BRIEF TECHNICAL DOCUMENTATION FOR WALLET SYSTEM - BY AJIBOLA OKESOLA

TABLE OF CONTENT

1.0 PROJECT OVERVIEW AND SCOPE

1.1 PROJECT OVERVIEW:

The Wallet System is a financial application designed to manage user wallets, allowing for secure credits and debits

1.2 DOCUMENT PURPOSE:

This document serves as a brief technical guide for Fincra, outlining the architecture, design decisions, and implementation details of the wallet system.

1.3 PROJECT SCOPE AND REQUIREMENTS:

The following is the specified scope and requirements of the Wallet system project

- Build with basic functionality to credit and debit a wallet account.
- Use node.js and Typescript
- Create a hosted API backend URL for testing
- Take into consideration all necessary guards against financial exploitations like race conditions, deadlocks, stable balance, and more.
- Share the GitHub repository, and ensure it is public.

1.4 OUT OF SCOPE LIBERTIES

Due to lack of specification, the following requirements will be implemented

- The Wallet system will be assumed to operate in one currency
- Users will have to SignUp and Sign in to access Wallets
- Wallets can only be debited from (and viewed by) the owning users

2.0 TECHNOLOGIES USED AND JUSTIFICATION

1. Node.js:
    - It was requested in the project requirements
    - Node.js is built on an asynchronous, event-driven architecture, making it highly efficient and scalable, especially for I/O-bound operations typical in financial systems
2. TypeScript
    - It was requested in the project requirements
    - TypeScript adds static typing to JavaScript, reducing runtime errors and improving code quality by catching type-related errors during development
3. Express.js

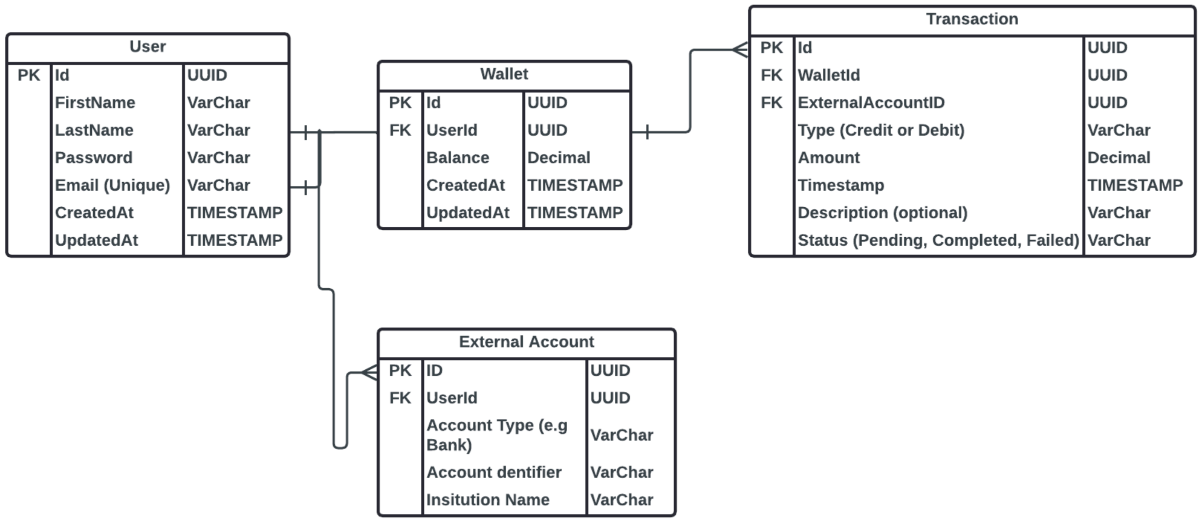
    - it is a minimal and flexible framework that supports middleware, which can be used to handle various tasks such as logging, authentication, and input validation
4. PostgreSQL
    - It is fully ACID-compliant, ensuring reliable transaction processing, which is crucial for financial applications
    - PostgreSQL's Multi-Version Concurrency Control (MVCC) efficiently handles concurrent transactions, reducing the risk of data conflicts and race conditions
5. Sequelize ORM
    - It simplifies database interactions, ensure transaction support, and manage complex data relationships efficiently

2.1 OTHER SECURITY CONCERNS AND JUSTIFICATION

1. **PostgreSQL was chosen over MongoDB:** Although MongoDB 4.0 now newly supports ACID, PostgreSQL was chosen because of some special advantages for financial transactions
    a. MongoDb uses document-level locking which is not as effective as handling concurrent issues as the row-level locking for PostgreSQL
    b. PostgreSQL's advanced query optimization, indexing, and support for complex joins make it ideal for transactional queries and reporting
    c. PostgreSQL's mature support for ACID transactions ensures data integrity and consistency, which are critical for financial operations
2. **Double Entry Accounting**: Both credit and debit transactions will be recorded for each transaction, external accounts will also be conceptualized in the database to make the transactions balanced
3. **Transaction timeouts:** This will aid deadlock resolution
4. **Authentication and Authorization:** Implemented using JWT to address security concerns
5. **Transactions will be Serialized**
6. **Row-Level Locking**: this will be used during updates as well to prevent concurrent modifications

3.0 SYSTEM ARCHITECTURE

3.1 DATABASE DESIGN CONCEPT



User		
PK	Id	UUID
	FirstName	VarChar
	LastName	VarChar
	Password	VarChar
	Email (Unique)	VarChar
	CreatedAt	TIMESTAMP
	UpdatedAt	TIMESTAMP

Wallet		
PK	Id	UUID
FK	UserId	UUID
	Balance	Decimal
	CreatedAt	TIMESTAMP
	UpdatedAt	TIMESTAMP

Transaction		
PK	Id	UUID
FK	WalletId	UUID
FK	ExternalAccountID	UUID
	Type (Credit or Debit)	VarChar
	Amount	Decimal
	Timestamp	TIMESTAMP
	Description (optional)	VarChar
	Status (Pending, Completed, Failed)	VarChar

External Account		
PK	ID	UUID
FK	UserId	UUID
	Account Type (e.g Bank)	VarChar
	Account dentifier	VarChar
	Insitution Name	VarChar

A User has one Wallet, and multiple transactions can be associated with the wallet

3.2 API ENDPOINTS

1. **POST /signUp:** Create a new User and an associated Wallet automatically
2. **POST /login:** Login to user account
3. **GET /wallet**: Retrieve wallet details.
4. **POST /wallet/credit**: Credit a specified amount to the wallet.
5. **POST /wallet/creditAnother**: Credit a specified wallet with the user's wallet.
6. **POST /wallet/debit**: Debit a specified amount from the wallet.

4.0 OTHER LINKS

1. **Postman Documentation:**
   https://documenter.getpostman.com/view/13234512/2sA3QsBYSZ
2. **GitHub link:** https://github.com/A-jibola/OkWallet
3. **Api Endpoint for Testing:** https://okwallet.onrender.com