BYRON OCHARA

Nissmart - FULL-STACK DEVELOPMENT ASSIGNMENT

Project Name: TRANSACTION SYSTEM LEDGER

System overview:

The system is a transaction ledger which consolidates and stores all the user's various ways of handling funds; which include depositing funds, withdrawing funds or transferring funds to another user in a digital format that can be and stored in a database.

1. ARCHITECTURE COMPONENTS:

DATABASE TABLES:

To record the various ways of handling funds, the database is designed with key tables to capture user information for every step they desire to implement.

The tables includes;

1. User Table
   This table stores all the users that would be doing a particular form of handling funds.
2. Wallet table
   The table records all the user's available funds before they do any operation. The idea here is, the user must have some funds before doing any kind of operation. Otherwise, they would not need to deposit funds or withdraw funds or transfer funds if their wallet balance has no funds.
   A user must have ONLY one wallet.
3. Transactions Table
   This table is the ledger. It records all the user operations and their statuses.
   The operation type could be, deposing funds or withdrawing funds or transferring funds.
4. Transfer table
   This table is junction table.
   It records transferring funds between two users where one is the sender and the other as the receiver, including other information that relates the two users.

API DESIGN:

The database tables are accessible throng API endpoints to record every operation being done by the user. They include

User API endpoints:

a. POST: /users/addUser - To create a new use
b. GET: /users/:id – To get user details
c. GET: /users - To view all the users

Wallet API endpoints:

a. GET: /wallet/:id – to view wallet details including its current balance

Transaction API endpoints:

a. GET: /transaction/:id – To retrieve all the transactions that belongs to a user as their history details.
b. POST: /deposit - To record a deposit transaction
c. POST: /withdraw - To record a withdraw transaction

Transfer API endpoint:

a. POST: /transfer - To record a transfer between to two users

TRANSACTION SAFETY:

For the records to be stored in the database in a clean format, the user needs to carry out the operations with clear common sense that is ideally expected in the real-world scenario. Which include;

1. Avoiding negative balance
   a. The user is not expected to do an operation like trying to withdraw the amount higher than the balance available in their wallet.
      That is to say, you cannot withdraw funds that are not available in your wallet since you do not have the funds. Hence, deposit first or ask to be transferred funds to your wallet, i.e. send money.

   b. Validation will be created on the UI to show the user keys in the right expect amount according to the type of operation they wish to transact to ensure correctness of the system flow.

2. Avoid double-spend
    a. After a transaction has been initiated, it is recommended for the system to not take any transaction of the same amount before they are validated and completed to ensure correctness of the flow process.
3. Idempotency and atomicity
   These go hand in hand as per how the system is architected such that one transaction once initiated by the use must flow all the way to the end and completed instead of switching to another transaction type.
   Example: If the user wishes to withdraw funds, and have less funds, do not switch to deposit unless they explicitly choose to deposit funds if they even have funds to deposit.
4. Validation
   Validation should be done on the front-end to alert the use before initiating unexpected transactions and also on the backend in case the defined logic does not meet the expected flow.

ERROR HANDLING:

Errors arising from the system will require different approaches, that may include;

a. If error from backend due to wrong inputs, alert the user to key correct values.
b. Failed withdrawals may arise from wrong inputs where users try to withdraw funds higher than their wallet balance, hence they will be marked as pending transactions, and internal employees shall take necessary actions after user visits the service provider.
c. Failed transfers may arise from system down time or upgrade, where the user will be notified to either wait or visit provider for further processing.
d. Backend errors will be identified and monitored by the developers and expected action taken to ensure the system operates as expected.

ASSUMPTIONS AND TRADE-OFFS:

The system should operate in a manner that makes an impact and provide value to the users.
The providers should continue to monitor and maintain the system to ensure efficiency with as the users continue to use the system.

Hopefully given a chance to join you, we shall build beautiful products that add value and grow together as a society levering the power of technology where my expertise lies.