Testudo Bank Sub Accounts Feature

Owner: Jefferson Taguba (jefferson.taguba@gmail.com)

Problem Statement

Customers of Testudo Bank are currently limited to managing a single account. For those customers who wish to create accounts for their children who are minors this means they do not have the ability to directly monitor or transfer funds into their childrens' accounts directly from their own parent account. The accounts created for their children are separate and independent and cannot be managed from a single logon to the parents' account.

Customers would benefit from being able to create sub accounts that can be managed from within the parent account. For customers this means that should they want to give allowance to their children they can do so through the security of a checking account. Their children would not have cash allowance on hand and the theft risks associated with that. For Testudo Bank this means that instead of having parents withdraw cash to give to their children they can simply transfer funds into their childrens' sub accounts so the money remains in the bank.

Solution Requirements

- Customers should be able to create sub accounts from their account by providing new account information.
- Customers should be able to view the balance as well as transfer money in and out of their sub accounts.
- The sub account should not have access to view or transfer to or from the parent account.
- Any transfers of money should be logged into the transaction history.
- Customers should be notified on their account if any sub account has a balance less than an amount they choose.

Solutions Considered

Account Includes New Sub Account Architecture Approach

In this approach, accounts are modified to create new sub account type objects. These accounts have different features to distinguish them from standard accounts as stated in the solution requirements.

Account References Standard Accounts as Sub Accounts Approach

In this approach, sub accounts share the same object type as standard accounts. However, the structure of an account is modified such that it contains **references** to other standard accounts that will serve as its sub accounts. References are stored as **mappings** that use account numbers for keys and names and/or passwords for values. This allows the parent account to have the login information to view and change the balance of a sub account using **existing functions**. Any transfers of money into a sub account will reflect the same amount deducted from the parent account (and vice versa).

Pro/Con of all approaches considered:

Account Includes New Sub Account Architecture Approach

Pros:

- New sub account type guarantees different functions can be implemented to prevent sub accounts from accessing parent accounts.
- With sub accounts being objects created within the parent account, they are not independent and can communicate with the parent account to report information like low balance.

Cons:

- Requires changing the structure of the account object to be able to create new sub account objects.
- Requires creating a new sub account object with new specific functions that replicate most functions of parent accounts. It would just not have the ability to create its own sub accounts.

Account References Standard Accounts as Sub Accounts Approach

Pros:

- Changes the structure of the account object minimally by
  - Creating new accounts.
  - Storing references of logon information for the new accounts.
- Uses existing account functions to view and change account balance.

Cons:

- Sub accounts are independent accounts and have no way of communicating directly with parent account.

Proposed Solution

The proposed solution is the **Account References Standard Accounts as Sub Accounts Approach** because it requires minimal change to the structure of the account object type. Having sub accounts still use the standard account object type means that the solution requirements for the parent over sub accounts can be met by using account functions that already exist. By storing references to logon information for sub accounts, parent accounts can view and make changes to the balances of those sub accounts.

