Technical Task: Flaschenpost

Scenario: Checkout Service Redesign

Flaschenpost's checkout process has historically been part of a monolithic application. The product team now wants **to modernize this flow** to improve **performance, flexibility,** and **resilience**, particularly during high-traffic times (e.g., wine promotions during major holidays or seasonal events).

Your task is to **design and implement a core part of the new checkout service** that fits into **a microservice-oriented, event-driven architecture.**

Goals

1. Design the System Architecture

Create a high-level technical design for a **Checkout Microservice** responsible for:

- Creating an order
- Validating a cart (e.g., pricing, stock, discounts)
- Handling payment authorization
- Emitting events like `order.created`, `order.failed`, and `payment.authorized`

Design Requirements:

- Microservice vs monolith: What do you choose and why?
- How does this service communicate with others (event bus, REST,...)?
- How would you ensure reliability and idempotency in payment/order processing?
- How would you model the data (brief schema examples welcome)?
- Show error handling strategy (e.g., for payment failure or stock reservation)

Deliverable:

- A short document with:
    - System diagram
    - Component overview
    - Technology stack justification (Node.js, Python, Kafka/NATS, Redis, etc.)
    - Trade-off analysis

---

2. Implement One Critical Piece

Implement one (or multiple as a bonus) of the following — your choice:

Option A: Order Creation Service
Receives a cart payload (products, user ID, promo codes), validates it, creates an order, and emits an event.

Option B: Payment Authorization Stub
Receives a payment request, simulates contacting a payment provider, and emits success/failure.

Option C: Event Consumer
Consumes order and payment events, logs them, and updates a simple in-memory "OrderStatus" tracker.

Requirements:

- Use Node.js or Python
- Include simple error handling and logging
- Use a lightweight message broker (Kafka, NATS, or even mock PubSub)
- Testing is a bonus, but include a plan/approach

Deliverable:

- Code in GitHub
- README with:
    - Setup instructions
    - Assumptions
    - Explanation of chosen components

---

3. Technical Reflection & Leadership Approach

Write a short (~1-page) reflection answering:

- How did you approach the trade-off between performance and complexity?
- How would you make this service resilient in production?
- How would you onboard a new developer to this service?
- If you joined Flaschenpost, how would you plan the transition from monolith to microservices?
- How would you integrate AI tools into your development workflow (e.g., design, code, testing, docs)?
- If you had access to an AI assistant (like GitHub Copilot or ChatGPT), how would you delegate tasks to it, and how would you ensure its outputs are trustworthy and maintainable?
- What responsibilities (if any) could AI take on as a "team member" in your squad?

Tools Allowed

- Node.js, Python (or another mainstream backend language)
- Any diagramming tool (draw.io, Excalidraw, etc.)
- AI tools
- Anything else that fits the task

Optional Bonus

If you'd like to go a bit further, here are some ideas:

- Show how you'd handle retry logic for failed events
- Sketch a CI/CD or monitoring strategy
- Add a test (unit or integration)