Ultimate Source Of Truth

1. Logic Flow (Ingestion Service, No Storage Phase)

a. Input:

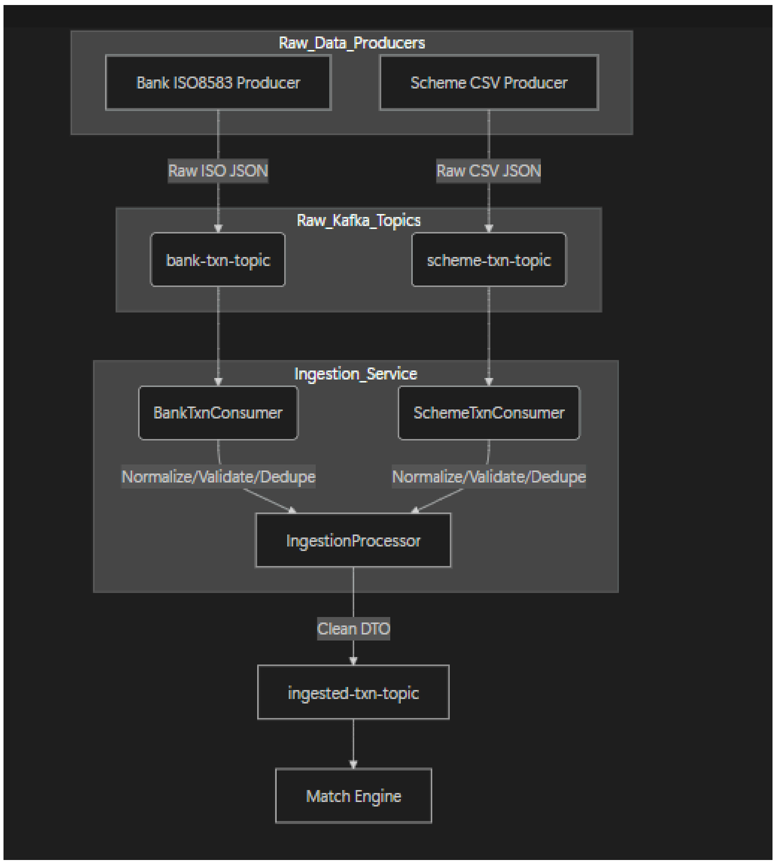
- **Source 1:** Bank/switch event (raw ISO8583, JSON, or other)
- **Source 2:** Scheme/clearing file line (raw CSV, JSON, etc.)

b. Process:

1. **Kafka Consumer receives raw message** from the appropriate topic.
2. **Normalize raw message** → `TxnRecordDTO`
   - Parse, map, validate all required fields.
   - Set source-specific fields (`sourceType`, `batchId`, `schemeName`, etc.).
3. **Validation & error handling:**
   - If invalid, log, update error metric, skip.
4. **Deduplication:**
   - Filter out duplicate transactions by txnId.
5. **Metrics update:**
   - Increment counters for total, deduped, errors, by source.
6. **Emit normalized DTO**:
   - Pass validated, deduped `TxnRecordDTO` **to the match-engine** (next microservice).
   - This is done by **pushing to a Kafka topic** consumed by the match-engine.

c. No storage is performed here—ingestion is stateless except for dedup memory/cache.

2. Mermaid Sequence Diagram



3. Service Handoff

- **What is passed on:**
  - **Type:** `TxnRecordDTO`
  - **Format:** As a JSON-serialized DTO, pushed to a Kafka topic (e.g., `ingested-txn-topic` or `match-engine-in`).
  - **Contents:**
    - All normalized fields (`txnId`, `amount`, `currency`, etc.)
    - `sourceType` (`BANK_SWITCH` or `SCHEME_FILE`)
    - Source-specific fields filled as needed (`batchId`, `schemeName`, etc.)
- **To whom:**
  - **Downstream consumer = Match Engine microservice**
    - Its job: fetch from this topic, perform matching, reconciliation, and *then* persist or further process.
- **What to know:**
  - **Ingestion service remains stateless.**
  - **No database/storage at this phase.**
  - **Metrics, dedup, and error handling all remain inside ingestion for observability only.**

This doc is the final truth for your ingestion-service phase-1.
Every engineer, PM, or reviewer can know the exact flow, what gets emitted, and what stays inside the service.

If you want a YAML/Markdown for doc, or a visual, say so—ready for copy-paste to a README or design doc.

