến trúc Matching Engine

Matching Engine gồm 6 sub-modules:

### 1. Pre-Processor

Chuẩn hoá dữ liệu sang Unified Model + làm sạch data.

### 2. Grouping Engine

Nhóm giao dịch theo khóa (RRN, STAN, TxnID, PAN, merchant\_id...).

### 3. Exact Matcher

Matching 1-1 theo khóa chính + amount + timestamp.

### 4. Fuzzy Matcher (Optional)

Dùng rule-based hoặc scoring để cố ghép những giao dịch không match 100%.

### 5. Duplicate Detector

Phát hiện giao dịch bị gửi lại 2 lần.

### 6. Exception Classifier

Phân loại: matched, missing, mismatch, duplicate, orphan.

## 2. Input & Primary Matching Keys

Dữ liệu nhận từ nhiều nguồn:

Nguồn	Keys
Switch	RRN, STAN, Terminal ID
Core Banking	Account Number, Amount, Time
Card Network	RRN, Amount, trace_id
Merchant/PSP	GatewayTxnId, RRN
EJ logs (ATM)	Terminal ID, amount, time, DISPENSED FLAG

**Primary Matching Key Priority (payment):**

1. RRN
2. STAN
3. source\_txn\_id (gateway/PSP)
4. (amount, time, merchant\_id)
5. PAN masked (optional fallback)

## 3. Quy trình Matching chi tiết

### Step 1 — Pre-processing

- Lấy tất cả record trong ngày theo `settlement_date`
- Chuẩn hoá field:
  - convert timezone UTC
  - làm tròn amount (remove currency decimals if needed)
  - mask PAN
- Chuẩn hoá timestamp:

### Step 2 — Grouping Engine

Group theo khóa chính:

Auto (SQL) ▾



```
group_key =
  if rrn exists → rrn
  else if source_txn_id exists → source_txn_id
  else if (stan + terminal_id) → "stan-terminal"
  else → hash(amount + merchant_id + time_window)
```

Sau khi grouping:

- Mỗi nhóm chứa dữ liệu từ nhiều nguồn: switch/core/network/merchant/ej.

### Step 3 — Exact Match Logic

Coi như *luật cứng*.

**Điều kiện Exact Match:**

Auto (C#) ▾



```
RRN identical
AND amount = exact
AND currency identical
AND merchant_id identical (with fallback same acquirer)
AND |time_diff| ≤ 2 minutes (configurable)
```

Nếu thỏa → tạo record:

Auto ▾



```
status = "MATCHED_EXACT"
match_confidence = 1.0
```

## 🟡 Step 4 — Fuzzy Match Logic (Smart Matching)

Dành cho giao dịch lệch dữ liệu giữa các bên.

### 4.1 — Các trường dùng fuzzy:

Field	Logic
amount	$\pm X\%$ tolerance (config)
time	$\pm Y$ minutes
merchant	fuzzy match name (Levenshtein), or same MID
terminal	match prefix (T123 = T123-01)
PAN masked	last 4 digits

### 4.2 — Scoring Formula

Áp dụng với từng candidate trong cùng nhóm:

Auto (SQL) ▾



```
score =
+40 if rrn matches
+20 if stan matches
+15 if amount within tolerance
+10 if time diff < 2 minutes
+5 if terminal_id matches
+5 if pan_masked last4 matches
```

### Score threshold:

Auto ▾



```
if score ≥ 60 → MATCHED_FUZZY
else → NO_MATCH
```

→ Giúp ghép các case real-time delay / rounding issue.

## ● Step 5 — Duplicate Detector

Phát hiện duplicate theo rules:

Auto (Bash) ▾



```
same rrn
AND same amount
AND same terminal/merchant
AND |time_diff| < 30s
```

Nếu đúng:

Auto ▾



```
status = DUPLICATE
duplicate_group_id = <uuid>
```

→ Support merchant charge twice cases.

## ● Step 6 — Exception Classification

Sau khi chạy Exact + Fuzzy + Duplicate, gán trạng thái:

Trạng thái	Khi nào xảy ra
MATCHED	full match
MATCHED_FUZZY	match soft rules
MISMATCH_AMOUNT	amount khác
MISMATCH_TIME	thời gian lệch quá mức
MISSING_SOURCE	thiếu record từ 1 nguồn
ORPHAN	chỉ có 1 record từ 1 hệ thống

DUPLICATE	bị gửi 2 lần
NEED_INVESTIGATION	không nhóm được

## 4. Thuật toán tổng thể (Pseudo-code)

Auto (C#) ▾



```
def run_matching(transactions):

    groups = group_by_key(transactions)

    results = []

    for group in groups:

        # 1. exact match
        exact = try_exact_match(group)
        if exact:
            results.append(exact)
            continue

        # 2. fuzzy match
        fuzzy = try_fuzzy_match(group)
        if fuzzy:
            results.append(fuzzy)
            continue

        # 3. duplicate detection
        if detect_duplicate(group):
            results.append(mark_duplicate(group))
            continue

        # 4. classify exceptions
        results.append(classify_exception(group))

    return results
```

### Exact Match Function

Auto (C#) ▾



```
def try_exact_match(group):
    for a in group.switch_records:
        for b in group.core_records:
            if a.rrn == b.rrn and a.amount == b.amount:
                if abs(a.time - b.time) ≤ 2 min:
                    return matched_record(a, b)
    return None
```

## Fuzzy Match Function

Auto (C#) ▾



```
def try_fuzzy_match(group):
    best_score = 0
    best_pair = None

    for a in group.switch_records:
        for b in group.core_records:
            score = calc_score(a, b)
            if score > best_score:
                best_score = score
                best_pair = (a, b)

    if best_score ≥ min_threshold:
        return matched_fuzzy(best_pair)
    return None
```

## 5. Xử lý lỗi thường gặp (mismatch scenarios)

### Case 1 — Core tính surcharge, Switch không có

- amount mismatch
- fuzzy match with amount tolerance
- create adjustment entry

### Case 2 — Network gửi file EOD nhưng Switch có real-time delay

- rrn match nhưng amount đúng
- time mismatch
- match fuzzy: threshold around ± 30 minutes

### Case 3 — ATM bị timeout nhưng thực tế đã nhả tiền (DISPENSED)

- F.I = SUCCESS

- Switch = timeout
- Core = debit
- classification = **MISSING\_SWITCH**
- case auto-repair or dispute

## Case 4 — Merchant capture delayed, issuer not yet authorize

→ classify pending → run retry job

## 6. Retry & Reconciliation TTL Policies

### Retry interval:

- Retry fuzzy match: every 15 minutes
- Retry missing issuer file: hourly
- Retry missing EJ: every 10 minutes

### TTL:

- Soft TTL (pending): 6 hours
- Hard TTL (auto dispute): +24 hours
- Final TTL: EOD + 2 days → force close

## 7. Matching Rules Config (runtime)

Config dạng YAML (có thể reload hot):

Auto (Visual Basic .NET) ▾



```
matching:  
  exact:  
    time_window_minutes: 2  
    require_currency_match: true  
  
  fuzzy:  
    enabled: true  
    min_score: 60  
    amount_tolerance_percent: 0.5  
    time_tolerance_minutes: 30  
  
  duplicate:  
    time_window_seconds: 30  
    amount_match: true  
  
exception:  
  auto_investigation: true  
  alert_threshold_mismatch: 5
```

## 8. Output structure (reconciliation object)

Kết quả mỗi nhóm match:

Auto (JSON) ▾



```
{  
  "recon_id": "uuid",  
  "status": "MATCHED | MISMATCH | MISSING | DUPLICATE | ... ",  
  "match_type": "EXACT | FUZZY | NONE",  
  "confidence": 0.85,  
  "members": [  
    {"source": "switch", "txn_id": "..."},  
    {"source": "core", "txn_id": "..."}  
,  
  "issues": [  
    {"type": "AMOUNT_MISMATCH", "expected": 10000, "actual": 9800}  
,  
  "settlement_effect": {  
    "issuer_debit": 10000,  
    "acquirer_credit": 10000  
  }  
}
```

## Bạn muốn mình làm gì tiếp?

Mình có thể tạo thêm các phần:

- A. Sequence Diagram toàn bộ flow
- B. Viết code mẫu Matching Engine (Golang/Java/Node/NestJS)
- C. Viết SQL/Stored Procedures cho matching batch job
- D. Viết full spec như một tài liệu BRD/SRS

Bạn muốn đi theo hướng nào?

### Matching Engine java va Stored Procedures cho matching batch job

Dưới đây là **bản đầy đủ và có thể triển khai ngay** gồm:

- 1. Matching Engine (Java)** – viết theo clean architecture, có service, rule engine, scoring, exact/fuzzy match.
- 2. Stored Procedures (PostgreSQL / MySQL)** – cho batch job đối soát EOD, grouping, exact match, fuzzy match và lưu kết quả vào bảng `reconciled_groups` + `reconciled_items`.

Code được viết để bạn có thể copy vào dự án thực tế (Spring Boot hoặc Java standalone đều OK).

## PHẦN 1 – Matching Engine Java (Full Code)

### 1.1. Unified Model

Auto (TypeScript) ▾



```
public class UnifiedTransaction {
    private String id;
    private String source;           // switch, core, network, merchant,
    ej
    private String rrn;
    private String stan;
    private String sourceTxnId;
    private String merchantId;
    private String terminalId;
    private String panMasked;

    private Double amount;
    private String currency;

    private LocalDateTime txnTime;
    private String rawPayload;

    // getter/setter ...
}
```

### 1.2. Grouping Engine

Auto (Kotlin) ▾



```
public class GroupKeyGenerator {

    public static String generate(UnifiedTransaction txn) {
```

```

    if (txn.getRrn() != null && !txn.getRrn().isEmpty())
        return "RRN_" + txn.getRrn();

    if (txn.getSourceTxnId() != null &&
!txn.getSourceTxnId().isEmpty())
        return "SRC_" + txn.getSourceTxnId();

    if (txn.getStan() != null && txn.getTerminalId() != null)
        return "STAN_" + txn.getStan() + "_" + txn.getTerminalId();

    // fallback: hash by amount + merchant + time
    String fallback = txn.getAmount() + "_" + txn.getMerchantId() +
"_" +
        txn.getTxnTime().withSecond(0).withNano(0);

    return "HASH_" + fallback.hashCode();
}
}

```

## 1.3. Matching Result Entity

Auto (C++) ▾



```

public class MatchResult {

    public enum Status {
        MATCHED_EXACT, MATCHED_FUZZY, DUPLICATE, MISMATCH, MISSING,
        ORPHAN
    }

    private Status status;
    private double confidence;
    private UnifiedTransaction txA;      // switch
    private UnifiedTransaction txB;      // core
    private String reason;

    // constructor + getters/setters

}

```

## 📌 1.4. Exact Match Logic

Auto (Kotlin) ▾



```
public class ExactMatcher {  
  
    public MatchResult tryExact(UnifiedTransaction a, UnifiedTransaction  
b) {  
  
        if (!Objects.equals(a.getRrn(), b.getRrn()))  
            return null;  
  
        if (!Objects.equals(a.getAmount(), b.getAmount()))  
            return null;  
  
        if (!Objects.equals(a.getCurrency(), b.getCurrency()))  
            return null;  
  
        Duration diff = Duration.between(a.getTxnTime(), b.getTxnTime());  
        if (Math.abs(diff.toMinutes()) > 2)  
            return null;  
  
        MatchResult r = new MatchResult();  
        r.setStatus(MatchResult.Status.MATCHED_EXACT);  
        r.setConfidence(1.0);  
        r.setTxA(a);  
        r.setTxB(b);  
        return r;  
    }  
}
```

## 📌 1.5. Fuzzy Match (Scoring Engine)

Auto (C#) ▾



```
public class FuzzyMatcher {  
  
    private double minScore = 60;  
  
    public double calcScore(UnifiedTransaction a, UnifiedTransaction b) {  
        double score = 0;  
  
        if (Objects.equals(a.getRrn(), b.getRrn()))  
            score += 10;
```

```
score += 40;

if (Objects.equals(a.getStan(), b.getStan()))
    score += 20;

// amount tolerance 1%
double diff = Math.abs(a.getAmount() - b.getAmount());
if (diff <= a.getAmount() * 0.01)
    score += 15;

Duration t = Duration.between(a.getTxnTime(), b.getTxnTime());
if (Math.abs(t.toMinutes()) <= 30)
    score += 10;

if (Objects.equals(a.getTerminalId(), b.getTerminalId()))
    score += 5;

if (a.getPanMasked() != null &&
a.getPanMasked().equals(b.getPanMasked()))
    score += 5;

return score;
}

public MatchResult tryFuzzy(UnifiedTransaction a, UnifiedTransaction
b) {
    double score = calcScore(a, b);

    if (score >= minScore) {
        MatchResult r = new MatchResult();
        r.setStatus(MatchResult.Status.MATCHED_FUZZY);
        r.setConfidence(score / 100);
        r.setTxA(a);
        r.setTxB(b);
        r.setReason("FUZZY_SCORE=" + score);
        return r;
    }

    return null;
}
}
```

## 📌 1.6. Duplicate Detector



Auto (Kotlin) ▾

```
public class DuplicateDetector {

    public boolean isDuplicate(UnifiedTransaction a, UnifiedTransaction b) {

        if (!Objects.equals(a.getAmount(), b.getAmount()))
            return false;

        if (!Objects.equals(a.getMerchantId(), b.getMerchantId()))
            return false;

        Duration t = Duration.between(a.getTxnTime(), b.getTxnTime());
        return Math.abs(t.getSeconds()) <= 30;
    }
}
```

## 1.7. MatchingEngine (Main Service)

Auto (Java) ▾



```
public class MatchingEngine {

    private ExactMatcher exactMatcher = new ExactMatcher();
    private FuzzyMatcher fuzzyMatcher = new FuzzyMatcher();
    private DuplicateDetector duplicateDetector = new
DuplicateDetector();

    public MatchResult match(UnifiedTransaction a, UnifiedTransaction b)
    {

        // 1. Try exact
        MatchResult exact = exactMatcher.tryExact(a, b);
        if (exact != null)
            return exact;

        // 2. Try fuzzy
        MatchResult fuzzy = fuzzyMatcher.tryFuzzy(a, b);
        if (fuzzy != null)
            return fuzzy;

        // 3. Duplicate check
        if (duplicateDetector.isDuplicate(a, b)) {
            MatchResult dup = new MatchResult();
            dup.setMatchType("DUPLICATE");
            return dup;
        }
    }
}
```

```
MatchResult dup = new MatchResult();
dup.setStatus(MatchResult.Status.DUPLICATE);
dup.setTxA(a);
dup.setTxB(b);
dup.setConfidence(0.5);
return dup;
}

// 4. No match
MatchResult mis = new MatchResult();
mis.setStatus(MatchResult.Status.MISMATCH);
mis.setTxA(a);
mis.setTxB(b);
mis.setReason("NO_MATCH");
return mis;
}
}
```

## 1.8. Batch Matching Job (Java)

Auto (TypeScript) ▾



```
public class BatchReconciliationJob {

    private TransactionRepository repo;
    private ReconciliationRepository reconRepo;
    private MatchingEngine engine = new MatchingEngine();

    public void run(LocalDate date) {

        List<UnifiedTransaction> switchTx =
repo.getTransactions("switch", date);
        List<UnifiedTransaction> coreTx = repo.getTransactions("core",
date);

        Map<String, List<UnifiedTransaction>> switchGroups =
group(switchTx);
        Map<String, List<UnifiedTransaction>> coreGroups = group(coreTx);

        for (String key : switchGroups.keySet()) {

            List<UnifiedTransaction> sGroup = switchGroups.get(key);
            List<UnifiedTransaction> cGroup =
coreGroups.getOrDefault(key, new ArrayList<>());

