KenTech

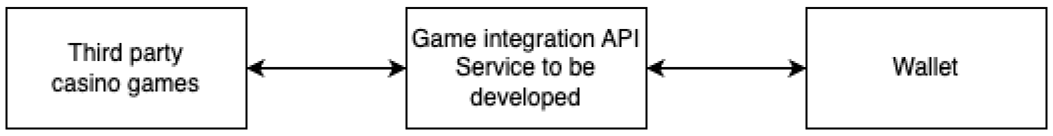
Backend Go Developer Technical Assessment

Overview

Your primary task is to develop a **Game Integration API** that facilitates third-party casino games on our platform. This new service is crucial for handling all financial transactions related to gameplay, with two key responsibilities:

1. **Managing user balances** through interactions with an existing, somewhat unreliable, backend service.
2. **Creating and updating bets** dynamically based on endpoint calls.

System Context



Wallet Service Integration

To streamline development, we've provisioned a mock wallet service with pre-configured users and initial balances:

- **ID**: 34633089486, **Currency**: "USD", **Balance**: $5,000.00
- **ID**: 34679664254, **Currency**: "EUR", **Balance**: €9,000,000,000.00
- **ID**: 34616761765, **Currency**: "KES", **Balance**: KSh 750.50
- **ID**: 34673635133, **Currency**: "USD", **Balance**: $31,415.25

Important Note: This in-memory service **does not persist with data**. All transactions and balance changes will be **lost upon service restart**, and balances will revert to their initial states.

You can access the Wallet Service Docker image: docker.io/kentechsp/wallet-client

Use the following token to authenticate requests with the wallet service: **Wj9QhLqMUPAHSNMxeT2o**.

Once the Docker image is running, you can view the API documentation via Swagger at: http://localhost:8000/swagger/index.html.

Requirements

The Game Integration API needs to expose five RESTful endpoints:

1. **Authentication**: Authenticates a player attempting to play a game.
    a. **Receives**: User credentials (username, password).
    b. **Returns**: JSON Web Token (JWT) for subsequent requests.
2. **Player Information**: Retrieves essential player details.
    a. **Receives**: User token (JWT).
    b. **Returns**: user id, balance, and currency.
3. **Withdraw**: Processes a withdrawal from a player's balance. Each request to this endpoint should be treated as a **bet placement action**.
    a. **Receives**: User token (JWT), single transaction details including currency, amount, and provider transaction id.
    b. **Returns**: A unique transaction ID from our system, the provider transaction id, the old balance, the new balance, and the transaction status.
4. **Deposit**: Processes a deposit into a player's account. This request represents a **bet settlement action**, and the bet's status must be determined by the transaction amount: if the amount is zero, the bet is LOST; otherwise, the bet is WON.
    a. **Receives**: User token (JWT), single transaction details including currency, amount, provider transaction id, and provider withdrawn transaction id.
    b. **Returns**: A unique transaction ID from our system, the provider transaction id, the old balance, the new balance, and the transaction status.
5. **Cancel**: Reverts to a previously processed transaction.
    a. Receives: User token (JWT), Provider transaction ID which should be rollbacked.
    b. Returns: Unique transaction ID from our side, provider transaction id, old balance, new balance and transaction status.

Must-Haves

Your solution must demonstrate proficiency in the following areas:

- **Language**: Implemented entirely in **Go**.
- **Database**: Data should be stored in a database of your choice.

- **Architecture**: Follows **Clean Architecture** principles.
- **Code Quality**: Simple, readable, and maintainable code.
- **Security**: Includes robust **authentication**, **authorization**, and mechanisms to **prevent SQL injection**.
- **Version Control**: Project uploaded to a **Git repository**.
- **Configuration**: Utilizes **environment variables** for all necessary configurations.

Nice-to-Haves

Consider incorporating the following to showcase a more comprehensive solution:

- **ORM**: Use of an Object-Relational Mapper.
- **Error Handling**: Comprehensive and graceful error handling.
- **Logging**: Effective logging for monitoring and debugging.
- **Deployment**: Clear deployment instructions.
- **Containerization**: Docker support (e.g., docker-compose) for automated setup and execution.
- **Testing**: Unit, integration, or end-to-end tests.
- **API Documentation**: Clear and concise API documentation (e.g., OpenAPI/Swagger).