Testudo Bank Transaction History Feature

Owner: Ananya Ramkumar (ananyaramkumar06@gmail.com)

Problem Statement

Customers of Testudo Bank are currently limited in managing their spending because Testudo Bank does not provide a transaction log for them to see their deposits and withdrawals. Also, in the event of fraud, customers do not have a way to dispute and revert transactions.

Customers would benefit from a Testudo Bank feature that enables them to view their last few transactions and undo (dispute) a transaction if need be. Creating this feature would grow our customer base because customers can now feel assured that their money is safe from fraud in TestudoBank.

Solution Requirements

- Customers should be able to see a log of their last 3 transactions (deposits or withdrawals) when they log into their account
- The transaction log should specify the date and time of the transactions as well as the action (deposit or withdrawal) and $ amount.
    - Ex. *Tuesday, October 13th at 4:13 PM: $15 was deposited.*
- Customers should be able to dispute a transaction that displays in the log, meaning they can revert the transaction and their balance will reflect this dispute.
    - Ex. Let us say that a customer's transaction log shows that they recently withdrew $500. If the customer disputes this claim, then Testudo Bank will automatically add $500 back to their balance.

    - **This reversal also needs to be reflected in the Transaction Logs.** For example, if a customer disputes a withdraw of $500, then you will need to add a **deposit $500** transaction log entry.
- If a customer reports fraud on more than two withdrawals, a hold will be placed on their account and no further transactions (deposit or withdrawal) can take place.

Proposed Solution

The proposed solution is a transaction history feature that allows customers to keep track of their transactions as well as dispute any potential fraud on their account. Customers will be able to view a list of their most recent transactions in the existing account_info page. Customers can also click a revert button to undo a recent transaction.

This solution was chosen because it re-uses existing pages in the application, and does not require any major restructuring to the underlying MySQL DB.

Pro/Con of all approaches considered:

Reversal Approach
(Approach already described above.)
Pros:

- No new pages/forms are added. Existing pages and forms are minimally changed.
- Gives the customer power/freedom over their own money
  - Typically, Banks require that a customer call the Bank and go through a multiple month-long process to report a fraudulent transaction
    - Simplifies development since all of the workflow stays in the `account_info` page and DB Schema

Cons:

- Depending on the implementation, would likely require another table for Transaction History that would store transaction information as rows
- This approach isn't as robust as the normal approach to fraud transaction reporting because there is no system to detect false claims of fraud

Account Lock Approach

This approach just freezes the customer's account as a means to prevent further fraud.

Pros:

- No new pages/forms are added. Existing pages and forms are minimally changed.
- Easy to implement because it just requires a check to see if a customer's account is frozen when a customer tries to log in

