Senior Engineer Technical Assessment

Objective:

This assessment evaluates your ability to design and implement a microservice in Go that integrates with multiple payment gateways. The service should manage deposit (cash-in) and withdrawal (cash-out) transactions, support different data interchange formats, handle asynchronous callbacks, and be designed with the anticipation that the number of external payment gateways will grow.

Estimated Time to Complete:

6-8 hours (with the option to extend to a maximum of 10 hours)

Problem Statement

You are tasked with building a payment gateway microservice in Go that integrates with two distinct payment gateways. More gateways will be added in the future. The service should manage deposit and withdrawal operations, handle asynchronous callbacks for transaction updates, and remain resilient to issues arising from external payment gateways. Moreover, the service should be designed to easily accommodate future integrations and adapt to evolving product requirements.

Requirements

1. Payment Gateways Integration

- **Gateway A**: Uses JSON over HTTP.
- **Gateway B**: Uses SOAP/XML over HTTP.

Implement integration for both gateways, handling deposit and withdrawal requests, and managing responses. Feel free to use mock services for the payment gateways.

2. Microservice Features

- **Deposit (Cash-in)** and **Withdrawal (Cash-out)**:
  - Accept and validate requests for deposits and withdrawals.
  - Forward requests to the correct gateway and handle the responses.
  - Store transaction details and update statuses accordingly.

- **Asynchronous Callbacks**:
  - Implement endpoints to handle callbacks from the payment gateways and update transaction statuses.

- **Extensibility**:
  - Design the service to be easily extensible, anticipating the integration of additional payment gateways in the future.

- Implement a flexible architecture that can accommodate different protocols and data formats (e.g. ISO8583 over TCP).
- Ensure that adding new gateways or adapting to evolving product requirements requires minimal changes to the existing codebase.

- **Resilience to Gateway Issues**:
  - Ensure the service remains operational even if one or more payment gateways experience outages, slowdowns, or other issues.
  - Implement strategies such as circuit breakers, timeouts, and retries to manage gateway failures.
  - Ensure that failures or delays in any gateway do not affect the overall availability and performance of the microservice.

- **Transaction Logging**:
  - Log all transactions, ensuring sensitive data is securely handled and masked.

- **Error Handling**:
  - Gracefully handle common errors such as timeouts, invalid responses, and service unavailability.
  - Return meaningful error messages and appropriate HTTP status codes.

- **Scalability and Fault Tolerance**:
  - Design the service with scalability and fault tolerance in mind.

3. Architecture Design

- Design the microservice to run on GCP or similar cloud providers.
- While architecture design is important, the primary focus is on Go expertise, systems design, and the ability to create a robust, future-proof solution.

4. Testing

- Implement unit tests for key service components, especially the gateway integrations.
- Provide instructions on running these tests.

5. Documentation

- Include a README with:
  - A high-level overview of the architecture.
  - Instructions on how to build, run, and test the service.
  - API documentation (preferably in OpenAPI format).

exinity

Time Management Guidance

We understand that the requirements are ambitious for a 6-8 hour timeframe. You are encouraged to spend no more than 10 hours on this project. Focus on delivering a functional, high-quality implementation for the key requirements. If you are unable to complete all parts within the time limit, prioritize what you believe to be the most critical aspects of the project and document what could be improved or added with more time.

Deliverables

1. **Code**:
    - A GitHub repository (or ZIP file) containing the source code.
    - A `Dockerfile` and `docker-compose.yml` for containerization.

2. **Design Document**:
    - A brief document explaining your design decisions, with a focus on how the service is designed to be easily extensible to accommodate additional payment gateways and evolving product requirements.

3. **API Documentation**:
    - An OpenAPI specification documenting the API endpoints.

4. **Instructions**:
    - A `README.md` file with setup, run, and test instructions.

Evaluation Criteria

1. **Go Expertise**:
    - Code clarity, maintainability, and proper use of Go idioms.

2. **Systems Design**:
    - Soundness and scalability of the microservice architecture.
    - Robust error handling, edge-case management, and resilience to external gateway issues.

3. **Extensibility and Future-Proofing**:
    - How well the microservice is designed to accommodate the addition of new payment gateways and evolving product requirements with minimal changes.
    - The flexibility of the architecture to support a variety of current and future data formats and protocols, including specialized standards like ISO8583.

4. **Architecture**:
    - Thoughtful choice of services, with justification in the architecture document.
    - The robustness of the design in handling gateway outages or slowdowns.

5. **Testing**:
    - Coverage and effectiveness of unit tests.
    - Simplicity and clarity in running the tests.

6. **Documentation**:
    - Clarity and completeness of the README and API documentation.

7. **Time Management**:
    - How well you managed the 10-hour limit, and the quality of work delivered within this timeframe.