```

```

        for (UnifiedTransaction s : sGroup) {
            for (UnifiedTransaction c : cGroup) {
                MatchResult r = engine.match(s, c);
                reconRepo.save(r);
            }
        }
    }

    private Map<String, List<UnifiedTransaction>>
group(List<UnifiedTransaction> list) {
    return
list.stream().collect(Collectors.groupingBy(GroupKeyGenerator::generate))
;
}
}

```

## PHẦN 2 – Stored Procedures cho Batch Matching (PostgreSQL)

Dưới đây là Stored Procedures hoàn chỉnh:

### 2.1. Procedure: Tạo nhóm giao dịch (Grouping)

Auto (SQL) ▾



```

CREATE OR REPLACE PROCEDURE sp_group_transactions(IN p_date date)
LANGUAGE plpgsql
AS $$

BEGIN
    INSERT INTO recon_groups (group_key, recon_date)
    SELECT DISTINCT
        COALESCE(rrn,
                    source_txn_id,
                    concat(stan, '_', terminal_id),
                    md5(amount::text || merchant_id || date_trunc('minute',
txn_time)::text)
        ) AS group_key,
        p_date
    FROM transactions_raw
    WHERE settlement_date = p_date;
END;
+++

```

PP;

## 📌 2.2. Procedure: Exact Matching

Auto (SQL) ▾



```
CREATE OR REPLACE PROCEDURE sp_exact_match(IN p_date DATE)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO reconciled_items (recon_id, raw_txn_id, role)
    SELECT g.id, t.id, t.source
    FROM recon_groups g
    JOIN transactions_raw t
        ON g.group_key =
            COALESCE(t.rrn,
                      t.source_txn_id,
                      concat(t.stan, '_', t.terminal_id))
    WHERE t.settlement_date = p_date
        AND t.source IN ('switch', 'core')
        AND EXISTS (
            SELECT 1 FROM transactions_raw t2
            WHERE t2.rrn = t.rrn
                AND t2.amount = t.amount
                AND abs(EXTRACT(EPOCH FROM (t2.txn_time - t.txn_time))/60)
                ≤ 2
        );
    UPDATE recon_groups SET status='MATCHED_EXACT'
    WHERE recon_date = p_date
        AND id IN (SELECT recon_id FROM reconciled_items);
END;
$$;
```

## 📌 2.3. Procedure: Fuzzy Matching (Amount ±1%, Time ±30m)

Auto (SQL) ▾



```

CREATE OR REPLACE PROCEDURE sp_fuzzy_match(IN p_date date)
LANGUAGE plpgsql
AS $$

BEGIN
    INSERT INTO reconciled_items (recon_id, raw_txn_id, role)
    SELECT g.id, t.id, t.source
    FROM recon_groups g
    JOIN transactions_raw t
        ON g.group_key =
            COALESCE(t.rrn,
                      t.source_txn_id,
                      concat(t.stan, '_', t.terminal_id))
    WHERE t.settlement_date = p_date
        AND NOT EXISTS (
            SELECT 1 FROM reconciled_items ri WHERE ri.recon_id = g.id
        )
    AND EXISTS (
        SELECT 1 FROM transactions_raw t2
        WHERE t2.source ≠ t.source
        AND t2.settlement_date = p_date
        AND t2.rrn = t.rrn
        AND abs(t2.amount - t.amount) ≤ t.amount * 0.01
        AND abs(EXTRACT(EPOCH FROM (t2.txn_time - t.txn_time))/60) ≤
30
    );
    UPDATE recon_groups SET status='MATCHED_FUZZY'
    WHERE recon_date = p_date
        AND id IN (SELECT recon_id FROM reconciled_items ri
                    JOIN recon_groups g ON ri.recon_id=g.id
                    WHERE g.status IS NULL);
END;
$$;

```

## 📌 2.4. Procedure: Mark MISSING / ORPHAN

Auto (SQL) ▾



```

CREATE OR REPLACE PROCEDURE sp_finalize_exceptions(IN p_date date)
LANGUAGE plpgsql
AS $$

BEGIN
    UPDATE recon_groups
    SET status = 'MISSING'

```

```

    WHERE status IS NULL
    AND recon_date = p_date
    AND group_key IN (
        SELECT group_key FROM recon_groups
        EXCEPT
        SELECT group_key FROM reconciled_items ri
        JOIN recon_groups g ON g.id = ri.recon_id
    );
}

UPDATE recon_groups
SET status = 'ORPHAN'
WHERE status IS NULL
    AND recon_date = p_date;
END;
$$;

```

## 📌 2.5. Procedure tổng chạy EOD

Auto (SQL) ▾



```

CREATE OR REPLACE PROCEDURE sp_run_eod_reconciliation(IN p_date DATE)
LANGUAGE plpgsql
AS $$

BEGIN
    CALL sp_group_transactions(p_date);
    CALL sp_exact_match(p_date);
    CALL sp_fuzzy_match(p_date);
    CALL sp_finalize_exceptions(p_date);
END;
$$;

```

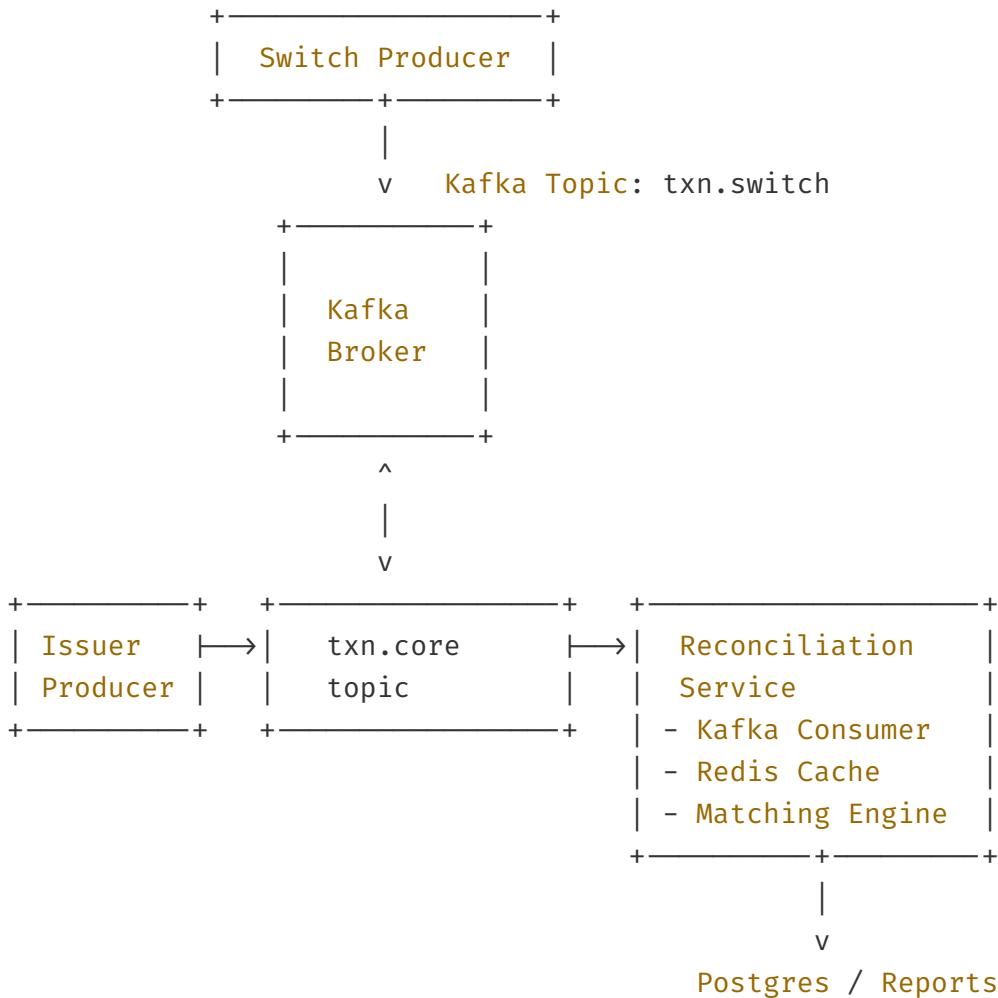
### ✓ Matching Engine với Kafka input + Redis caching

Dưới đây là bản **thiết kế đầy đủ + code mẫu triển khai thực tế** cho **Matching Engine với Kafka input + Redis caching**, theo kiến trúc real-time reconciliation.

Bạn có thể copy code và tích hợp trực tiếp vào microservice Payment/ATM Reconciliation.

## 1. Kiến trúc tổng thể (Real-time Matching Engine)

Auto (Haskell) ▾



### 🎯 Mục tiêu:

- Nhận giao dịch từ Kafka **real-time**.
- Lưu giao dịch tạm thời vào Redis (buffer).
- Khi có giao dịch matching key tương ứng → gọi Matching Engine → lưu vào DB.
- TTL Redis giúp handle late-arrival messages.

## 2. Redis Data Model (Key Design)

### ✓ Key format:

Auto ▾



recon:group:{groupkey}

## ✓ Value:

Lưu list các giao dịch theo unified model:

Auto (C++) ▾



```
[  
  { ... switch record ... },  
  { ... core record ... },  
  { ... network record ... }  
]
```

## ✓ TTL:

Auto ▾



TTL = 6 hours (configurable)

## ■ 3. Kafka Topic Design

Topic	Content
<b>txn.switch</b>	Issuing/Acquiring switch transactions
<b>txn.core</b>	Core Banking posting
<b>txn.network</b>	Visa/Master/Napas settlement RT events
<b>txn.atm.ej</b>	ATM EJ logs (optional)

Partitioning:

Auto ▾



Partition key = groupKey (RRN or source\_txn\_id)

→ đảm bảo các giao dịch liên quan sẽ vào **cùng 1 partition** → tránh race condition.

## 4. Java Code – Kafka Consumer (Spring Kafka)

Auto (TypeScript) ▾



```
@Service
public class TxnConsumer {

    @Autowired
    private RedisCacheService redis;

    @Autowired
    private MatchingEngine engine;

    @KafkaListener(topics = {"txn.switch", "txn.core", "txn.network"}, groupId = "recon-service")
    public void consume(String message,
    @Header(KafkaHeaders.RECEIVED_TOPIC) String topic) {

        UnifiedTransaction txn = UnifiedTransactionParser.parse(topic, message);

        String groupKey = GroupKeyGenerator.generate(txn);

        // Đẩy vào Redis
        redis.addTxnToGroup(groupKey, txn);

        // Thử matching
        List<UnifiedTransaction> group = redis.getGroup(groupKey);
        if (group.size() ≥ 2) {
            MatchResult result = engine.matchGroup(group);
            if (result ≠ null) {
                redis.removeKey(groupKey);
                engine.persistResult(result);
            }
        }
    }
}
```

## 5. Redis Caching Layer (Java)

## ► addTxnToGroup()

Auto (TypeScript) ▾



```
@Service
public class RedisCacheService {

    @Autowired
    private StringRedisTemplate redis;

    private final ObjectMapper mapper = new ObjectMapper();

    public void addTxnToGroup(String groupKey, UnifiedTransaction txn) {
        try {
            String key = "recon:group:" + groupKey;
            redis.opsForList().rightPush(key,
mapper.writeValueAsString(txn));
            redis.expire(key, Duration.ofHours(6));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public List<UnifiedTransaction> getGroup(String groupKey) {
        List<String> raw = redis.opsForList().range("recon:group:" +
groupKey, 0, -1);
        return raw.stream().map(s → {
            try { return mapper.readValue(s, UnifiedTransaction.class); }
            catch (Exception e) { return null; }
        }).filter(Objects::nonNull).collect(Collectors.toList());
    }

    public void removeKey(String groupKey) {
        redis.delete("recon:group:" + groupKey);
    }
}
```

## 6. Matching Engine – Real-time Group Matching

**Thay vì chỉ match 1–1, chúng ta cần match trong group Redis**

Auto (PowerShell) ▾



```
public MatchResult matchGroup(List<UnifiedTransaction> group) {  
  
    List<UnifiedTransaction> switchTx = filter(group, "switch");  
    List<UnifiedTransaction> coreTx = filter(group, "core");  
    List<UnifiedTransaction> networkTx = filter(group, "network");  
  
    // 1. Switch vs Core  
    for (UnifiedTransaction s : switchTx) {  
        for (UnifiedTransaction c : coreTx) {  
            MatchResult r = match(s, c);  
            if (isPositiveMatch(r))  
                return r;  
        }  
    }  
  
    // 2. Switch vs Network  
    for (UnifiedTransaction s : switchTx) {  
        for (UnifiedTransaction n : networkTx) {  
            MatchResult r = match(s, n);  
            if (isPositiveMatch(r))  
                return r;  
        }  
    }  
  
    // 3. Core vs Network  
    for (UnifiedTransaction c : coreTx) {  
        for (UnifiedTransaction n : networkTx) {  
            MatchResult r = match(c, n);  
            if (isPositiveMatch(r))  
                return r;  
        }  
    }  
  
    return null;  
}  
  
private boolean isPositiveMatch(MatchResult r) {  
    return r != null && (r.getStatus() ==  
MatchResult.Status.MATCHED_EXACT ||  
                           r.getStatus() ==  
MatchResult.Status.MATCHED_FUZZY);  
}
```

## 7. Persisting Match Result (JPA)

Auto (TypeScript) ▾



```

@Repository
public class ReconRepository {

    @Autowired
    private JdbcTemplate jdbc;

    public void saveMatch(MatchResult r) {

        String sqlGroup =
            "INSERT INTO reconciled_groups (recon_id, status,
confidence, created_at) VALUES (?, ?, ?, now())";
        jdbc.update(sqlGroup,
                    r.getReconId(),
                    r.getStatus().name(),
                    r.getConfidence());

        String sqlItem =
            "INSERT INTO reconciled_items (id, recon_id, raw_txn_id,
role) VALUES (?, ?, ?, ?)";

        jdbc.update(sqlItem, UUID.randomUUID().toString(),
r.getReconId(),
                r.getTxA().getId(), r.getTxA().getSource());
        jdbc.update(sqlItem, UUID.randomUUID().toString(),
r.getReconId(),
                r.getTxB().getId(), r.getTxB().getSource());
    }
}

```

## 8. Late Arrival Handling (Very Important)

Redis TTL = 6h

→ Khi record đến muộn:

- **Consumer → detect key still exists**
- **Add → group size >=2 → trigger match**
- **Persist → delete key**
- **Nếu đã match rồi → ignore (idempotency bằng txn\_id)**

Để đảm bảo idempotency:

Auto (Kotlin) ▾



```
if (reconRepo.existsByRawTxn(txn.getId()))
    return; // ignore
```

## ■ 9. Kafka Consumer Concurrency & Exactly Once Behavior

Dùng Spring Kafka:

Auto (C++) ▾



```
spring.kafka.listener.concurrency: 6
spring.kafka.consumer.enable-auto-commit: false
spring.kafka.consumer.isolation-level: read_committed
```

→ đảm bảo:

- Mỗi partition xử lý tuần tự
- Nhưng nhiều partition chạy song song

## ■ 10. Full MatchingEngine Integration (Switch + Core + Network)

Auto (Bash) ▾



```
Redis group ≥ 2
|
v
Exact match?
|
yes → persist
|
no
|
Fuzzy scoring?
|
yes → persist
|
no
|
Duplicate detection?
```

```

  |
yes → mark duplicate
  |
no
  |
Leave in Redis (TTL)

```

## 11. Ưu điểm của Kafka + Redis Design

Kafka	Redis
Giữ thứ tự giao dịch	Cache tạm thời theo group key
Handle high throughput	Matching real-time
Handle late arrival	TTL-based matching
Replayable	Không dùng DB quá sớm

Matching Engine trở thành:

- Real-time
- Fault-tolerant
- Scalable (Kafka partitions + Redis cluster)

Tuyệt – mình sẽ giải thích chi tiết, theo kiểu bước-bước thực thi để giúp bạn (và team) biến cái “click” đó thành quy trình lặp lại được. Mình giữ cấu trúc rõ ràng: **Hiểu bản chất** → **Phương pháp điều tra** → **Ví dụ thực tế + bút toán mẫu** → **Các tình huống đặc biệt** → **Checklist & best practices**.