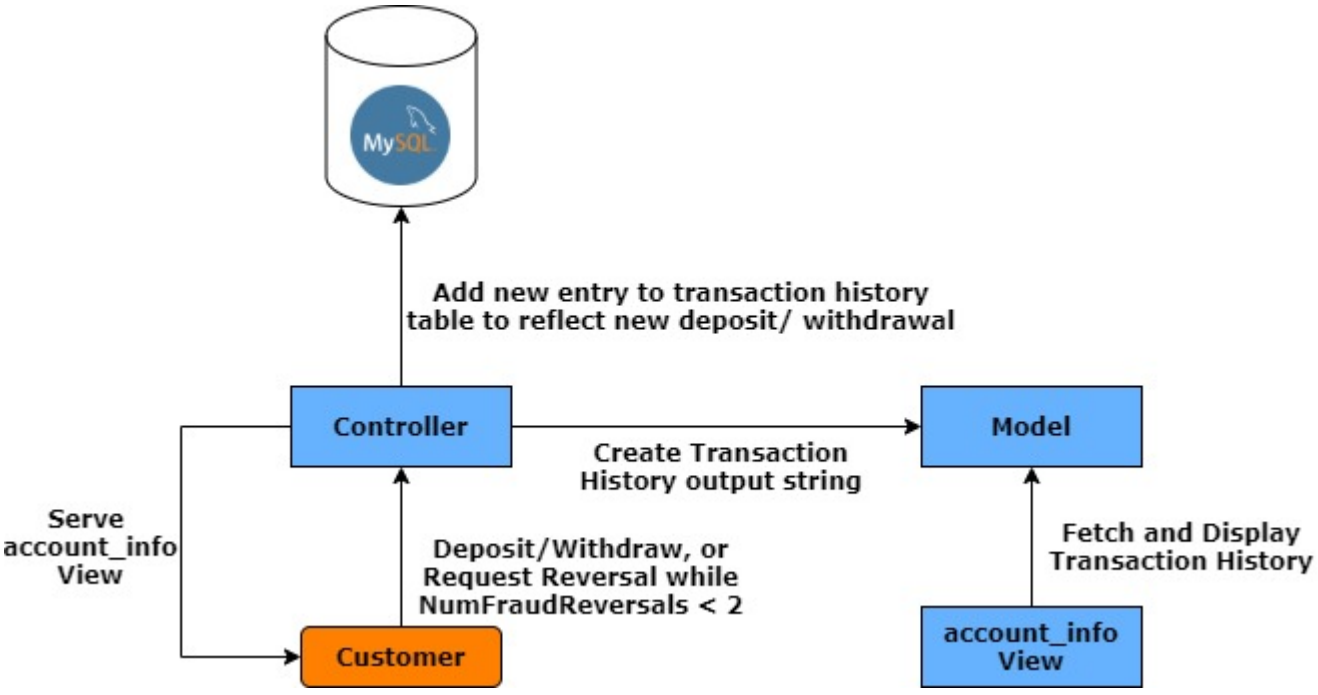
Cons:

- Doesn't give the customer power/freedom over their own money
  - The customer won't be able to access their money or account for a given period of time
- Depending on the implementation, would likely require another column for maintaining the date until the account is unfrozen.

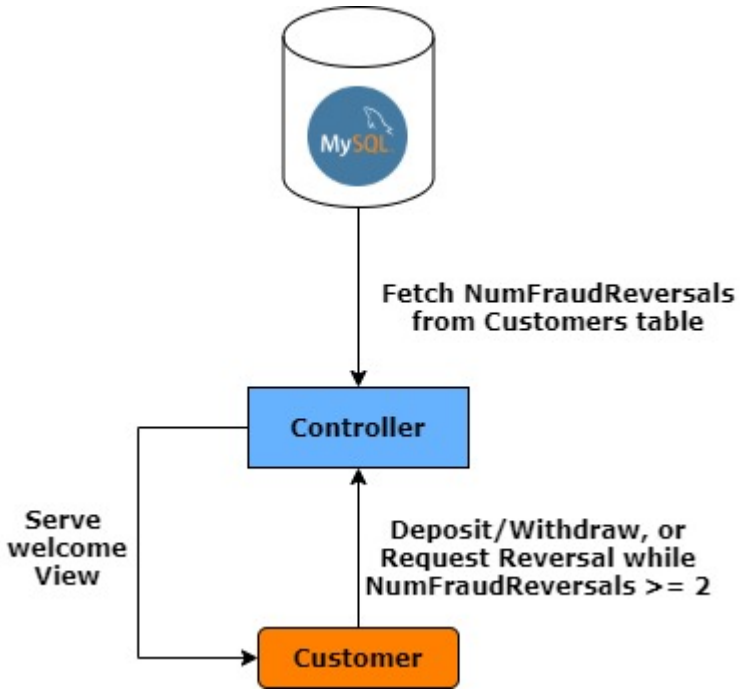
Technical Architecture (Reversal Approach)

