Ledger-Safe

POS event ingestion that stays correct when streams get messy.

What it demonstrates

Integration correctness: idempotency, conflict detection, quarantine.
Operational readiness: operator replay, audit trail, health metrics.

Idempotency | **Quarantine** | **Replay** | **Auditability**

Goal: make ingestion trustworthy for duplicates, retries, and conflicting corrections.

The problem

POS events are not clean, and correctness is a trust problem.

- Retries and duplicates are normal (timeouts, offline devices, retries).
- Late arrivals happen (batch sync, store reconnects, network partitions).
- Conflicting corrections happen (voids, adjustments, unstable producers).
- If ingestion silently posts the wrong thing, the ledger is corrupted.

Business impact

Double-counted revenue, broken reporting, and expensive reconciliation.
Customer impact: refunds, loyalty, and inventory all drift.

Ledger-safe ingestion

Guardrails that prevent silent corruption.

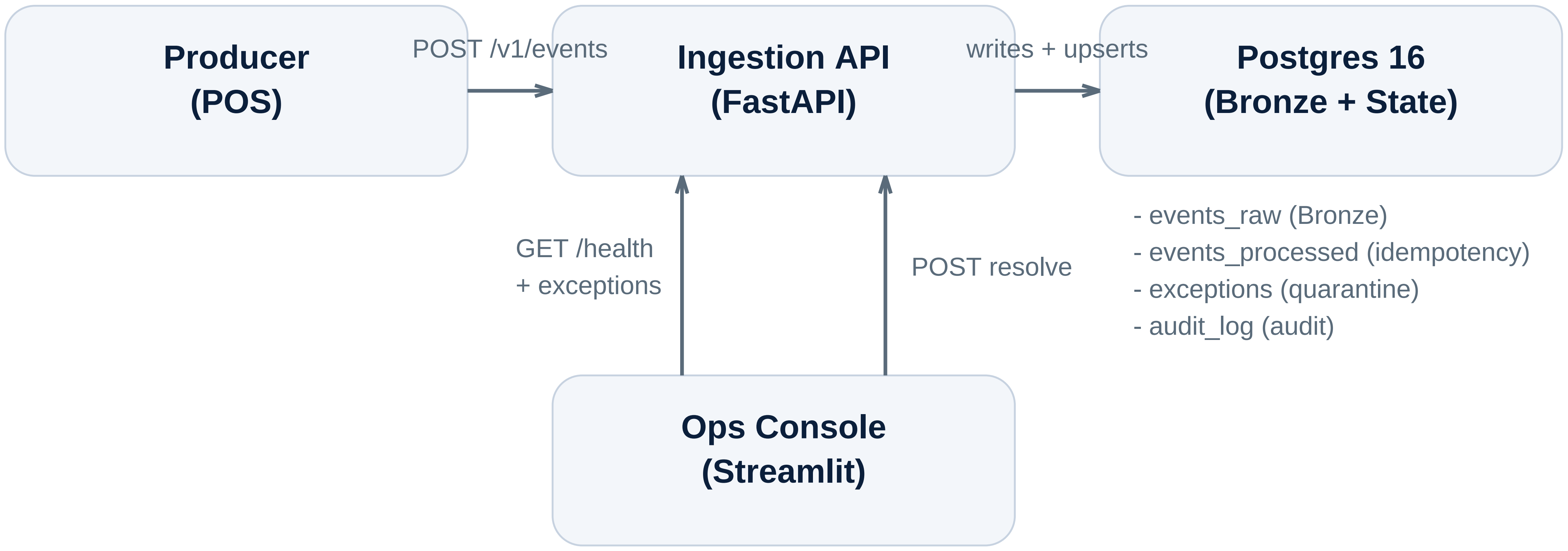
- Capture every arrival in Bronze (append-only).
- Enforce idempotency per tenant and event ID.
- Quarantine ambiguity instead of guessing.
- Enable controlled replay and backfill under audit.
- Expose health signals that ops teams can monitor.

Design principle

When you cannot be certain, do not mutate the ledger.
Quarantine makes ambiguity operational, not hidden.

Architecture

Small system, enterprise patterns.



Key guarantee

Every POST is written to events_raw first.
Then the idempotency gate decides: processed, duplicate, or quarantined.

Idempotency in practice

Exact retries become safe duplicates.

1st arrival

POST /v1/events
-> 201 processed
state: events_processed.status = processed

Exact retry

POST /v1/events (same payload)
-> 200 duplicate
state unchanged (no double-posting)

Important detail

Even duplicates are still recorded in Bronze.
You can prove what arrived and how many times.

Conflict handling

Same event_id with different payload is quarantined.

Scenario

event_id = evt-1001 arrives again
but payload hash differs from the first arrival

API outcome

POST /v1/events -> 202 quarantined
reason_code = IDEMPOTENCY_CONFLICT
exception_id created for operator triage

Why this is the point

A conflict is not a technical edge case.
It is a correctness decision that must be observable and auditable.

Operator workflow

Quarantine creates a clean, repeatable resolution loop.

- Review exception detail (raw payload + reason details).
- For conflicts: compare FIRST vs LAST payload side-by-side.
- Choose canonical event (FIRST or LAST).
- Optionally apply an override patch (JSON merge patch).
- Resolve + replay, or resolve without replay.
- Every action is written to audit_log.

Observability signals

Operational readiness is measurable.

Health endpoint

GET /v1/health returns counters:
- events_raw (volume)
- exceptions_open (risk)
- idempotency: processed / quarantined / ignored

```
{
  "status": "ok",
  "counts": {
    "events_raw": 4,
    "exceptions_open": 2,
    "idempotency": {
      "processed": 1,
      "quarantined": 1,
      "ignored": 0
    }
  }
}
```

Why it matters

These counters are what let ops teams set alerts and run incident playbooks.
Correctness is not hidden inside code. It is visible as state.

60-second demo flow

Show correctness and ops control in one minute.

Commands

```
docker compose up -d --build
demo/run-demo.ps1
UI: http://localhost:8501
```

Expected outcomes

Processed events do not double-post on retry.
Conflicts become exceptions, not ledger mutations.
Operators can replay safely with a canonical choice.

What is next

From ingestion correctness to a full ledger pipeline.

- Silver ledger tables: normalized sales, returns, and corrections.
- Gold metrics: daily net sales, store KPIs, reconciliation views.
- Expanded reason codes and validation rules.
- Per-tenant dashboards and operational SLOs.

Repo

github.com/108thecitizen/ledger-safe-pos-sim

Run locally in one command: docker compose up -d --build