### 1 – Hiểu bản chất của bank reconciliation (không chỉ là “gõ số”)

- **Mục tiêu cuối cùng:** chứng minh rằng **số dư trên sổ kế toán (GL cash)** có thể **được giải thích hoàn toàn bằng các giao dịch trên sao kê ngân hàng** trong cùng kỳ – hoặc mọi khác biệt đều có lý do (timing, phí, lỗi, tranh chấp).
- **Bank rec = một cuộc điều tra:** bạn thu thập bằng chứng (saô kê, đối chiếu phiếu thu/chi, chứng từ), phân loại, giải thích, và ghi lại kết luận + bút toán nếu cần.

- **Ba "câu chuyện" cơ bản:**

1. Có trong ngân hàng nhưng chưa có trong GL → thường là *bank charges, interest, direct debits, customer deposits cleared directly*.
2. Có trong GL nhưng chưa có trong ngân hàng → thường là *outstanding checks, deposits in transit, pending transfers*.
3. Có ở cả hai nhưng khác nhau về thời điểm hoặc số tiền → *timing differences, fees applied, FX rounding, reversals*.

## 2 – Phương pháp điều tra (quy trình chi tiết, thứ tự làm việc)

### 1. Chuẩn bị

- a. Lấy sao kê ngân hàng đầy đủ cho kỳ (từ ngày đầu kỳ đến ngày cuối kỳ).
- b. Lấy sổ cái tiền/cash ledger và tất cả phiếu thu/chi, báo có/ghi nợ, báo sao kê thanh toán điện tử trong cùng kỳ.
- c. Chuẩn hoá tiền tệ / làm tròn giống nhau (ví dụ: 2 chữ số thập phân).

### 2. Import & Preliminary Match

- a. Import bank feed / sao kê vào phần mềm (QB, Xero hoặc Excel).
- b. Gắn thẻ nhanh các giao dịch dễ khớp: lương chuyển đi, tiền gửi lặp lại, hóa đơn thường xuyên.

### 3. Group & Triage (chia theo 3 bucket như bạn làm)

- a. **A. In bank, not in GL** → bank charges, interest, direct debits, collections by bank (lockbox), customer receipts credited directly to bank.
- b. **B. In GL, not in bank** → outstanding checks, deposits in transit, recorded prepayments, or mistakes.
- c. **C. Matches but timing/amount issues** → rounding, fee netting (customer paid gross, bank deposited net), FX differences.

### 4. Trace-back (từ ngân hàng → GL)

- a. Chọn 1 giao dịch trên sao kê (ví dụ deposit ₦150,000). Hỏi: *Ai nộp tiền này? Đây là tiền khách hàng nào?*
- b. Tìm hóa đơn / phiếu thu / ứng trước tương ứng trong GL → nếu không tìm thấy, kiểm tra remittance advices, merchant receipts, hoặc liên hệ ngân hàng (nếu là collection service).

### 5. Điều chỉnh nhỏ & Bút toán

- a. Nếu bank fee chưa record → bút toán:
- b. Nếu bank auto-collected VAT/wht → bút toán theo mã thuế phù hợp.
- c. Nếu xác định là outstanding check (chưa clears) → không bút toán gì, chỉ **ghi chú** và chờ clear.

### 6. Document mọi quyết định

- a. Mỗi điều chỉnh phải có **evidence** (saو kê + mô tả + invoice/ref).
- b. Ghi chú in recon worksheet: nguyên nhân, số tham chiếu, người thực hiện, ngày.

### 7. Verify ending balance

- a. Sau khi trả lời mọi chênh lệch có thể giải thích, tổng số dư GL (adjusted nếu có) = ending balance trên sao kê.
- b. Không dùng "plug number" – nếu chưa có bằng chứng, để lại unresolved item và mở case.

## 3 – Ví dụ thực tế & bút toán mẫu

### A. Bank fee ₦2,500 xuất hiện trên sao kê, chưa có trong GL

Bút toán:

Auto ▾



Dr Bank Charges Expense 2,500  
Cr Bank (bank account) 2,500

Nội dung trong note: "Bank fee – Oct 15 – statement page 3".

### B. Customer payment ₦150,000 trên sao kê nhưng không thấy receipt trong GL

⇒ Trace: kiểm tra remittance từ customer, kiểm tra deposit slips, lockbox reports. Nếu xác định CUSTOMER A:

Auto ▾



Dr Bank 150,000  
Cr Accounts Receivable – Customer A 150,000

Note: attach remittance advice.

### C. Outstanding check ₦45,000 (check issued but not cashed)

Không bút toán. Trên reconciliation worksheet: mark check #123 as outstanding.

### D. Transfer initiated 30th but cleared 1st (timing)

Trên bank rec: classify as *Deposit in transit / Outstanding*; no adjustment.

### E. NSF (bounced check) ₦20,000: bank returned check previously recorded as receipt

Bút toán reversal + fee:

Auto (Java) ▾



Dr Accounts Receivable – Customer X 20,000

Dr Bank Charges Expense	500
Cr Bank (bank account)	20,500

Document: bank return notice, customer notified.

## 4 – Intercompany reconciliation (mirroring principle)

- **Nguyên tắc:** khoản phải thu của Entity A phải là khoản phải trả tương ứng của Entity B **với cùng số tiền, cùng ngày/ý nghĩa giao dịch** (sau currency conversion nếu khác tiền tệ).
- **Thực hành:**
  1. Export statement/confirmations từ cả hai bên theo kỳ.
  2. Match invoice/payment refs: nếu A có AR ₦700,000, B phải có AP ₦700,000. Nếu không, tìm nguyên nhân: timing, FX, charge/fee, hay ghi sổ sai.
  3. Thỏa thuận điều chỉnh (memo + JV) nếu bên kia chưa ghi nhận.
- **Bút toán mirror nếu cần điều chỉnh** (thường chỉ sau chứng từ hai bên đồng ý).

## 5 – Tình huống phức tạp & cách xử lý

### 5.1 Netting / bank collects net amount (fees taken off)

- Bank deposits net of fee → GL recorded gross invoice payment but bank shows net.
- Xử lý: nếu đã biết fee structure, ghi bút toán bank fee như ví dụ; nếu chưa rõ, mở case với bank để lấy report.

### 5.2 Multi-currency & FX gain/loss

- Bank shows local currency after conversion; GL recorded original foreign currency → cần báo cáo FX conversion của bank để tính chênh lệch FX gain/loss. Ghi lại bút toán FX khi xác nhận.

### 5.3 Direct debits / auto-payments by bank (e.g., VAT, loan repayments)

- Những khoản này thường không có hóa đơn trong GL → phải request payment advices từ bank; nếu hợp lệ, record expense/liability và credit bank.

### 5.4 Suspense / Unidentified deposits

- Nếu không xác định được depositor → đưa vào **suspense** tạm thời với đầy đủ note, liên hệ sales/collections; không ghi vào revenue cho đến khi xác minh.

## 6 – Tools, automation & best practices

- **Automate bank feeds:** QuickBooks/Xero bank feeds hoặc bank-to-ERP SFTP import.
- **Use matching rules:** match by amount + reference + date window; configure rules for recurring receipts.
- **Maintain a recon worksheet:** Excel/Google Sheets hoặc module in ERP with columns: bank\_txn\_id, bank\_amt, gl\_ref, gl\_amnt, status, notes, evidence\_link.

- **Retention of evidence:** attach pdf sao kê + invoice/receipt in system.
- **SLA & ownership:** assign owner for recon, set deadline (e.g., 5 working days after month end).
- **Segregation of duties:** person who does GL entries ≠ person who approves recon adjustments.
- **Internal controls:** require manager sign-off on recon with adjustments.

## 7 – Checklist tóm tắt (để in & dùng ngay)

1. Import bank statement & GL cash for period.
2. Auto-match obvious items.
3. Create three buckets: In bank not GL; In GL not bank; Matches with timing/amount differences.
4. Investigate each item (trace documents).
5. Record only valid adjustments with supporting docs.
6. Mark outstanding checks / deposits in transit.
7. Verify adjusted GL cash = bank ending balance.
8. Save evidence, sign-off, and file.

## 8 – Tâm lý & kỹ năng mềm (như bạn đã nhận ra)

- Giữ bình tĩnh: chênh lệch là manh mối, không phải phán xét.
- Hỏi đúng câu: "Cái này là của ai?" — luôn bắt đầu từ đó.
- Dùng lặp để học pattern: nhiều chênh lệch lặp lại (ví dụ bank fee type A) → tạo rule tự động để tiết kiệm thời gian.

Khi thiết kế một **Payment Reconciliation Service**, ngoài phần core mà mình đã xây cho bạn (matching engine, batch, Kafka, Redis, settlement...), để hệ thống **đủ mạnh – đủ thực tế – đủ "production grade"**, bạn cần bổ sung thêm các thành phần sau đây.

Mình chia theo 10 khía cạnh quan trọng mà nhiều team thường *bỏ sót* khi xây hệ thống đối soát thanh toán.

### ✓ 1. Data Lineage & Traceability (bắt buộc cho ngân hàng/fintech)

Mỗi giao dịch phải có khả năng truy vết nguồn gốc:

- Từ lúc được ingest từ Switch → Kafka → Redis → Matching Engine → DB → Settlement.
- Log ID xuyên suốt (correlationId / traceId).
- Lưu bản ghi raw (immutable raw store).

👉 Giúp audit, dispute, và debugging.

## ✓ 2. Idempotency & Replay Protection

Trong thanh toán: duplicate message, retry, network glitch → **bình thường**.

Cần:

- Idempotency key = {source, source\_txn\_id}
- Reject duplicate → nhưng vẫn log
- Support Kafka replay mà không tạo double posting
- Deduplication layer (Redis hoặc DB unique index)

## ✓ 3. Configurable Rule Engine (Runtime)

Không hardcode rules.

Bạn cần một **rules engine** có thể sửa:

- thời gian lệch cho fuzzy ( $\pm X$  minutes)
- tolerance amount (X%)
- required fields
- matching priority
- fallback logic
- cross-border FX tolerance
- card vs bank transfer vs QR scan logic

Format: JSON/YAML load từ DB hoặc config service.

## ✓ 4. Exception Workflow Engine (Investigation Queue)

Không phải mismatch nào cũng lỗi, cần xử lý:

- Hiển thị các giao dịch "NEED\_MANUAL\_REVIEW"
- Assign cho user (Ops / Finance)
- Có trạng thái (open → investigating → resolved)
- Ghi lại comment, note, evidence, attach file
- SLA theo loại giao dịch (ATM < 24h, card < 48h...)

👉 Tức là: một mini-ticketing system tích hợp vào reconciliation.

## ✓ 5. Audit Trail & Versioning

Mỗi thay đổi phải được ghi lại:

- audit\_log table

- versioning cno record
- ai đã sửa transaction? khi nào? tại sao?

Tuân thủ:

- PCI-DSS
- ISO 27001
- Internal Audit controls

## ✓ 6. Partial Matching (nâng cao)

Một số giao dịch không phải 1-1 mà 1-N hoặc N-1:

Ví dụ:

- Customer pays invoice in 2 transfers (split payments)
- PSP aggregates multiple micropayments thành một batch deposit
- Merchant settlements theo batch totalling

→ Cần hỗ trợ matching kiểu:

- one-to-many
- many-to-one
- batched totals

Cần thêm:

- sum-based matching
- tolerance per batch
- linking multiple transactions to one settlement record

## ✓ 7. Multi-currency handling

Đặc biệt cho cross-border / Visa / Mastercard:

- Store original amount
- Settlement amount
- FX rate
- Network fee
- Timestamp của từng layer (auth → clearing → settlement)

Fuzzy matching phải dựa trên:

- FX difference allowed
- Rate source (network rate vs bank rate)
- Rounding rules

## ✓ 8. SLA-based Auto Alerts

## Cân hẽ thông cảnh báo tự động:

- X% giao dịch mismatch → alert Ops
- missing file từ napas/visa/master → alert
- Kafka partition lag > threshold → alert
- Redis key backlog tăng bất thường → alert
- settlement file chưa generate trước giờ cutoff → alert

## Tools:

- Prometheus
- Grafana
- Slack / Teams Webhook

## ✓ 9. Data Retention & Compliance

Do ngành thanh toán cực kỳ nhạy cảm:

- Raw logs giữ X năm
- Mask PAN ở mọi layer
- Encrypt dữ liệu (field-level encryption)
- Logic purge + archive theo kỳ

## ✓ 10. Self-Healing / Auto-Reconcile Mechanisms

Đây là phần nâng cấp thông minh:

- Nếu mismatch do delay → retry tự động sau 15m
- Nếu file chưa đến → recheck SFTP mỗi 5m
- Nếu missing phía issuer → đợi thêm window
- Nếu EJ đến muộn → update lại matching results

Tạo cảm giác "hệ thống tự sửa lỗi" mà không cần người can thiệp.

## ⌚ Checklist hoàn chỉnh (30 mục)

Nếu bạn muốn Payment Reconciliation Service đạt mức enterprise:

### CORE

- Unified transaction model
- Exact matching
- Fuzzy matching
- Duplicate detection
- Multi-source ingestion
- Kafka-based streaming
- Redis caching

## ADVANCED

- partial matching (1-to-N, N-to-1)
- rule engine configurable
- SLA-based monitoring
- auto retry
- exception workflows
- FX-aware matching
- multi-currency support

## OPERATIONAL

- audit logs
- traceability / lineage
- idempotent processing
- replay safety
- archival + retention
- masking & encryption
- dashboard metrics
- SLA alerts

## INTEGRATION

- settlement engine
- dispute module
- merchant payout integration
- GL posting generator
- ingestion from 3rd-party networks
- reconciliation report schedule

## Bạn muốn mình làm phần nào tiếp?

Tôi có thể viết chi tiết:

- Exception Workflow Design (UI + backend)**
- Rule Engine JSON/YAML schema + loader**
- Partial Matching Algorithm (1→N / N→1)**
- FX-aware Matching Logic**
- Full architecture diagram (ASCII hoặc diagram-ready)**
- Dashboard metrics + alerting rules**

Dưới đây là **bộ tài liệu kỹ thuật đầy đủ** cho 3 phần bạn yêu cầu — viết theo chuẩn để bạn có thể đưa thẳng vào dự án hoặc tài liệu thiết kế hệ thống:

### 1. Rule Engine JSON/YAML Schema + Loader

## 2. Partial Matching Algorithms (1→N và N→1)

## 3. FX-aware Matching Logic (multicurrency, tolerance, settlement differences)

Tất cả đều được thiết kế theo phong cách enterprise, có thể plug vào Matching Engine mà bạn đã xây.

