Resilient Event Orchestration Gateway

The Context

At Fincart, our infrastructure sits at the heart of the digital revenue cycle for thousands of merchants. We manage the bridge between volatile courier data and real-time merchant operations. Our systems frequently encounter **massive, unpredictable traffic bursts**—such as flash sales or regional carrier outages—where tracking updates and system events flood our gateway simultaneously.

The Problem

Our current event processing must evolve. Processing high-volume events synchronously is no longer viable; it leads to event-loop blocking, latency spikes, and timed-out connections from our upstream providers. We need a system that can absorb these bursts without degrading performance or losing data.

The Goal

Your objective is to design and implement a **high-concurrency, asynchronous event gateway**. This system must act as a resilient buffer, ensuring that external event producers receive near-instant acknowledgments while internal business logic is processed reliably and at a controlled pace.

Technical Constraints & Objectives

1. Low-Latency Ingestion

- Implement a secure entry point for external events.
- **Security:** Ensure payload authenticity through HMAC signature validation.
- **Performance:** The handshake must be optimized for speed (Target: **< 150ms**). The ingestion layer should remain "thin," offloading the heavy lifting to a background layer.

2. Orchestration & Flow Control

- Implement a robust queueing strategy to manage the lifecycle of an event.
- **Concurrency:** Define a worker strategy that processes events without exhausting system resources.
- **Infrastructure:** Use a Redis-backed queueing system to manage job persistence.

3. Asynchronous Processing Logic

- **Contextual Awareness:** The system must retrieve stateful data (Active Shipments/Orders) from a persistent store (MongoDB) to inform routing decisions.
- **External Dependency Management:** Integrate with a simulated high-latency "Routing Service" (mimic a 2-second delay). Your architecture must handle this delay without stalling the rest of the pipeline.

4. Resilience & Data Integrity

- **Failure Recovery:** Implement a sophisticated retry strategy. We expect to see **exponential backoff** to handle transient failures in the Routing Service.
- **Integrity:** The system must be **idempotent**. Duplicate events are common in logistics; your system should guarantee that processing logic is only executed once per unique event.

Deliverables

1. **Repository:** A clean, production-grade codebase. We value **Modular Monolith** or **Clean Architecture** patterns.
2. **Infrastructure:** A `docker-compose.yml` that allows us to spin up the entire environment (App, Redis, MongoDB) with a single command.
3. **Proof of Load:** Provide evidence (logs or screenshots) demonstrating the gateway handling **100 simulated concurrent requests** with zero failures (including successful retries).
4. **Strategic Documentation:** A `README.md` that addresses:
    - Your strategy for **Dead Letter Queues** (DLQs).
    - How you manage **Eventual Consistency** across the system.
5. **AI Disclosure:** If you utilize Generative AI (GPT, Claude, Copilot), include a `PROMPTS.md` documenting the prompts used to shape the architecture or code.