🔧 Octoco Backend Design Quiz – Intermediate Level

Objective:
Design a backend system that manages user accounts with deposit and withdrawal functionality using a third-party payment provider. The system must be secure, consistent, and resilient to concurrency issues.

📘 Overview

You're required to write a **system design document** for a backend service that allows users to deposit and withdraw funds using Revio. The service must track and manage each user's balance accurately and safely.

This task is meant to assess your **backend design skills**, including architecture, data modeling, webhook handling, and system security.

You must write this as a document, not as code. Assume you are working with a senior frontend developer who can implement any frontend UI or client-side logic you require. Focus on how you would design and build the **backend system**.

📜 Your Deliverable

Produce a **detailed design document** (PDF, Markdown, or Google Doc) containing:

1. **Tech Stack**
   List the technologies you would use (language, framework, database, hosting, etc.) and why.

2. **System Architecture**
   - Describe the architecture in broad strokes (e.g., monolith vs microservices).
   - Include a **diagram** of how components interact.
   - Describe how you would **authenticate and authorize** users.

3. **Database Design**

- Describe your schema: users, transactions, balances, and any other relevant entities.
- Indicate indexes and any constraints.

4. **Deposit & Withdrawal Flow**
    - Describe the flow for each operation from the user's perspective and the backend's.
    - Show how deposits are initiated and how **Revio webhooks** are handled.
    - Include flowcharts or sequence diagrams where appropriate.
5. **Webhook Handling**
    - Show how you verify and trust incoming webhooks.
    - Indicate where you store webhook delivery logs or audit trails.
6. **Security Considerations**
    - Include strategies for:
        - Authentication and authorization.
        - Protecting against replay attacks or spoofed webhooks.
        - Storing sensitive data securely (e.g., API keys, tokens).
        - Preventing common issues like SQL injection or privilege escalation.
7. **Deployment & Hosting**
    - Briefly describe how you would deploy and run the system (e.g., cloud provider, CI/CD pipeline, containerization).
    - Consider environment management (staging vs prod), logging, monitoring, and alerting.

8. **Potential Failure Points & Mitigations**
    - List edge cases or risks (e.g., duplicate transactions, Revio downtime).
    - Describe how you would detect and recover from failures (e.g., retries, dead letter queues).
9. **Bonus (Optional)**

- How would you expose a dashboard for users to view their transaction history?
- How would you scale this system if usage grew 100x?

🧠 Tips for Candidates

- Think like an engineer building a production system, not just a prototype.
- Diagrams go a long way.
- Use real terminology: e.g., **serializable isolation**, **pessimistic locking**, **JWT**, **event sourcing**, etc.
- Show tradeoffs in your decisions.
- Don’t over-engineer – start with a solid base and explain how you'd evolve it.