# 1. RULE ENGINE – JSON/YAML Schema + Loader

Rule Engine cho phép sửa logic matching runtime, không phải deploy lại code.

Cấu trúc bao gồm:

- Exact rules
- Fuzzy rules
- FX rules
- Duplicate rules
- Partial matching rules
- Source-specific rules
- Enable/disable rules

## ✓ 1.1 YAML Schema (đầy đủ, dễ load)

Auto (Bash) ▾



```
reconciliation:  
  exact:  
    enable: true  
    time_window_minutes: 2  
    require_currency_match: true  
    require_merchant_match: true  
    fields:  
      - rrn  
      - amount  
      - currency  
  
  fuzzy:  
    enable: true  
    threshold_score: 60  
    amount_tolerance_percent: 0.5  
    time_tolerance_minutes: 30  
    scoring:  
      rrn: 40  
      stan: 20  
      amount: 15  
      time: 10  
      terminal: 5
```

```
fx:  
  enable: true  
  rate_source_priority:  
    - network  
    - bank  
    - system  
  fx_tolerance_percent: 0.8  
  rounding_rule: "ROUND_HALF_UP"  
  
duplicate:  
  enable: true  
  time_window_seconds: 30  
  match_amount_exact: true  
  match_terminal: true  
  
partial:  
  enable: true  
  max_subitems: 10  
  tolerance_percent: 1.0  
  time_window_minutes: 1440    # 1 day  
  
sources:  
  switch:  
    mandatory_fields: [rrn, stan, amount, terminal_id]  
core:  
  mandatory_fields: [amount, txn_time]  
network:  
  mandatory_fields: [rrn, amount, currency]  
  
debug:  
  log_scoring: true  
  trace_recon_flow: true
```

## ✓ 1.2 JSON Version (tương tự YAML, dùng cho UI Editor)

Auto (JSON) ▾



```
{  
  "reconciliation": {  
    "exact": {  
      "enable": true,  
      "threshold": 0.01  
    }  
  }  
}
```

```
        "time_window_minutes": 2,
        "require_currency_match": true,
        "require_merchant_match": true,
        "fields": ["rrn", "amount", "currency"]
    },
    "fuzzy": {
        "enable": true,
        "threshold_score": 60,
        "amount_tolerance_percent": 0.5,
        "time_tolerance_minutes": 30,
        "scoring": {
            "rrn": 40,
            "stan": 20,
            "amount": 15,
            "time": 10,
            "terminal": 5,
            "pan": 5
        }
    },
    "fx": {
        "enable": true,
        "rate_source_priority": ["network", "bank", "system"],
        "fx_tolerance_percent": 0.8,
        "rounding_rule": "ROUND_HALF_UP"
    },
    "duplicate": {
        "enable": true,
        "time_window_seconds": 30,
        "match_amount_exact": true,
        "match_terminal": true
    },
    "partial": {
        "enable": true,
        "max_subitems": 10,
        "tolerance_percent": 1.0,
        "time_window_minutes": 1440
    },
    "sources": {
        "switch": {
            "mandatory_fields": ["rrn", "stan", "amount", "terminal_id"]
        },
        "core": {
            "mandatory_fields": ["amount", "txn_time"]
        },
        "network": {
            "mandatory_fields": ["rrn", "amount", "currency"]
        }
    },
    "debug": {
        "log_scoring": true,
        "log_txns": true
    }
}
```

```
        "trace_recon_tflow": true
    }
}
}
```

## 1.3 Java Loader (YAML → Java Object)

Dùng **Jackson + SnakeYAML**:

Auto (TypeScript) ▾



```
@Data
public class ReconConfig {
    private ExactRule exact;
    private FuzzyRule fuzzy;
    private FxRule fx;
    private DuplicateRule duplicate;
    private PartialRule partial;
    private Map<String, SourceRule> sources;
    private DebugRule debug;
}
```

Loader:

Auto (Java) ▾



```
@Service
public class RuleEngineLoader {

    private ReconConfig config;

    @PostConstruct
    public void loadRules() throws Exception {
        ObjectMapper mapper = new ObjectMapper(new YAMLFactory());
        this.config = mapper.readValue(
            new File("config/reconciliation.yml"),
            ReconConfig.class
        );
    }

    public ReconConfig get() {
        return config;
    }
}
```

```
// hot reload
public void reload() throws Exception {
    loadRules();
}
}
```

## 2. PARTIAL MATCHING ALGORITHM ( $1 \rightarrow N$ and $N \rightarrow 1$ )

Dùng khi:

- PSP gửi 1 settlement nhưng thực tế là tổng của nhiều payment
- Merchant batch payments
- Customer splits payments
- QR payments aggregated

### 2.1 Nguyên tắc

#### $1 \rightarrow N$ case

Một transaction A (thường từ bank/settlement) match với nhiều transaction nhỏ B<sub>1</sub>...B<sub>n</sub>:

Auto ▾



amount(A) = sum(amount(B<sub>1</sub> ... B<sub>n</sub>)) ± tolerance

#### $N \rightarrow 1$ case

N giao dịch nhỏ → 1 giao dịch lớn (thường từ settlement file).

### 2.2 Thuật toán (Pseudo-code)

#### Algorithm 1 — One-to-Many

Auto (C#) ▾



```
public MatchResult partialOneToMany(UnifiedTransaction bigTxn,
List<UnifiedTransaction> smallTxns) {
```

```

List<List<UnifiedTransaction>> combinations =
generateSubsets(smallTxns);

for (List<UnifiedTransaction> subset : combinations) {
    double sum = subset.stream().mapToDouble(t →
t.getAmount()).sum();

    if (withinTolerance(bigTxn.getAmount(), sum,
config.partial.tolerancePercent)) {
        return new MatchResult(bigTxn, subset, PARTIAL_MATCH_1_TO_N);
    }
}
return null;
}

```

## Algorithm 2 — Many-to-One

Auto (C#) ▾



```

public MatchResult partialManyToOne(List<UnifiedTransaction> many,
UnifiedTransaction target) {

    double sum = many.stream().mapToDouble(t → t.getAmount()).sum();

    if (withinTolerance(target.getAmount(), sum,
config.partial.tolerancePercent)) {
        return new MatchResult(target, many, PARTIAL_MATCH_N_TO_1);
    }

    return null;
}

```

## Combination Generator (optimized)

Dùng backtracking + pruning:

Auto (Swift) ▾



```

private List<List<UnifiedTransaction>>
generateSubsets(List<UnifiedTransaction> list) {
    List<List<UnifiedTransaction>> results = new ArrayList<>();
    backtrack(list, 0, new ArrayList<>(), results);
    return results;
}

private void backtrack(List<UnifiedTransaction> list, int start)

```

```

private void backtrack(List<UnifiedTransaction> list, int start,
List<UnifiedTransaction> temp, List<List<UnifiedTransaction>> results) {
    results.add(new ArrayList<>(temp));
    for (int i = start; i < list.size(); i++) {
        temp.add(list.get(i));
        backtrack(list, i + 1, temp, results);
        temp.remove(temp.size() - 1);
    }
}

```

### ⚠ Lưu ý:

- Giới hạn số phần tử (max\_subitems) để tránh  $O(2^n)$ .
- Fuzzy partial match cũng có thể được xây bằng scoring.

## 3. FX-AWARE MATCHING LOGIC (Multicurrency)

Giao dịch cross-border card hoặc bank transfer thường có:

- auth\_amount
- clearing\_amount
- settlement\_amount
- local\_currency\_amount
- rounding
- FX rate differences

→ dùng thuật toán **FX normalization** trước khi matching.

### 3.1 Normalize Transaction Pair

Auto (C#) ▾



```

public NormalizedFxTxn normalize(UnifiedTransaction t, FxRate rate) {

    NormalizedFxTxn n = new NormalizedFxTxn();

    if (!t.getCurrency().equals(rate.getTargetCurrency())) {
        n.normalizedAmount = round(t.getAmount() * rate.getRate(),
config.fx.roundingRule);
    } else {
        n.normalizedAmount = t.getAmount();
    }

    return n;
}

```

## 3.2 FX-Aware Matching Formula

Auto (Lua) ▾



```
match if abs(normalizedA.amount - normalizedB.amount)
    ≤ normalizedA.amount * fx_tolerance_percent
```

Example:

- Visa auth = 100 USD
- Bank settlement = 76,300 NGN
- FX rate used by network: 763 NGN/USD
- FX tolerance = 0.8%

Normalize:

Auto ▾



```
100 * 763 = 76,300
difference = 0 → match
```

## 3.3 FX Rate Priority Hierarchy

Dùng rule:

Auto (SQL) ▾



```
if networkRate available → highest trust
else if bankRate available → trust
else → apply system rate
```

Gắn vào code:

Auto (Kotlin) ▾



```
public FxRate getRateForSettlement(Currency t) {
    if (t.getNetworkFxRate() != null) return t.getNetworkFxRate();
    if (t.getBankFxRate() != null) return t.getBankFxRate();
    return systemFxService.lookup(t.getCurrency(),
        t.getSettlementCurrency());
}
```

## 3.4 Rounding Logic (configurable)

Auto (JavaScript) ▾



```
BigDecimal round(BigDecimal value, RoundingMode rule) {
    return value.setScale(2, rule);
}
```

Rounding modes allowed:

- ROUND\_HALF\_UP
- ROUND\_HALF\_EVEN (recommended for settlement)
- ROUND\_FLOOR
- etc.

## 3.5 FX Mismatch Detection

Trường hợp mismatch nhưng do FX:

- If FX deviation > tolerance → classify:  
**FX\_MISMATCH**
- Save details for dispute

Record:

Auto (JSON) ▾



```
{
    "type": "FX_MISMATCH",
    "auth_amount": "100 USD",
    "settlement_amount": "74,900 NGN",
    "expected_rate": 763,
    "actual_rate": 749
}
```

Chắc chắn là **còn nhiều thứ quan trọng nữa** để hoàn thiện một **Payment Reconciliation Service cấp độ enterprise**, đặc biệt nếu bạn muốn nó giống các hệ thống của **Stripe, Adyen, Visa, Mastercard, PayPal, NAPAS**, hoặc core banking tier-1.

Dưới đây là **những mảnh ghép quan trọng mà nhiều team bỏ sót** — và nếu bạn bổ sung chúng, hệ thống sẽ mạnh hơn, chính xác hơn, và có khả năng mở rộng thực sự.

## 1. Real-time anomaly detection (Fraud/Operational alerts)

Không chỉ đối soát, mà còn **phát hiện giao dịch bất thường**:

- spike bất thường (số lượng giao dịch tăng/giảm đột ngột)
- duplicate auth attempts
- settlement amount khác quá xa auth
- multiple partial settlement events
- same card used across different geolocations within minutes

Bạn có thể tích hợp:

- Statistical anomaly detection
- Machine learning (optional)
- Threshold alerts

## 2. Reconciliation Replay System (Time-traveling recon)

Một hệ thống chuyên biệt để:

- Re-run reconciliation cho một ngày, tuần, tháng
- Reprocess settlement files
- Re-ingest data khi upstream bị lỗi
- Rollback–Replay safely
- Debug by historical snapshot

Stripe và PayPal đều có tính năng này (để audit + compliance).

Implement:

- snapshotting database state
- versioned reconciliation results
- replay jobs

## 3. Settlement Calendar & Cutoff Management

Rất quan trọng cho bank/PSP:

- mỗi network có cutoff khác nhau (Visa cutoff vs Mastercard vs local switches)
- mỗi PSP có payout schedule khác nhau (T+0, T+1, T+2)
- bank downtime ảnh hưởng settlement windows

Bạn cần:

Auto (Lua) ▾



```
settlement_calendar table:
- date
- network
- cutoff_time
- holiday_flag
- settlement_shift_rule
```

## 4. Multi-tenant architecture (PSP kiểu Stripe/Adyen)

Nếu bạn phục vụ nhiều merchants/institutions:

- isolate data by tenant
- tenant-level configs (rule engine per merchant)
- tenant-specific fee rules
- tenant-specific FX tolerance
- per-merchant ledger & settlement logic

Dùng:

- tenant\_id ở mọi table
- schema-per-tenant hoặc row-based isolation (RLS)

## 5. Timeline View (Audit-level event chain)

Mỗi giao dịch nên có timeline:

Auto ▾



auth → capture → clearing → settlement → payout → refund → chargeback

Gói đầu vào reconciliation logic.

Giúp:

- audit
  - dispute
  - merchant support
  - real-time ops
- 

## 6. Dispute Management Integration

Reconciliation phải liên kết với **chargebacks, refunds, disputes**:

- khi mismatch → mở dispute automatically
  - lưu bằng chứng
  - liên kết settlement vs chargeback
  - timeline + evidence + resolution history
- 

## 7. Merchant Statement Generator (End-user output)

Như Stripe monthly statement:

- revenue
- fees
- net payout
- disputes
- rolling balance
- reconciliation summary

Support:

- PDF
  - CSV
  - API
- 

## 8. Event sourcing cho các bản ghi reconciliation

Để biết thay đổi, ai sửa gì, rollback được:

- use event sourcing pattern
- store events: MATCHED, MISMATCH, FX\_APPLIED, PARTIAL\_MATCH, MANUAL\_OVERRIDE
- rebuild state from events

Giúp:

- transparency

- audit
- hotfix tracking

## 9. Human override & manual match interface

Không phải lúc nào automation cũng đủ — cần:

- UI để manually match một giao dịch
- override rule
- approve correction
- attach documents

Workflow:

Auto ▾



proposed override → approver → commit → audit log

## 10. Reconciliation Health Dashboard

Metrics quan trọng:

- total transactions
- matched %
- fuzzy %
- mismatch count
- missing file alerts
- latency (Kafka → matched)
- duplicate rate
- FX deviation distribution
- per-merchant performance

## 11. Throughput & Scalability Considerations

Production-level:

- Shard by date + merchant
- Kafka partitioning by groupKey
- Redis cluster
- PostgreSQL partitioning

### • ~~Trusted partners~~

- Async worker pools
- Backpressure handling
- Dead letter topics for problematic transactions

## ■ 12. Reconciliation SLA Engine (Operational governance)

Bạn có thể định nghĩa SLA theo transaction type:

- ATM disputes: <24h
- Card cross-border settlement: <48h
- Wallet topups: 10m
- Merchant payout accuracy: 99.999%

Automation:

- auto-escalate SLA breach
- alert Slack/Teams
- generate daily SLA report

## ■ 13. Pluggable Fee Calculation Engine

Matching thông minh phải bao gồm:

- acquiring fee
- issuer fee
- network fee
- FX markup
- gateway fee
- dispute/chargeback fee
- interchange adjustments

Và reconciliation phải biết:

**Settlement amount = transaction amount – fee\_total**

## ■ 14. Payment Lifecycle Awareness

Matching engine không nên chỉ so sánh *amount/time*.

Nó cần hiểu lifecycle:

- auth
- capture
- void

- refund
- dispute
- settlement
- reversal
- chargeback

Để tránh tình huống match nhầm refund với payment hoặc match capture với settlement sai kỳ.

