Distributed Trade Analyzer (DTA) Project: Java Pipeline Specification

Goal: Implement the core data transformation pipeline in Java to showcase expertise in microservices architecture, fault tolerance, and stateful stream processing.

1. Feature 1: Java Core Trade Processor (MVP)

DTA-FEAT-001-JAVA: Implement Java Trade Processor (Validation, Scoring, DLQ)

Field	Specification
Service Name	`java-trade-processor-service`
Primary Technology	Java (Spring Boot / Spring Kafka)
Function	Consume raw trades, validate against business rules, calculate risk score, and route malformed messages to the DLQ.
I/O	**Input:** `DTA_INPUT_TRADES` (RawTradeEvent) **Output:** `DTA_OUTPUT_PROCESSED` (ProcessedTradeEvent) **DLQ:** `DTA_DLQ_TRADES` (RawTradeEvent)

Functional Specification & Use Cases

Area	Description / Use Case
Core Processing Logic	**Sequential Action:** Deserialize RawTradeEvent $\rightarrow$ Apply Validation $\rightarrow$ Calculate Risk Score $\rightarrow$ Serialize and Produce ProcessedTradeEvent.
Business Validation	**Negative Volume Check (DLQ Trigger):** If `volume` ≤ 0 or the message is structurally malformed, route the **original** message to the DLQ (`DTA_DLQ_TRADES`). **Risk Scoring:** If `price` < 10.00, assign `RiskScore` = "HIGH"; otherwise, assign "LOW".
Fault Tolerance (DLQ)	The service must implement a non-fatal error handler to ensure that a bad message only routes to the DLQ and **does not halt the consumption stream** for subsequent messages.

Operational Logging	Log the offset, `tradeId`, and processing duration for every successfully processed message.

2. Feature 2: Java Stateful Data Aggregator

DTA-FEAT-002-JAVA: Implement Java Stateful VWAP Aggregator

Field	Specification
Service Name	`java-data-aggregator-service`
Primary Technology	Java (Spring Boot / Kafka Streams recommended for state management)
Function	Consume validated trades, maintain continuous state, and publish the Volume-Weighted Average Price (VWAP) report in fixed time intervals.
I/O	**Input:** `DTA_OUTPUT_PROCESSED` (ProcessedTradeEvent) **Output:** `DTA_OUTPUT_REPORTS` (VWAPReportEvent)

Functional Specification & Use Cases

Area	Description / Use Case
Core Processing Logic	1. **Keying:** Process messages using the `symbol` field as the key to guarantee all related trades for a symbol are processed sequentially. 2. **Windowing:** Use a **fixed 5-second time window** to accumulate state (Total Value and Total Volume) per symbol. 3. **VWAP Calculation:** At the end of the window, calculate $\text{VWAP} = \frac{\sum(\text{Price} \times \text{Volume})}{\sum(\text{Volume})}$.
State Resilience	**Goal:** Ensure data continuity. The service must utilize Kafka Streams' built-in state store or equivalent features to guarantee that if the application crashes and restarts, the current 5-second window's accumulated volume/value is restored and processing continues from the last committed offset.
State Management	Use the `ProcessedTradeEvent`'s `symbol` and the message timestamp for keying and window alignment. The logic is stateless between windows but stateful *within* a window.
Operational Logging	Log the full contents of the `VWAPReportEvent` whenever a 5-second window is closed and the report is published.

3. Data Contract Schemas (The Shared Interface)

A. Topic: DTA_INPUT_TRADES (Contract: RawTradeEvent)

Field Name	Type	Constraints	Producer
`tradeId`	UUID (string)	Mandatory, Unique ID.	Python Injector
`symbol`	String	Mandatory.	Python Injector
`price`	Decimal (float)	Mandatory, Must be ≥ 0.	Python Injector
`volume`	Integer	Mandatory, Must be ≥ 1 for valid (DLQ Check).	Python Injector
`timestamp`	Epoch (long)	Mandatory, Time of trade execution.	Python Injector

B. Topic: DTA_OUTPUT_PROCESSED (Contract: ProcessedTradeEvent)

Input for FEAT-002, Output from FEAT-001

Field Name	Type	Description	Producer
`tradeId`	UUID (string)	Inherited.	Trade Processor
`symbol`	String	Inherited.	Trade Processor
`price`	Decimal (float)	Inherited.	Trade Processor
`volume`	Integer	Inherited.	Trade Processor
`riskScore`	String	Calculated: "HIGH" or "LOW".	Trade Processor
`validationStatus`	Boolean	Always `True` on this topic.	Trade Processor
`processedAt`	Epoch (long)	Timestamp of processing completion.	Trade Processor

C. Topic: DTA_OUTPUT_REPORTS (Contract: VWAPReportEvent)

Output from FEAT-002, Input for Python Analyzer

Field Name	Type	Description	Producer
`symbol`	String	The financial instrument.	Data Aggregator

`vwapValue`	Decimal (float)	The calculated VWAP for the window.	Data Aggregator
`windowStart`	Epoch (long)	Start time of the 5-second window.	Data Aggregator
`windowEnd`	Epoch (long)	End time of the 5-second window.	Data Aggregator