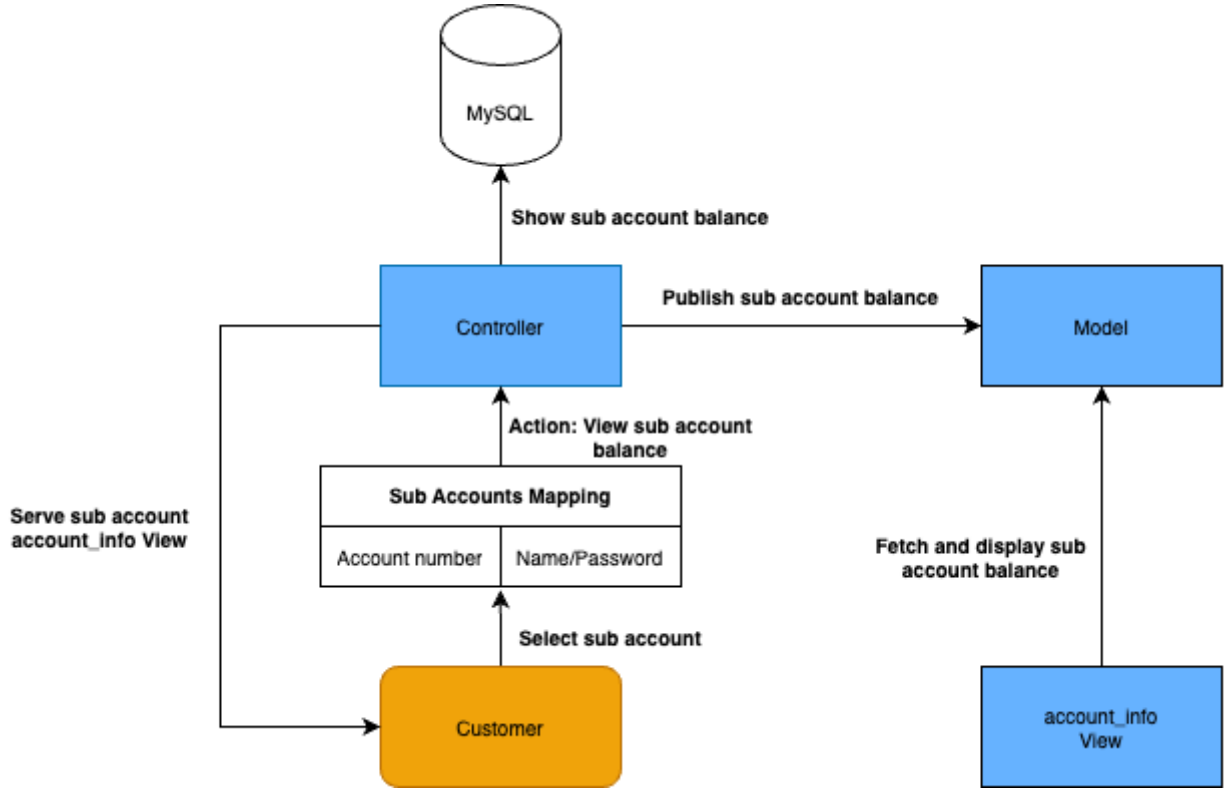
Although the sub account cannot communicate a low balance to the parent account (since it has no reference to the parent account) the parent account can always view the balance of the sub account and check it against the desired low balance amount.

Minimal changes to the account object type as well as the fulfillment of requirements through existing account functions means a Minimal Viable Product (MVP) can be all the more quickly developed by Testudo Bank Engineers.