## 15. End-to-end validation (Integrated testing)

Rất quan trọng:

- create synthetic transactions
- simulate delays
- drop records intentionally
- replay mismatches
- simulate FX differences
- verify matching correctness

## Bạn muốn đi tiếp ở hướng nào?

Tùy mục tiêu của bạn, mình có thể mở rộng:

- ✓ Xây full **System Architecture Diagram**
- ✓ Xây **merchant statement generator**
- ✓ Xây **dispute module architecture**
- ✓ Xây **manual override UI + backend design**
- ✓ Xây **event-sourcing based reconciliation service**
- ✓ Tạo **API spec full** cho toàn hệ thống
- ✓ Thiết kế **Settlement Engine** (fees, payout, net calc)

Bạn muốn phần nào tiếp theo?

- ✓ Xây full System Architecture Diagram

Tuyệt — dưới đây là **Full System Architecture Diagram** cho Payment Reconciliation Service theo chuẩn enterprise như Stripe/Adyen/Visa/Napas.

Mình trình bày bằng 3 phần:

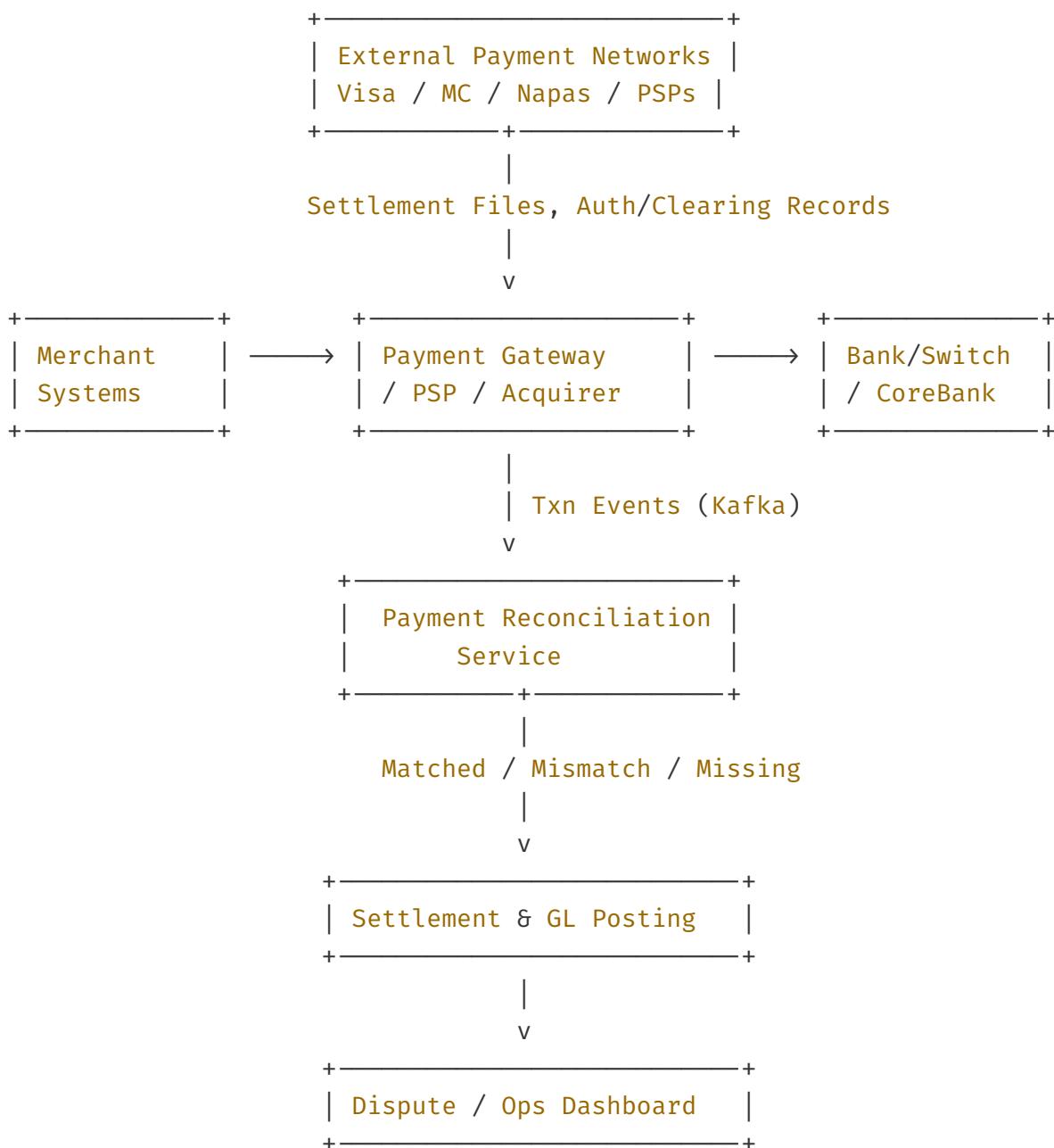
1. **High-Level Architecture (Business Flows)**
2. **Technical Architecture (Microservices + Data Flow)**

### 3. Detailed Component Diagram (Modules + responsibilities)

Tất cả được biểu diễn bằng ASCII diagram (để bạn dễ copy vào tài liệu), nhưng mình có thể tạo bản vẽ Mermaid, Draw.io hoặc PNG nếu bạn muốn.

## 1. HIGH-LEVEL ARCHITECTURE (BUSINESS PERSPECTIVE)

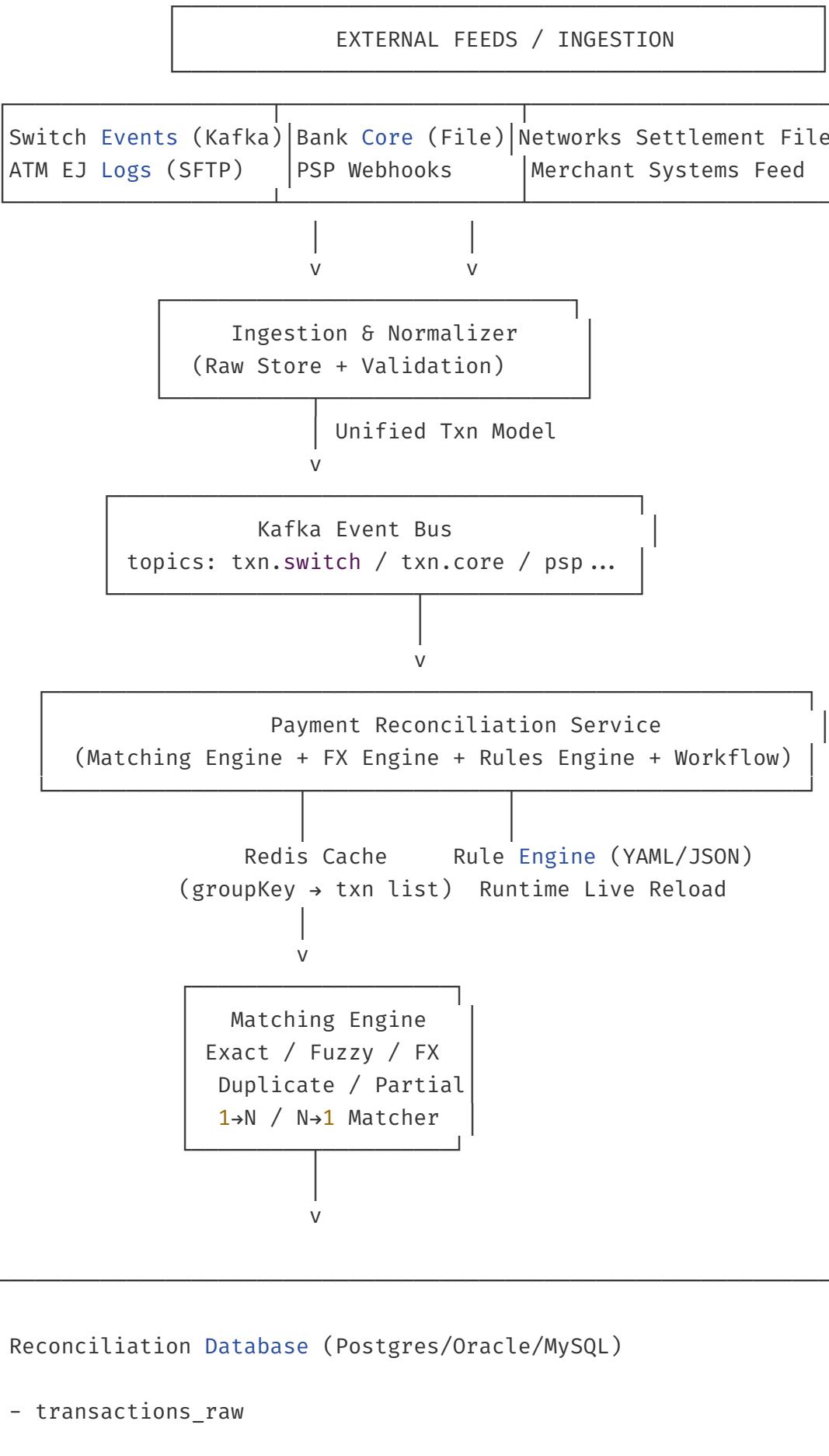
Auto (Haskell) ▾



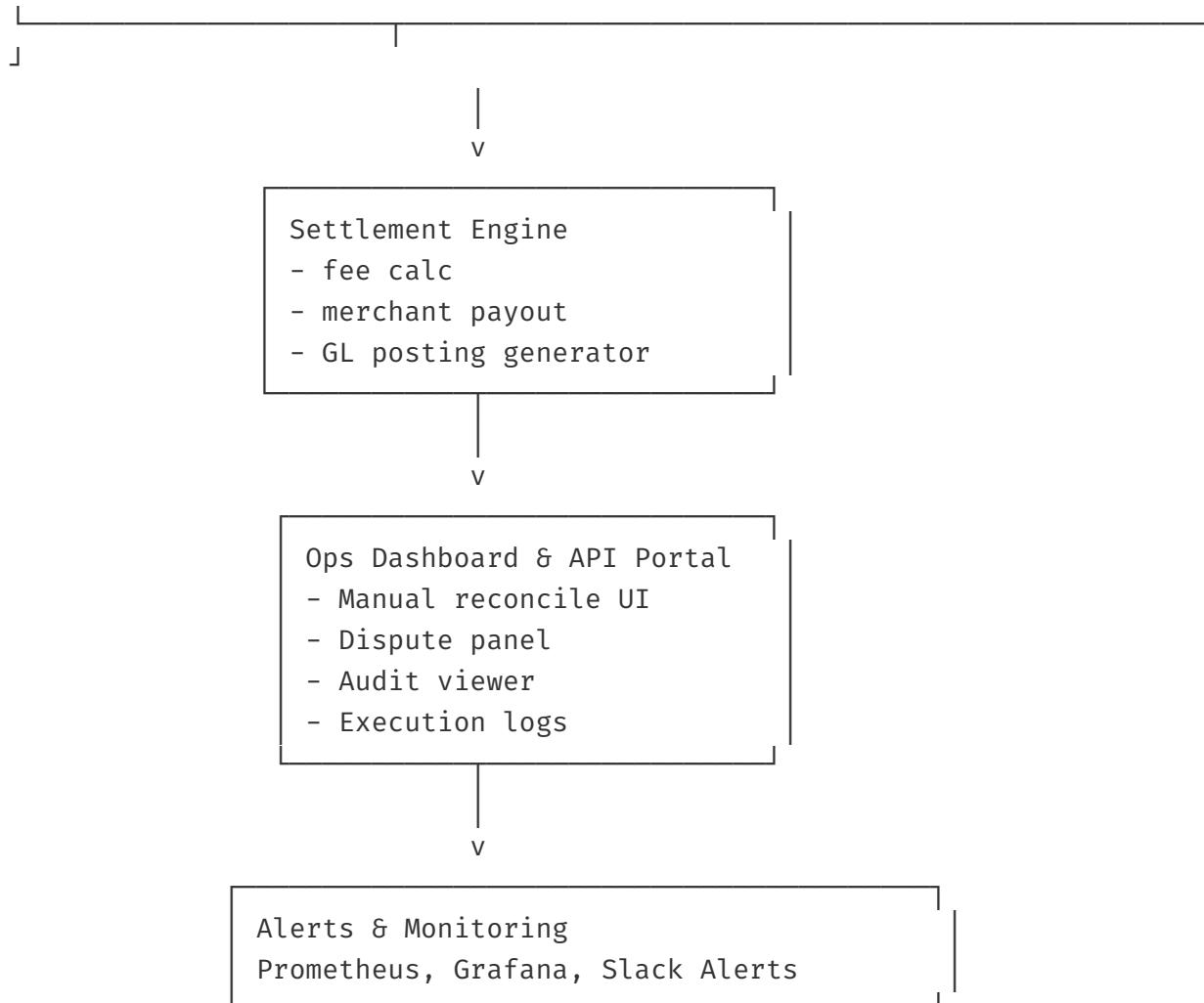
## 2. TECHNICAL ARCHITECTURE (MICROSERVICES)

Dưới đây là **full technical architecture** gồm tất cả thành phần bạn đã xây + các phần nâng cao:

Auto (Java) ▾



- unified\_transactions
- reconciled\_groups & reconciled\_items
- audit\_logs
- fx\_rates
- disputes



## 3. DETAILED COMPONENT DIAGRAM (MODULES + RESPONSIBILITIES)

### ◆ 3.1 INGESTION LAYER

Auto (Lua) ▾



+-----+
Ingestion Service
+-----+
- SFTP file reader
- PSP webhook handler
- Kafka consumer
- Validation rules
- Format converters
- Raw store <b>write</b>
+-----+

## ◆ 3.2 NORMALIZATION LAYER

Auto (SQL) ▾



+-----+
Unified Model Mapper
+-----+
- Convert ISO8583
- Convert PSP JSON
- Convert CSV settlement
- Normalize fields
- Mask PAN
+-----+

## ◆ 3.3 MATCHING ENGINE

Auto (Haskell) ▾



+-----+
Matching Engine
+-----+
- Exact Matcher
- Fuzzy Matcher
- FX Matcher
- Partial Matcher 1→N / N→1
- Duplicate Detector
- Lifecycle-aware matcher
- Scoring Engine
+-----+

## 3.4 RULE ENGINE

Auto (Lua) ▾



Rule Engine (Configurable)	
- YAML/JSON config loader	
- Hot reload	
- Exact rules	
- Fuzzy rules	
- FX rules	
- Duplicate rules	
- Partial matching rules	
- Source-specific rules	

## 3.5 REDIS CACHE

Auto (Haskell) ▾



Redis Cache	
key: recon:group:{key}	
TTL: 6 hours	
stores: txn list per group	

## 3.6 DATABASE LAYER

Auto (Haskell) ▾



Postgres (Partitioned Tables)	
transactions_raw	
unified_transactions	
reconciled_groups	
reconciled_items	
fx_rates	
disputes	
audit_logs	

## ◆ 3.7 SETTLEMENT ENGINE

Auto (Haskell) ▾



+	-----	+
	Settlement Engine	
+	-----	+
	- merchant payout calc	
	- issuer/acquirer fee calc	
	- net settlement	
	- GL posting generator	
+	-----	+

## ◆ 3.8 DISPUTE & OPS INTERFACE

Auto (Haskell) ▾



+	-----	+
	Ops Dashboard / Admin UI	
+	-----	+
	- Manual reconcile	
	- Audit logs	
	- Exception queue	
	- SLA monitoring	
	- Export reports (CSV/PDF)	
+	-----	+