▼ SOURCES

Field Name	Common?	BANK_SWITCH	SCHEME_FILE	Notes
txnId	✅ Common	✅ Required	✅ Required	Always present
cardNumber	✅ Common	✅ Required	✅ Required	Always present
amount	✅ Common	✅ Required	✅ Required	Always present
txnTimestamp	✅ Common	✅ Required	✅ Required	Always present
currency	✅ Common	✅ Required	✅ Required	Always present
merchantId	✅ Common	✅ Optional	✅ Optional	Usually present
terminalId	✅ Common	✅ Optional	✅ Optional	Usually present
responseCode	✅ Common	✅ Optional	✅ Optional	Usually present
channel	❌ Source-specific	✅ Often Present	🚫 *Not present/missing*	e.g., "POS", "ATM"—bank only
batchId	❌ Source-specific	🚫 *Not present/missing*	✅ Often Present	Scheme only (e.g., Visa batch/file ID)
schemeName	❌ Source-specific	🚫 *Not present/missing*	✅ Often Present	Scheme only (e.g., "VISA")
authCode	❌ Source-specific	✅ Often Present	🚫 *Not present/missing*	e.g., for authorizations
sourceType	✅ Common	"BANK_SWITCH"	"SCHEME_FILE"	Enum, always present
rawSourceRecord	✅ Common	✅ Optional	✅ Optional	Raw/original message

▼ sending

```
[Producer Java Object] DTO
      |
      | (Jackson/Gson: Object → JSON String)
      v
[JSON String]
      |
      | (StringSerializer: String → Bytes)
      v
[Kafka Topic (Bytes = JSON String)]
      |
      | (StringDeserializer: Bytes → String)
      v
[JSON String]
      |
      | (Jackson/Gson: JSON String → Object)
      v
[Consumer Java Object] DTO
```

Recon Ingestion Service: Source of Truth

1. What This Service Does

- **Acts as the "cleaning gateway"** for all transactions.
- **Consumes raw, dirty, and inconsistent transaction records** from upstream (bank switch/ISO8583 and scheme CSV).
- **Normalizes** them into a single DTO format.
- **Validates** every record (schema, constraints, required fields, data types).
- **Deduplicates** (using composite key: txnId + sourceType).
- **Feeds only clean, valid, deduped data** to the match engine via the "ingested-txn-topic."
- **Tracks metrics** for all stages (total, success, deduped, error).
- **Separation of raw and clean data** (raw topics for forensics/auditing, clean topic for matching engine)

---

3. Components & Contracts

A. Producers

- *TestTxnProducer* creates intentionally dirty and clean bank/scheme messages.
- Pushes to `bank-txn-topic` and `scheme-txn-topic`.

B. Kafka Topics

- **bank-txn-topic:** Raw ISO8583-like, JSONified by producer.
- **scheme-txn-topic:** Raw CSV, JSONified by producer.
- **ingested-txn-topic:** Clean, validated, normalized, deduped DTOs.

C. Consumers

- **BankTxnConsumer**: Reads from `bank-txn-topic`, deserializes, hands to processor.
- **SchemeTxnConsumer**: Reads from `scheme-txn-topic`, deserializes, hands to processor.

D. IngestionProcessor

- **Normalization:** Maps raw fields to internal DTO (TxnRecordDTO).
- **Validation:** Uses Jakarta Validation to enforce constraints (not null, >0, proper types).
- **Deduplication:** Only allows unique (txnId + sourceType) to pass through.
- **Metrics:** Exposes counters for monitoring.

- **Downstream Producer:** Pushes to `ingested-txn-topic` for match engine.

---

4. Key Lessons / Gotchas Fixed

- **Deduping only on txnId is WRONG.** Must dedupe on *both* txnId and sourceType (bank vs scheme).
- **Raw records are for forensics and reprocessing, not for main engine.**
- **Validation failures and deduped records are never sent downstream.**
- **All logic is service-injected, ObjectMapper as Bean, no static hackery in prod pipeline.**
- **Metrics and logging added at all key points.**
- **Easy to extend for distributed cache or persistent dedupe later.**

---

5. Reference Implementation Notes

- *BankTxnConsumer* & *SchemeTxnConsumer*:

  Constructor-inject `ObjectMapper`, hand off to `IngestionProcessor`.
- *IngestionProcessor*:
  - `ConcurrentHashMap.newKeySet()` for deduplication, composite key is `"txnId|sourceType"`.
  - All validation/dedup logic is centralized and logged.
  - **Pushes to `ingested-txn-topic` using the composite key as Kafka key.**
- *TestTxnProducer*:

  Standalone utility for simulating real traffic.