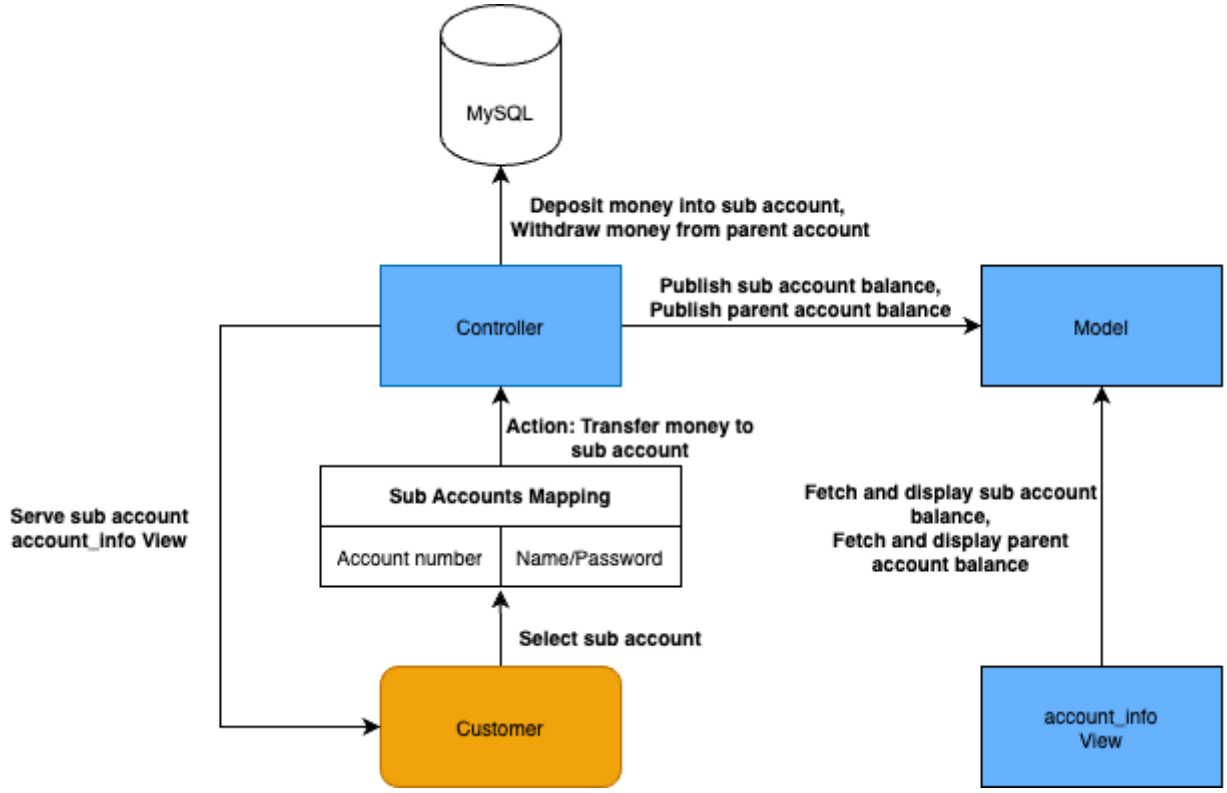
Technical Architecture

MVC Logic Diagrams

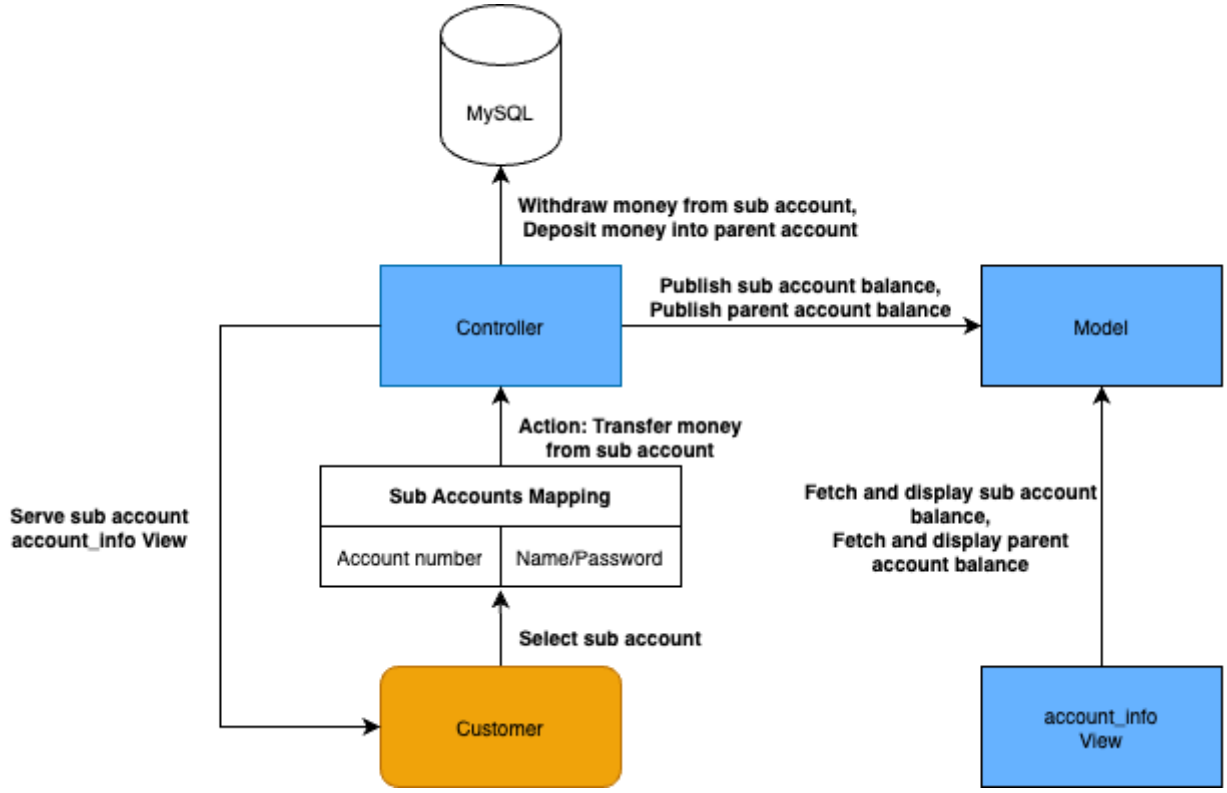
View Sub Account Balance



Transfer into Sub Account



Transfer from Sub Account



MySQL DB Schema

DB Schema Notes

- `CustomerID` from `Customers` table has a **one-to-many** relationship with `CustomerID` in both `CryptoHoldings` and `CryptoHistory` table. This is because a single customer can hold more than one Cryptocurrency (eventually, when we expand beyond just Ethereum) and will buy/sell more than once (which means more than just 1 record in `CryptoHistory` table).
- `Action` in `CryptoHistory` is limited to the Strings `"Buy"` or `"Sell"`.
- `CryptoName` in `CryptoHoldings` and `CryptoHistory` will be limited to just `"ETH"` for now.
- `CryptoAmount` is a `float`, which diverges from the USD-to-pennies (as `int`) paradigm that the rest of the codebase follows. Just use the Cryptocurrency value returned by the JSoup API.
- Two new allowed `Action` values must be added for `TransactionHistory`: `CryptoBuy` and `CryptoSell`. When a customer buys a Cryptocurrency, the reduction in the customer's main balance must be reflected in `TransactionHistory` with a new `CryptoBuy` record. Whenever a customer sells a Cryptocurrency, the US Dollar value of the sell should be logged in `TransactionHistory` as a new `CryptoSell` record. Additionally, any Overdraft Repayment that occurs when selling a Cryptocurrency must be logged in `OverdraftLogs` table.
    - **TIP:** Simply use the existing `submitDeposit()` and `submitWithdraw()` handlers to handle all this logic. The only new code that needs to be added is to make sure the `CryptoBuy` and `CryptoSell` actions are used. You can see an example of how this is implemented in `submitTransfer()`.

Simple, Error, and Edge Cases

Simple Cases

- A simple Crypto Buy when the customer is not in overdraft, and the US Dollar amount for the buy would not bring the customer into overdraft.
- A simple Crypto Sell when the customer is not in overdraft.

Error Cases (return `welcome.jsp` page in all of these cases)

- Customer attempts to Buy Crypto when the US Dollar amount for the buy exceeds their main balance. They should not be allowed to go into overdraft to buy Crypto.
- Customer attempts to Buy Crypto while currently in overdraft.

Edge Cases

- Customer Buys Crypto safely (like the Simple Buy case above). Then, they withdraw enough from main account that they go into overdraft. Then, the customer Sells their Crypto holdings. Ensure that the Crypto Sell first pays off the Overdraft Balance, **and any excess goes into main account balance**. (Again, this logic is already largely implemented for you in `submitDeposit()`, so just call that method in your new `submitCryptoSell()` handler.