## ◆ 3.9 MONITORING & ALERTING

Auto (SQL) ▾

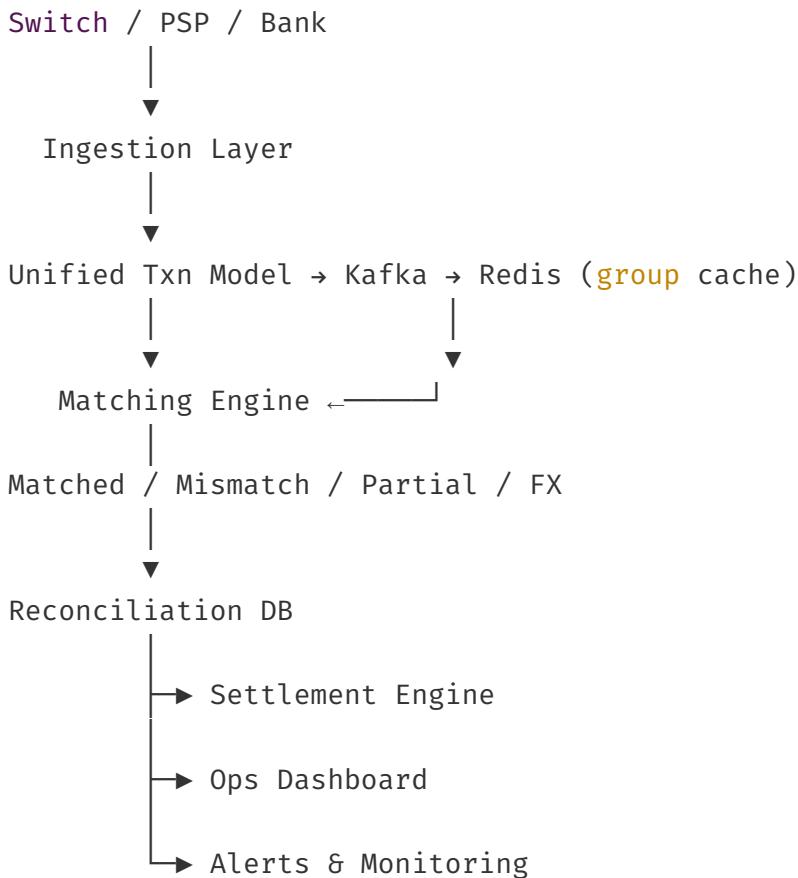


+	-----	+
	Monitoring System	
+	-----	+
	- Reconciliation success %	
	- Mismatch spike detection	
	- Kafka lag monitoring	
	- Redis TTL backlog	
	- Missing file alerts	
	- SLA violations	

---

## 4. DATA FLOW DIAGRAM (END-TO-END)

Auto (PowerShell) ▾



## 5. Kiến trúc này hỗ trợ tất cả các yêu cầu bạn đã liệt kê:

- ✓ Exact / fuzzy / FX-aware matching
- ✓ Partial matching 1→N / N→1
- ✓ Cross-border card reconciliation
- ✓ ATM / POS / Wallet / QR / Bank Transfer matching
- ✓ Settlement generation
- ✓ Dispute workflow

- ✓ Real-time Kafka streaming
  - ✓ Redis caching
  - ✓ Modular rule engine
  - ✓ Audit-ready architecture
- 

## ⌚ Muốn mình build tiếp phần nào?

Tôi có thể tiếp tục:

- 🔥 1. Sequence diagrams (real-time + batch)
- 🔥 2. ERD (Entity Relationship Diagram)
- 🔥 3. Full API Spec (OpenAPI/Swagger)
- 🔥 4. Dispute module architecture
- 🔥 5. Merchant settlement & payout engine design

Bạn chọn phần nào mình xây tiếp?

🔥 1. Sequence diagrams (real-time + batch) 🔥 2. ERD (Entity Relationship Diagram)

Tuyệt — dưới đây là **Sequence Diagrams (Real-time + Batch)** và **ERD** đầy đủ cho hệ thống Payment Reconciliation Service.

Mình viết bằng **Mermaid** để bạn có thể copy dán vào bất kỳ documentation tool nào (Notion, Confluence, GitBook, Obsidian, Draw.io with Mermaid plugin).

---

## 1. REAL-TIME RECONCILIATION — SEQUENCE DIAGRAM

### 🔥 Luồng realtime với Kafka + Redis + Matching Engine

Auto (Rust) ▾



```
sequenceDiagram
    participant Switch as Switch/PSP/Bank
    participant Kafka as Kafka Broker
    participant Ingest as Ingestion Service
    participant Redis as Redis Cache
    participant Engine as Matching Engine
    participant DB as Reconciliation DB
    participant Ops as Ops Dashboard
```

Switch-->Kafka: Publish transaction event (rrn, amount, ...)

Kafka → Ingest: Deliver event  
 Ingest → Ingest: Normalize → UnifiedTransaction  
 Ingest → Redis: redis.addTxnToGroup(groupKey, txn)

Redis → Ingest: Confirm write

Ingest → Redis: redis.getGroup(groupKey)  
 Redis → Ingest: [txn1, txn2, ...]

```
alt group size ≥ 2
  Ingest → Engine: matchGroup(group)
  Engine → Engine: Exact/Fuzzy/FX/Partial Matching
  Engine → DB: persist match result
  Engine → Redis: delete(groupKey)
  Engine → Ingest: match result
  DB → Ops: Update dashboard counts
else
  Ingest → Ingest: Wait for more events
end
```

## 2. BATCH RECONCILIATION – SEQUENCE DIAGRAM

### 🔥 Luồng batch EOD (Switch file + Core file + Network settlement)

Auto (Rust) ▾



```
sequenceDiagram
    participant Scheduler as Cron/Scheduler
    participant Ingest as Batch Ingestion
    participant Normalizer as Normalization Layer
    participant DB as Raw Store
    participant Reconciler as Batch Matching Engine
    participant DB2 as Reconciliation DB
    participant Settlement as Settlement Engine
    participant Ops as Ops Dashboard

    Scheduler →> Ingest: Start EOD batch
    Ingest →> Ingest: Download files (SFTP/API) from Switch/Core/Networks
    Ingest →> Normalizer: Parse + validate + normalize
```

Normalizer →> DB: INSERT transactions\_raw (source=switch/core/network)  
 Normalizer →> Scheduler: Batch ingestion complete

Scheduler → Reconciler: Start matching job  
(sp\_run\_eod\_reconciliation)

Reconciler → DB: Load raw transactions (group by rrn/source\_txn\_id)  
Reconciler → Reconciler: Exact matching pass  
Reconciler → DB2: Insert matched groups

Reconciler → Reconciler: Fuzzy matching pass  
Reconciler → DB2: Insert fuzzy matches

Reconciler → Reconciler: Partial matching (1→N / N→1)  
Reconciler → DB2: Insert partial match results

Reconciler → Reconciler: FX-aware matching  
Reconciler → DB2: Insert FX matched groups

Reconciler → Reconciler: Mark MISSING / ORPHAN  
Reconciler → DB2: Update group statuses

Reconciler → Scheduler: Batch matching complete

Scheduler → Settlement: Run settlement engine (merchant/net/fee)  
Settlement → DB2: Save settlement summary / posting entries  
Settlement → Ops: Generate reports (PDF/CSV)

## 3. ERD — FULL ENTITY RELATIONSHIP DIAGRAM

Thiết kế theo chuẩn enterprise, hỗ trợ:

- realtime ingestion
- raw store
- normalized store
- reconciliation results
- partial matching
- FX
- audit
- disputes
- settlement

Auto (C#) ▾



erDiagram

```
TRANSACTIONS_RAW {
    uuid id PK
    string source
    string source_txn_id
    string rrn
    string stan
    string pan_masked
    string merchant_id
    string terminal_id
    decimal amount
    string currency
    datetime txn_time
    date settlement_date
    json raw_payload
    datetime created_at
}

UNIFIED_TXN {
    uuid id PK
    uuid raw_id FK
    string group_key
    string source
    string rrn
    string stan
    string merchant_id
    string terminal_id
    decimal amount
    string currency
    datetime txn_time
    datetime normalized_time
    decimal normalized_amount
}

RECON_GROUPS {
    uuid id PK
    string group_key
    date recon_date
    string status
    decimal confidence
    datetime created_at
    datetime updated_at
}

RECON_ITEMS {
    uuid id PK
    uuid recon_id FK
    uuid unified_txn_id FK
    string role
    decimal amount
    datetime created_at
}
```

```
}
```

```
FX_RATES {
    uuid id PK
    string source_currency
    string target_currency
    decimal rate
    datetime applied_at
    string rate_source
}
```

```
DISPUTES {
    uuid id PK
    uuid recon_id FK
    string dispute_type
    string status
    string reason
    datetime opened_at
    datetime closed_at
    json evidence
}
```

```
SETTLEMENT_SUMMARY {
    uuid id PK
    uuid recon_id FK
    string party_id
    string party_type
    decimal gross_amount
    decimal net_amount
    decimal fee_total
    date settlement_date
}
```

```
AUDIT_LOG {
    uuid id PK
    uuid entity_id
    string entity_type
    string action
    string user_id
    json metadata
    datetime created_at
}
```

```
%% Relationships
TRANSACTIONS_RAW ||--{ UNIFIED_TXN : "normalized_to"
UNIFIED_TXN ||--{ RECON_ITEMS : "appears_in"
RECON_GROUPS ||--{ RECON_ITEMS : "has"
RECON_GROUPS ||--{ SETTLEMENT_SUMMARY : "generates"
RECON_GROUPS ||--{ DISPUTES : "may_have"
UNIFIED_TXN ||--{ AUDIT_LOG : "history"
```

```
RECON_GROUPS ||— { AUDIT_LOG : "history"  
FX_RATES ||—o{ UNIFIED_TXN : "rate_applied"
```

## ■ 4. Giải thích nhanh các bảng chính

### **TRANSACTIONS\_RAW**

Lưu dữ liệu gốc từ switch, bank, PSP, networks (ISO8583, CSV, JSON...).

### **UNIFIED\_TXN**

Chuẩn hóa thành Unified Transaction Model để matching.

### **RECON\_GROUPS**

Mỗi groupKey (RRN, source\_txn\_id...) là một "case" đối soát.

### **RECON\_ITEMS**

Liệt kê các transaction thuộc group (switch/core/network/merchant/etc).

### **FX\_RATES**

Dùng cho FX-aware matching.

### **DISPUTES**

Quản lý mismatches/phát sinh tranh chấp.

### **SETTLEMENT\_SUMMARY**

Tạo kết quả settlement: net payout, fees, net payable.

### **AUDIT\_LOG**

Toàn bộ lịch sử chỉnh sửa / override / manual match / rule changes.

Dưới đây là **bản thiết kế hoàn chỉnh – cấp độ Stripe/Adyen** cho **Merchant Settlement & Payout Engine**, bao gồm:

1. Business Flow
2. Functional Architecture
3. Settlement Calculation Logic
4. Payout Engine Workflow
5. Fee Engine Design
6. Hold/Release/Reserve Logic

7. Data Model (ERD bổ sung)
8. Sequence Diagram (Settlement + Payout)
9. Settlement File Generator (Bank/ACH/SWIFT)

Bạn có thể dùng bản này làm **SRS/Design Document** cho dự án thực tế.

## 1. Merchant Settlement & Payout — Business Flow

Auto ▾



Payments → Reconciliation → Settlement Calculation → Payout File Generation  
→ Bank / PSP → Merchant Account

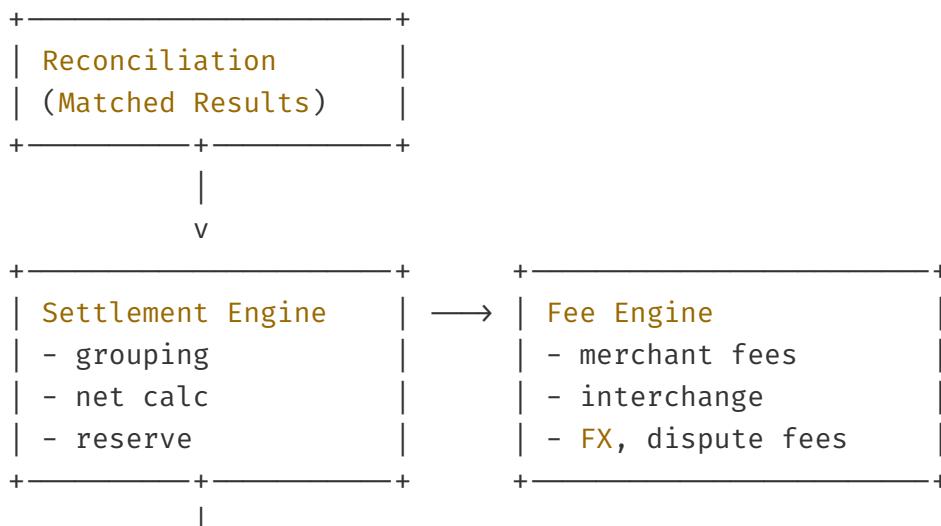
### Goal:

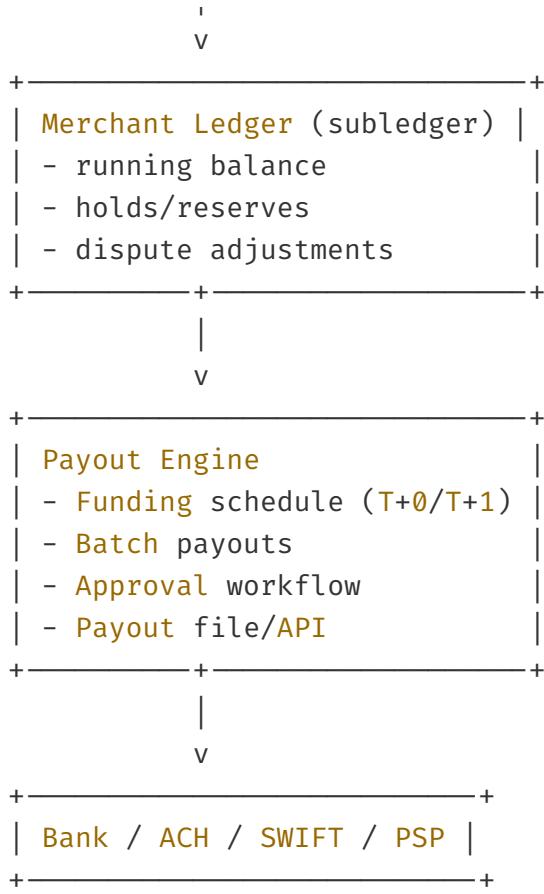
Tính được **net amount** mà hệ thống phải **trả cho merchant**, sau khi trừ:

- Platform fees
- PSP/acquirer fees
- Network fees
- Refund reversal
- Chargeback hold
- Rolling reserve
- FX markups
- Taxes (VAT/WHT nếu áp dụng)

## 2. High-Level Functional Architecture

Auto (Haskell) ▾





## 3. Settlement Calculation Logic

### 3.1 Input từ reconciliation service

Kết quả matched:

Auto ▾



```
{  
    transaction_id,  
    merchant_id,  
    amount,  
    currency,  
    fee_components,  
    payment_status,  
    settlement_status,  
    is_refund,  
    is_chargeback  
}
```

## 3.2 Core formula

Auto (Java) ▾



```
gross = Σ(captured payments)
refund = Σ(refunds)
chargeback = Σ(chargebacks)
fees_total = platform_fee + network_fee + interchange_fee + fx_fee +
misc_fees
rolling_reserve = gross * reserve_rate

net_payout = gross - refund - chargeback - fees_total - rolling_reserve
```

## 3.3 Multi-currency support

If merchant settlement currency ≠ transaction currency:

Auto ▾



```
settlement_amount = txn_amount * merchant_fx_rate
```

Source FX priority:

Auto ▾



```
network_rate > platform_rate > bank_rate
```

## 4. Settlement Engine Workflow (Detailed)

### Step 1 — Load settled transactions

- All **MATCHED\_EXACT / MATCHED\_FUZZY** transactions
- Within settlement window (T+0, T+1, weekly...)