MVC Logic

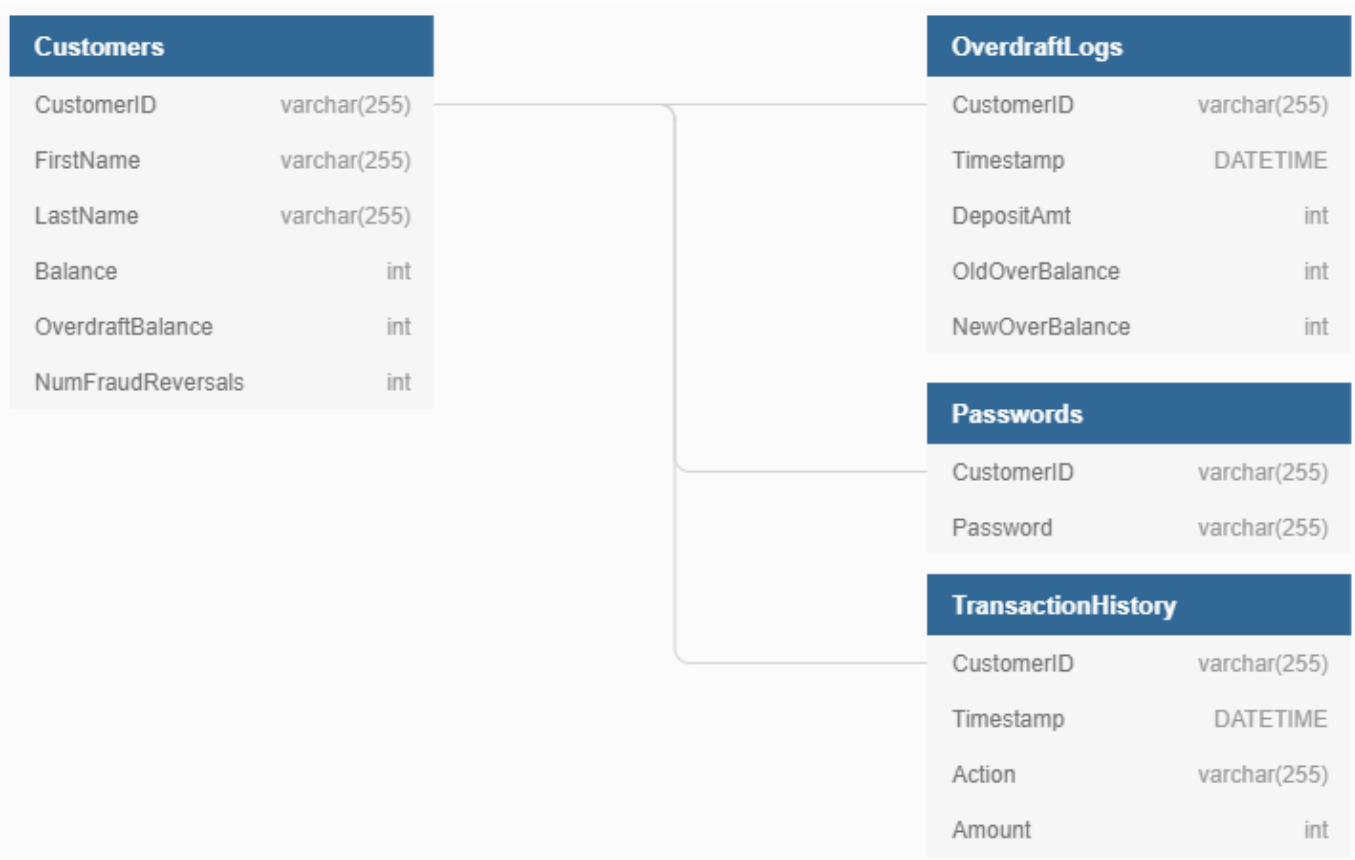
Deposit / Withdraw / Reverse when NumFraudReversals < 2 (and account is not yet frozen).



Attempting to Deposit / Withdraw / Reverse when NumFraudReversals >= 2 (and account is already frozen).



MySQL DB Schema



- `NumFraudReversals` column added to `Customers` table. Account should be frozen if `NumFraudReversals` >= 2.
- `TransactionHistory` table added.

- `CustomerID` from `Customers` table is a foreign key for `TransactionHistory`. This is a **one-to-many relationship**, as a single `CustomerID` from the `Customers` table can map to many `TransactionHistory` table rows since a single customer will do many deposits & withdrawals that get logged in the `TransactionHistory` table.