Dino Ventures: Backend Engineer Assignment

Internal Wallet Service

1. Problem Statement

Build a **wallet service** for a high-traffic application like a gaming platform or a loyalty rewards system. This service keeps track of each user's balance of **application-specific credits or points** (for example, "Gold Coins" or "Reward Points"). It is a **closed-loop system**, meaning these credits only exist and can only be used inside our application—they are not real money, not crypto, and cannot be transferred between users like in a payment app. Even though the currency is virtual, **data integrity is extremely important**: every credit added or spent must be recorded correctly, balances must never go negative or out of sync, and no transactions can be lost. For example, if a user earns 100 reward points for completing a game level and later spends 30 points to buy an in-game item, the ledger must reliably record both actions and always show the correct remaining balance of 70 points, even under heavy traffic or system failures.

2. Core Requirements

The solution **must** meet the following functional and non-functional requirements to be considered.

A. Data Seeding & Setup

You must provide a script (e.g., seed.sql or a migration file) that initializes the database with the following data so we can run your code immediately:

1. **Asset Types:** Define the application assets (e.g., "Gold Coins", "Diamonds", "Loyalty Points").
2. **System Accounts:** Create at least one "System" wallet (e.g., a "Treasury" or "Revenue" account) to act as the source/destination for funds.
3. **User Accounts:** Create at least two users with initial balances.

B. API Endpoints

You must expose RESTful endpoints to execute **transactions, check balance, etc**.

C. Functional Logic

Tech Stack: You are free to use **any backend language** (Go, Java, Python, Node.js, Rust, etc.) and **any relational database** (PostgreSQL, MySQL, SQLite, etc.) that supports ACID transactions.

Core Task: The service must handle the following **three specific flows** transactionally:

1. **Wallet Top-up (Purchase):** A user purchases credits using real money. The system must credit their wallet (Assume a fully working payment system already exists).
2. **Bonus/Incentive:** The system issues free credits to a user, such as a referral bonus.
3. **Purchase/Spend:** A user spends their credits to buy a service within the app.

D. Critical Constraints (The "Hard" Part)

1. **Concurrency & Race Conditions**
2. **Idempotency**

E. Deliverables

- Source code (GitHub link or zipped folder).
- seed.sql / setup.sh: A script to insert the pre-seed data.
- A README.md explaining:
  - How to spin up the database and run the seed script.
  - Your choice of technology and why.
  - Your strategy for handling concurrency.

3. Brownie Points

While the requirements above constitute a passing grade, we look for engineering excellence. Implementing the following will score you extra points.

🌟 **Deadlock Avoidance:** Implement a mechanism to avoid database deadlocks under high load.

🌟 **Ledger-Based Architecture:** Instead of simply updating a balance column, implement a double-entry ledger system for auditability.

🌟 **Containerization:** Include a Dockerfile and docker-compose.yml to automatically spin up the app, database, and seed script.

🌟 **Hosting:** Deploy the application to a public cloud provider and provide the live URL.