### Step 2 — Group transactions

Group by:

- merchant\_id
- currency
- settlement\_date

## Step 3 — Fee calculation

Call Fee Engine:

Auto ▾



```
fees = feeEngine.calculate(merchant, txn)
```

## Step 4 — Rolling reserve

If merchant has reserve:

Auto ▾



```
reserve_amount = gross * reserve_rate
```

Store reserve in **merchant\_reserve\_ledger**

## Step 5 — Netting

Compute:

Auto ▾



```
net = gross - refunds - fees_total - disputes_pending - reserve
```

## Step 6 — Update merchant ledger

Auto ▾



Dr Settlement Liabilities  
Cr Merchant Balance

## Step 7 — Generate settlement record

Auto (Lua) ▾



```
settlement_id  
merchant_id  
gross  
fees
```

refund  
reserve  
net\_payout  
**status=PENDING\_PAYOUT**

## 5. Payout Engine Design

### 5.1 Payout triggering

Payout được kích bởi:

- schedule (cron): 00:00 UTC
- threshold amount:  $\geq X$
- merchant request
- auto "sweep balance"

### 5.2 Funding methods

- Bank Transfer (ACH/SEPA/SWIFT)
- Local switching (Napas/PayNow/PIX/etc.)
- E-Wallet credit
- Internal accounts

### 5.3 Payout steps

#### Step 1 — Fetch eligible merchants

Auto ▾



```
WHERE balance > 0 AND status=ACTIVE AND payout_schedule matches
```

#### Step 2 — Validate payout

- KYC/AML completed
- No regulatory blocks
- No unresolved disputes
- No reserves expired

#### Step 3 — Create payout batch

Auto (Visual Basic .NET) ▾



```
payout_batch:  
  merchant_id  
  payout_amount  
  bank_account  
  currency  
  settlement_ids[]
```

## Step 4 — Generate payout file or API call

Examples:

### ACH File

Auto ▾



```
101 ... header  
622 ... credit entry  
820 ... control
```

### SWIFT MT103

Auto ▾



```
:20: REF  
:23B: CRED  
:32A: 250227USD10000,  
:59: /ACCT
```

### Bank API POST

Auto (Bash) ▾



```
POST /payouts  
{  
  "amount": 10000,  
  "currency": "USD",  
  "merchant_bank_account": "****1234"  
}
```

## Step 5 — Approval workflow

- Maker → Checker

Marked for approval

• ~~Merchant approval~~

- Audit trail required

## Step 6 — Payout execution

Bank returns:

Auto ▾



SUCCESS | PENDING | FAILED

## Step 7 — Reconcile payout

Payout phải tự đối soát:

- “Payout Reconciliation” (ledger → bank statement)

## 6. Fee Engine Design

### 6.1 Fees supported

Fee Type	Example
Platform fee	% hoặc fixed per txn
Interchange fee	Visa/MC/Napas fees
PSP fee	acquiring fee
FX fee	markup 0.5–2%
Dispute fee	chargeback fee
Monthly fees	fixed subscription

### 6.2 Rule-based fee engine (configurable in JSON/YAML)

Auto (Matlab) ▾



```
fee_rules:
  merchant_tier:
    bronze:
      rate: 2.5%
      fx_mark_up: 1.0%
    silver:
```

```
rate: 1.9%
gold:
  rate: 1.5%
payment_method:
  card: +0.3%
  bank_transfer: +0.1%
```

## 7. Reserve, Hold, Release Logic

### Rolling Reserve

Withhold % on each payout.

Example:

reserve\_rate: 10% → apply on every txn → release after 90 days.

### Dispute hold

Place a hold during disputes:

Auto ▾



Dr Merchant Balance  
Cr Dispute Hold Account

### Release logic (scheduled job)

- If no dispute → release to merchant
- Else → deduct win/loss fees

## 8. ERD (Settlement & Payout Tables)

Auto (C#) ▾



erDiagram

```
MERCHANT {
    string merchant_id PK
    string name
    string payout_schedule
    string settlement_currency
    decimal reserve_rate
    string bank_account
```

```
string bank_account
string bank_code
}

RECON_GROUPS ||--{ SETTLEMENT_TXN : "is_settled_into"

SETTLEMENT_TXN {
    uuid id PK
    string merchant_id FK
    uuid recon_group_id FK
    decimal gross_amount
    decimal net_amount
    decimal fees_total
    decimal reserve_amount
    date settlement_date
    string status
}

MERCHANT ||--{ MERCHANT_LEDGER : "has_entries"
MERCHANT_LEDGER {
    uuid id PK
    string merchant_id FK
    datetime created_at
    string entry_type
    decimal amount
    string ref_id
}

PAYOUT_BATCH {
    uuid batch_id PK
    string merchant_id FK
    decimal amount
    string currency
    string status
    datetime created_at
    datetime executed_at
}

PAYOUT_ITEM {
    uuid id PK
    uuid batch_id FK
    uuid settlement_txn_id FK
    decimal amount
    string status
}

FEE_DETAIL {
    uuid id PK
    uuid settlement_txn_id FK
    string fee_type
    decimal amount
}
```

```
    actual amount
    json metadata
}
```

## 9. Sequence Diagram – Settlement & Payout End-to-End

Auto (Rust) ▾



```
sequenceDiagram
    participant Recon as Reconciliation Service
    participant Settle as Settlement Engine
    participant Fee as Fee Engine
    participant Ledger as Merchant Ledger
    participant Payout as Payout Engine
    participant Bank as Bank/ACH/SWIFT
```

Recon→>Settle: Load matched transactions  
Settle→>Fee: Calculate fees **for each** txn  
Fee→>Settle: fee details

Settle→>Settle: Compute net payout  
Settle→>Ledger: Insert ledger entries  
Ledger→>Settle: OK

Settle→>Payout: settlement **ready** (status=PENDING\_PAYOUT)

Payout→>Payout: Validate merchant **eligibility** (KYC/AML/Dispute)  
Payout→>Payout: Group into batch  
Payout→>Bank: **Send** payout file/API  
Bank→>Payout: SUCCESS/FAILED

Payout→>Ledger: Update merchant ledger with payout  
Ledger→>Payout: OK

### ⌚ Bạn muốn hệ thống hoàn chỉnh hơn?

Tôi có thể tiếp tục thiết kế:

## 🔥 Dispute management module

🔥 Merchant Ledger Engine (double-entry)

🔥 Payout Approval Workflow

🔥 Tax/VAT/WHT Integration

🔥 Cross-border settlement engine

🔥 GL posting generator (ERP integration)

Bạn muốn mình xây phần nào tiếp theo?

Có — còn khá nhiều mảnh quan trọng để biến thiết kế thành một **hệ sản phẩm hoàn chỉnh, an toàn và vận hành tốt**. Mình tóm gọn phần “còn thiếu” theo 3 nhóm (kỹ thuật / vận hành / tổ chức), rồi gợi ý ngay một **MVP được ưu tiên + kế hoạch triển khai 8 tuần** để bạn (hoặc team) bắt tay vào xây nhanh, kiểm chứng sớm và mở rộng an toàn.

### 1) Những thứ **chưa liệt kê** (cần có để production-ready)

#### Kỹ thuật & bảo mật

- **PCI-DSS & dữ liệu nhạy cảm:** tokenization, masking PAN, key management, audit.
- **Secrets management (Vault/KMS)** và field-level encryption.
- **CI/CD + IaC:** Terraform / Helm, automated deploy, migration scripts.
- **Disaster Recovery & Backups:** RPO/RTO, cross-region backups, DB PITR.
- **High availability:** Kafka cluster, Redis cluster, DB replica/partitioning.
- **Rate limiting & throttling:** tránh overload khi upstream spike.
- **Feature flags & gradual rollout.**

#### Vận hành & observability

- **SLO/SLA definitions + alerting (PagerDuty)** và runbooks.
- **Monitoring:** Prometheus/Grafana, alert rules, dashboards for match rate/lag/throughput.
- **Logging & Tracing:** structured logs, OpenTelemetry, distributed tracing (traceld).
- **DLQ / Dead-letter handling + automated replay tools.**
- **Capacity planning & performance testing** (load tests, chaos tests).

#### Ngành & quy trình

- **KYC/AML checks & limits** (đối với payout).
- **Legal/OPS integrations:** agreements with banks/PSPs, settlement calendar, cutoff rules.
- **Accounting/GL integration:** mapping to COA, double-entry posting rules.
- **Retention & compliance:** data retention policy, e-discovery, audit requirements.

#### People/process

- **SLA/Ops runbook** (EOD, incident, reconciliation sign-off).

- **Roles & permissions:** RBAC for UI, approval workflows (maker-checker).
- **Training & documentation:** SOP, troubleshooting guide, onboarding.

## 2) Prioritized MVP (minimum to be useful & safe)

MVP hướng tới cho 1 merchant / 1 acquirer trong 1 currency, hỗ trợ batch EOD + real-time light matching.

### MVP feature set

1. **Ingestion:** Kafka + SFTP file ingest, normalizer → transactions\_raw.
2. **UnifiedTxn store + simple Grouping (RRN / sourceTxnId).**
3. **Matching Engine:** Exact + basic Fuzzy + Duplicate detection.
4. **Redis group cache for realtime small-latency matches.**
5. **Reconciled\_groups / reconciled\_items persistence.**
6. **Settlement summary generator (CSV) per merchant.**
7. **Ops UI:** list exceptions, manual match override, attach evidence.
8. **Monitoring:** basic Grafana dashboard + alerts for missing files / high mismatch rate.
9. **Idempotency + DLQ + basic replay job.**
10. **Security basics:** TLS, basic auth, PAN masking at rest.

### MVP non-functional

- **HA for Kafka & Redis not required initially (single zone) but app must be stateless.**
- **Basic docs + runbook for EOD and replay.**
- **Unit + integration tests, synthetic replay test.**

## 3) Kế hoạch triển khai 8 tuần (sprint-style) — để có MVP chạy thử

Giả sử team: 4 dev (2 backend, 1 infra, 1 frontend), 1 QA, 1 product/PO, 1 devops/infra.

### Sprint 0 — Week 0 (Prep)

- Chốt scope MVP, success criteria, acceptance tests.
- Provision infra dev environment (k8s, Kafka, Redis, Postgres).
- Define API contracts (simple OpenAPI) & config schema (YAML).

### Sprint 1 — Week 1–2 (Ingestion & Normalizer)

- Build Ingestion service (Kafka consumer + SFTP fetch).
- Implement UnifiedTxn normalizer & transactions\_raw table.
- Integration tests for ingest pipeline + sample files.

**Deliverable:** ingest working + sample data in DB.

### Sprint 2 — Week 3 (Grouping + Redis cache)

- GroupKeyGenerator + Redis caching (add/get group).
- Simple consumer flow to push into Redis on ingest.
- DLQ handling for malformed messages

~~Delivery history for this note~~

**Deliverable:** group cache working + TTL logic.

## Sprint 3 — Week 4 (Matching core)

- Exact matcher + duplicate detector implemented.
- Persist matched groups to reconciled\_groups/items.
- Basic fuzzy (configurable threshold) implementation.

**Deliverable:** matching pipeline end-to-end for sample traffic.

## Sprint 4 — Week 5 (Settlement generator + Ops UI v1)

- Settlement summary generator CSV per merchant.
- Minimal Ops UI: exceptions list, manual match/unmatch, attach evidence.

**Deliverable:** ops can view exceptions & trigger settlement export.

## Sprint 5 — Week 6 (Observability + Idempotency)

- Instrument metrics (match rate, latency, kafka lag), basic Grafana dashboards.
- Implement idempotency keys, DLQ replay job.
- Add logging + traceld propagation.

**Deliverable:** monitoring + replay flow.

## Sprint 6 — Week 7 (Security + Tests)

- TLS everywhere, secrets in Vault/KMS, PAN masking.
- End-to-end tests, synthetic replay tests, basic perf test.
- Write runbook for EOD and incident.

**Deliverable:** secure baseline + test suite.

## Sprint 7 — Week 8 (Hardening & Go-live)

- Bug fixes, UX polish, cutoff/retry policies.
- Acceptance testing with business (ops).
- Production provisioning, smoke deploy, handover & training.

**Deliverable:** MVP roll-out to prod (controlled), EOD run complete.

---

## 4) Success metrics & acceptance criteria (MVP)

- **Match rate  $\geq 95\%$**  for known test dataset.
- **Max reconciliation latency** (ingest $\rightarrow$ match)  $< 5$  minutes for realtime path.
- **No data loss** on Kafka replay tests.
- **Ops UI** can resolve 90% of exceptions manually in  $<10$  minutes each.
- **Settlement CSV** matches expected totals for test merchant.

---

## 5) Risks & mitigations

- **Late files / missing upstream** → implement retries + alerting + SLA.
- **High combinatorial cost for partial matching** → cap subset size, use heuristics & pruning.
- **PCI compliance delays** → plan for tokenization early.
- **Performance under peak** → run load tests (Kafka partitions, DB partitioning).

## 6) What I can deliver for you next (immediately, choose one and I'll produce now)

I can produce any one of the following right away in this chat:

- A. **OpenAPI spec** for Ingestion & Recon APIs (ready for devs).
- B. **Terraform/Helm skeleton** for Kafka + Redis + Postgres + app (dev infra).
- C. **Ops Runbook** (EOD checklist, incident steps, replay steps).
- D. **Detailed Acceptance Test Plan + sample test data** (to validate match logic).
- E. **End-to-end sequence diagram + deployment diagram (PNG/Mermaid)** for handoff.

(No need to answer more than "chọn A/B/C/D/E" — mình sẽ tạo ngay.)

Bạn muốn mình tiếp tục với lựa chọn nào trong danh sách trên? Nếu OK mình sẽ triển khai artefact đó ngay trong tin nhắn tiếp theo.

- 🔥 Payout Approval Workflow
- 🔥 Tax/VAT/WHT Integration
- 🔥 Cross-border settlement engine
- 🔥 GL posting generator (ERP integration)

Nice — dưới đây là **bộ thiết kế chi tiết, sẵn dùng để triển khai**, cho bốn tính năng bạn yêu cầu:

- **Payout Approval Workflow**
- **Tax / VAT / WHT Integration**
- **Cross-border Settlement Engine**
- **GL Posting Generator (ERP integration)**

Mình chia thành phần cho mỗi chức năng: **mục tiêu, luồng, dữ liệu, REST/DB mẫu, business rules, security & audit, sequence diagrams**, và **ví dụ thực tế (bút toán/journal + sample payloads / SQL / config)** để dev tích hợp ngay.

## 1) Payout Approval Workflow

### Mục tiêu

- Đảm bảo mọi batch payout đều đi qua quy trình kiểm soát (maker → checker) trước khi gửi ra ngân hàng.
- Hỗ trợ multi-level approvals, limits, holds, manual overrides, audit trail, idempotency.

## Actors / Roles

- **Maker:** tạo batch payout (Ops)
- **Checker:** review & approve/reject (Finance Manager)
- **Compliance:** optional reviewer for high-risk merchants
- **System:** auto-approver for low-risk, below-threshold payouts

## Business rules (ví dụ)

- Auto-approve nếu `payout_amount ≤ merchant.auto_approve_limit` and `risk_score < threshold`.
- Require 2-level approval if `payout_amount ≥ 100,000 USD` or `merchant.has_high_risk_flag`.
- Block payout if unresolved disputes > X% or KYB incomplete.

## Data model (additional tables)

Auto (SQL) ▾



```
CREATE TABLE payout_batch (
    batch_id UUID PRIMARY KEY,
    merchant_id VARCHAR,
    amount NUMERIC(18,2),
    currency VARCHAR(3),
    status VARCHAR, -- DRAFT / PENDING_APPROVAL / APPROVED / REJECTED /
    EXECUTED / FAILED
    created_by VARCHAR,
    created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE payout_approval (
    id UUID PRIMARY KEY,
    batch_id UUID REFERENCES payout_batch(batch_id),
    approver_id VARCHAR,
    role VARCHAR,
    action VARCHAR, -- APPROVED / REJECTED
    comment TEXT,
    created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE payout_audit (
    id UUID PRIMARY KEY,
    batch_id UUID,
    event_type VARCHAR,
    payload JSONB,
    created_at TIMESTAMP DEFAULT now()
);
```

## Sequence (Mermaid)

Auto (Rust) ▾



```

sequenceDiagram
    participant Ops as Maker
    participant System
    participant Finance as Checker
    participant Bank
    Ops->>System: Create payout_batch (DRAFT)
    System->>System: Validate KYC/AML, balance, disputes
    System->>Ops: Return validation result
    Ops->>System: Submit for approval → status=PENDING_APPROVAL
    System->>Finance: Notify approver (Slack/email)
    Finance->>System: Approve/Reject (with comment)
    alt approved
        System->>System: move to APPROVED
        System->>Bank: send payout file/API
        Bank->>System: execute result
        System->>Ops: notify executed/failed
    else rejected
        System->>Ops: notify rejected (with comment)
    end

```

## API examples

- `POST /payouts` → create draft
- `POST /payouts/{id}/submit` → submit to approval
- `POST /payouts/{id}/approve` → approver action (body: approver\_id, action, comment)
- `POST /payouts/{id}/execute` → execute approved batch (idempotent)

Sample approve payload:

Auto (JSON) ▾



```
{
    "approver_id": "user_123",
    "action": "APPROVED",
    "comment": "Checked KYC, funds available"
}
```

## Identity & Security

- Use `batch_id` as idempotency key.
- Once APPROVED, only allow EXECUTE if status still APPROVED .
- Approvals must be stored with audit (who/when/what).

## Audit & Controls

- Store events in `payout_audit` with full payload.
- Maker cannot be same user as Checker for amounts above threshold.
- All approvals require digital signature / 2FA for high-value batches.

## 2) Tax / VAT / WHT Integration

### Mục tiêu

- Tự động tính và ghi nhận VAT/WHT trên settlement/payout flow.
- Sinh báo cáo thuế (VAT returns, Withholding tax remittance) và tích hợp vào GL.

### Concepts & Rules

- **VAT (Sales tax / Output VAT)**: áp dụng trên merchant fee hoặc sales depending on jurisdiction.
- **WHT (Withholding Tax)**: giữ lại phần trăm từ merchant payout and remit to authority.
- **Tax on Fees vs Tax on Gross**: configurable per country/merchant contract.
- **Taxable base**: gross transaction, fee, or net payout — configurable.

### Data model additions

Auto (SQL) ▾



```
CREATE TABLE tax_rule (
    tax_rule_id UUID PRIMARY KEY,
    country_code VARCHAR(2),
    tax_type VARCHAR, -- VAT / WHT
    taxable_base VARCHAR, -- GROSS / FEES / NET
    rate NUMERIC(5,4), -- e.g. 0.075 = 7.5%
    comportment VARCHAR, -- COLLECTED / WITHHELD
    effective_from DATE,
    effective_to DATE
);
```

```
CREATE TABLE tax_records (
    id UUID PRIMARY KEY,
    settlement_id UUID,
    tax_type VARCHAR,
    taxable_amount NUMERIC,
    tax_amount NUMERIC,
```

```

    currency VARCHAR,
    rate NUMERIC,
    created_at TIMESTAMP DEFAULT now()
);

```

## Tax calculation workflow (on settlement)

1. Settlement engine fetches applicable `tax_rule` for merchant country/date.
2. Calculates `taxable_amount` per rule (e.g., `gross`, or `fees` ).
3. Compute `tax_amount = taxable_amount * rate`.
4. If `WHT` : deduct from net payout and create `tax_withheld liability` .
5. If `VAT (collected)` : record `VAT Output` liability for platform (if platform collects) or for merchant depending on contract.

### Example (simplified):

- Gross = 100,000
- Platform fee = 2,500 (2.5%)
- WHT rate = 5% on gross → WHT = 5,000 → withheld from payout
- Net payout = 100,000 - 2,500 - 5,000 = 92,500

### Journal sample (accrual):

Auto (Java) ▾



Dr Settlement Payable (liability) 100,000  
 Cr Merchant Revenue (credit) 100,000

Dr Merchant Revenue 2,500  
 Cr Fee Income 2,500

Dr Settlement Payable 5,000  
 Cr WHT Payable (liability) 5,000

Dr Settlement Payable 92,500  
 Cr Merchant Bank (payout) 92,500

## Tax remittance flow

- Accumulate tax\_records daily/monthly by tax authority.
- Generate remittance file (CSV/PDF) with summary & invoices (if required).
- On remittance execution: create payment to tax authority, clear `WHT Payable` .

## VAT invoicing (if platform issues invoices)

- System can auto-generate invoice per settlement summarizing VAT on fees.
- Attach invoices to settlement record and send to merchant.

## Edge cases

- Reverse charge rules, exempt merchants, zero-rated supplies — rules engine must support exceptions per merchant.
- Tax treaties for cross-border WHT reduction — need validator to apply reduced rates when valid documentation (W-8BEN etc.) present.

## 3) Cross-border Settlement Engine

### Mục tiêu

- Xử lý settlement for cross-currency transactions: normalization, fees, correspondent routing, netting across currencies, and accounting for nostro/vostro flows.

### Requirements / Capabilities

- Support **multi-currency normalization** (use rate priority: network → bank → platform).
- Support **netting & batching**: net across merchants/issuers to minimize FX & fees.
- Support **correspondent bank fees** and deduct/allocate them.
- Support generation of payment instructions (MT103/ISO20022 / bank API).
- Maintain **nostro/vostro balances** and reconciliation for correspondent accounts.

### Components

- **FX Service**: rates, source priority, rate history.
- **Netting Engine**: group transactions by currency pair & counterparty; compute net positions.
- **Routing Engine**: choose payment rails (local clearing, correspondent bank, SWIFT) based on cost/availability.
- **Correspondent Fee Model**: know fee types (fixed per transfer, percentage, cover charges BOR/OUR/SHA).
- **Nostro/Vostro Ledger**: track balances and reconciliation.

### Data model additions

Auto (SQL) ▾



```
CREATE TABLE fx_rates (
    id UUID PRIMARY KEY,
    from_currency VARCHAR(3),
    to_currency VARCHAR(3),
    rate NUMERIC,
    source VARCHAR, -- NETWORK / BANK / PLATFORM
    applied_at TIMESTAMP
);
```

```
CREATE TABLE net_positions (
    id UUID PRIMARY KEY,
    counterparty_id VARCHAR,
    base_currency VARCHAR,
    counter_currency VARCHAR,
    net_amount NUMERIC,
    created_at TIMESTAMP DEFAULT now()
);
```

```
CREATE TABLE nostro_vostro_balances (
    id UUID PRIMARY KEY,
    bank_account VARCHAR,
    currency VARCHAR,
    balance NUMERIC,
    updated_at TIMESTAMP DEFAULT now()
);
```

## Cross-border flow (sequence)

Auto (Haskell) ▾



### sequenceDiagram

```
participant Recon as Reconciliation
participant Netting as Netting Engine
participant FX as FX Service
participant Routing as Routing Engine
participant Bank as Correspondent Bank
participant Ledger as Nostro/Vostro Ledger
```

```
Recon-->Netting: get unsettled cross-border txns
Netting-->FX: get_rates (priority)
FX-->Netting: rates
Netting-->Netting: compute net positions per counterparty/currency
Netting-->Routing: request best route (OUR/SHA/beneficiary)
Routing-->Bank: send payment (MT103 / ISO20022 / bank API)
Bank-->Routing: confirmation
Routing-->Ledger: update nostro balance
Routing-->Netting: result
Netting-->Recon: mark settlements executed
```

## FX & Rounding

- Normalize txn\_amount into settlement currency using selected rate, then round per rounding\_rule (configurable).

- Log applied rate + source for audit.

## Netting algorithm (high level)

1. Aggregate outgoing and incoming flows per counterparty & currency.
2. For each pair (A ↔ B) compute net = Σ(A → B) - Σ(B → A).
3. If net > 0 → create single payment A → B for net.
4. Consider thresholds to avoid tiny payments (aggregate to next window).
5. If currencies differ, apply FX conversion on net using selected rate.

## Fee cover strategies

- **OUR**: sender pays all fees (higher cost)
- **SHA**: fees shared (bank fees deducted)
- **BEN**: beneficiary bears fees  
Routing engine picks method based on merchant preference/cost.

## Correspondent / bank reconciliation

- Reconcile nostro statements from correspondent banks daily; match to sent MT103s using `instruction_id` / `endToEndId`.

## 4) GL Posting Generator (ERP integration)

### Mục tiêu

- Tự động sinh các Journal Entries (JEs) cho ERP/GL từ settlement & payout events.
- Hỗ trợ mapping configurable (per merchant/company/ledger).
- Guarantee idempotency & audit.

### Principles

- Journal entries must be **balanced** and include references to recon/settlement ids.
- Provide both **batch file** (CSV/ISO20022/ERP-specific) and **API-based posting** (REST with idempotency key).
- Keep a `posting_status` with each JE (PENDING, POSTED, FAILED, REVERSED).

### Mapping config example (JSON)

Auto (JSON) ▾



```
{
  "company_id": "COMPANY_A",
  "mappings": {
    "merchant_settlement_gross": "4000-1000",
    "fees_income": "7000-2000",
    "wht_payable": "2100-3000",
  }
}
```

```

    "merchant_payable": "2000-1000",
    "bank_account": "1000-0100"
},
"currency": "NGN",
"posting_rules": {
    "auto_post": true,
    "batch_size": 200
}
}

```

## Sample JE templates (for a settlement with WHT & fees)

**Assume: Gross 100,000; Fees 2,500; WHT 5,000; Net Payout 92,500**

### Journal 1 — Record Settlement

Auto (Java) ▾



Dr Bank (1000-0100)	100,000
Cr Merchant Payable (2000-1000)	100,000

### Journal 2 — Fees

Auto (Java) ▾



Dr Merchant Payable (2000-1000)	2,500
Cr Fee Income (7000-2000)	2,500

### Journal 3 — WHT withheld

Auto (Java) ▾



Dr Merchant Payable (2000-1000)	5,000
Cr WHT Payable (2100-3000)	5,000

### Journal 4 — Payout execution (bank transfer)

Auto (Java) ▾



Dr Merchant Payable (2000-1000)	92,500
Cr Bank (1000-0100)	92,500

## Posting modes

1. **Push to ERP via REST API** (preferred if ERP supports)
  - a. Endpoint: `POST /api/journal_entries`
  - b. Payload includes `idempotency_key = "je:{recon_id}:{batch_id}"`, lines, date, currency, attachments.
2. **Generate batch CSV** (for ERPs that accept file import)
  - a. Fields: date, ref, account\_code, debit, credit, description
  - b. Signed & uploaded to SFTP by scheduled job.

## DB tables for GL posting

Auto (SQL) ▾



```
CREATE TABLE journal_batch (
    batch_id UUID PRIMARY KEY,
    recon_id UUID,
    company_id VARCHAR,
    status VARCHAR,
    payload JSONB,
    created_at TIMESTAMP DEFAULT now()
);
```

```
CREATE TABLE journal_entry (
    id UUID PRIMARY KEY,
    batch_id UUID REFERENCES journal_batch(batch_id),
    account_code VARCHAR,
    debit NUMERIC,
    credit NUMERIC,
    description TEXT,
    posted BOOLEAN DEFAULT false
);
```

## Idempotency & reconciliation

- Each JE stores `external_post_id` returned by ERP.
- If posting fails, retry with same `idempotency_key`.
- Periodic reconciliation: compare `journal_entry` rows with ERP posted entries using `external_post_id` & amount.

## Sample REST payload to ERP

Auto (JSON) ▾



```
{
  "idempotency_key": "je:settle:8b1f- ...",
  "company": "COMPANY_A",
  "date": "2025-11-30",
  "currency": "NGN",
  "lines": [
    {"account": "1000-0100", "debit": 100000, "credit": 0, "desc": "Settlement gross"},
    {"account": "2000-1000", "debit": 0, "credit": 100000, "desc": "Merchant payable"}
  ],
  "attachments": ["s3:// ... /settlement_8b1f.pdf"]
}
```

## Error handling

- Mark `journal_batch.status = FAILED` with error payload.
- Alert finance ops if critical.
- Provide UI to re-run/rollback JE when necessary.

## Cross-cutting concerns (applies to all four modules)

### Security & Compliance

- All APIs over TLS 1.3.
- Use RBAC for maker/checker UI.
- PCI: never store full PAN; use tokenization and field-level encryption.
- Audit trail for every action (payout approval, manual JE override, tax rule change).

### Observability

- Metrics: payout success rate, approval latency, tax calculation errors, netting efficiency, GL posting success rate.
- Traces: propagate `trace_id` through ingestion → matching → settlement → payout → GL posting.

### Testing

- Unit & integration tests for fee & tax rules.
- E2E tests using synthetic data: cross-border flows, WHT scenarios, chargebacks.
- Load test netting & payout generator at projected peak throughput.

### Documentation & Ops

- SLA runbooks: what to do if payout failed, if bank returns with `INSUFFICIENT_FUNDS`, if tax

authority rejects remittance.

- Access to audit logs and attachments via Ops UI.

## Suggested Implementation Roadmap (prioritized)

1. Implement Payout Approval Workflow + DB + API + UI (Maker/Checker).
2. Add Tax Rules engine + tax\_records + tax reporting export.
3. Implement GL Posting Generator (REST & CSV modes).
4. Implement Cross-border Netting + FX Service + Routing Engine.
5. End-to-end testing + go-live on sample merchant.

## Quick checklist / cheatsheet (what to hand to devs)

- DB tables: `payout_batch`, `payout_approval`, `tax_rule`, `tax_records`, `fx_rates`, `net_positions`, `nostro_vostro_balances`, `journal_batch`, `journal_entry`.
- APIs: create/submit/approve/execute payouts; tax rule management; FX rate feed; netting trigger; generate JE endpoint.
- Config: fee rules, tax rules, payout approval limits, routing preferences.
- Monitoring: Grafana dashboards + Prometheus metrics for payout latency/success and GL posting success.

Nếu bạn muốn, mình có thể **lập sẵn** cho bạn ngay bây giờ:

- A. **OpenAPI spec** cho payout approval & GL posting APIs.
- B. **SQL migration scripts** cho tất cả bảng đã liệt kê.
- C. **Sample service code (Spring Boot)** cho approval workflow + posting to ERP.
- D. **Mermaid sequence diagrams** cho each flow as PNG/mermaid blocks ready to paste.

Chọn A/B/C/D hoặc nói "tất cả" và mình sẽ produce tiếp phần bạn muốn.