Testudo Bank Transfer Feature

Owners: Robert Schmidt (Robert301301@gmail.com), Adithya Solai (adithyasolai7@gmail.com), Sid Cherukupalli (cherukusid@gmail.com)

Problem Statement

Customers of Testudo Bank are currently limited to only withdrawing and depositing money. Customers would benefit from a service that enables them to transfer money from their account to another customer's account. This service helps customers by letting them move money to others in the same application they use to store their money (no third-party software needs to be used). This service will help Testudo Bank attract new customers.

Solution Requirements

- Customers should be able to view their past transfer history (both as a sender & recipient) in the front-end webpage.
- Customers should have a separate page in the front-end to submit a Transfer form.
- Customers that are in overdraft will automatically have all money received from others go towards paying off their overdraft balance.
- Customers should be able to transfer up to $1000 beyond their account balance to another customer of Testudo Bank. Any transfers above their account balance will follow the existing Overdraft logic.
- Disputes are not allowed on transfers as this can be taken advantage of: a customer sends money to another customer and disputes the transfer after the other customer has already withdrawn the money they received. If the person who received the money does not want it, they can just send it back.

Solutions Considered

Minimum features in all potential approaches: Customer must already know their recipient's customer ID to initiate a transfer. Customers will be provided a new page to send money to another customer that they can reach from the home page and `account_info` page. Customers will also see their transfer history on the `account_info` page.

Pro/Con of all approaches considered:

Transaction History Approach

In the **TransactionHistory** approach, we try to log every transfer using the existing `TransactionHistory` table.

Pros:

- No new MySQL DB tables, only a modification of an existing one. Less overall complexity of DB Schema.
- `TransactionHistory` table already has `Timestamp` & `Amount` columns for recording Transfers, and we can label the record as a Transfer by expanding the existing `Action` column to include `"Transfer"` as a valid Action in addition to the existing `"Deposit"` and `"Withdraw"` actions.

Cons:

- `TransactionHistory` table only has one `CustomerID` column, so it is not possible to store information about both the Sender and Recipient of a Transfer. It is also not feasible to add a new column for this purpose since Deposits & Withdraws are also stored in this table (which only involve one customer, unlike Transfers).

Transfer Approach

In the **TransferHistory** approach, a new `TransferHistory` table will be added to the DB Schema that stores the Sender, Recipient, Timestamp, and TransferAmount of each Transfer. This table will be updated every time a transfer occurs.

Pros:

- Able to store information about both the Sender and Recipient of each Transfer, unlike the `TransactionHistory` table, which only has one `CustomerID` column.

Cons:

- An extra table that needs to be maintained in our DB Schema, whereas the TransactionHistory approach just re-uses the existing `TransactionHistory` table.
- Logging Transfers in this table can create gaps in a customer's balance history in the `TransactionHistory` table; Transfers can increase/decrease a customer's balance but that information is only logged in the new `TransferHistory` table.

Proposed Solution

The proposed solution is a **hybrid** of the TransferHistory & TransactionHistory approaches described above. We will store information about each Transfer in both the existing `TransactionHistory` table and a new `TransferHistory` table.

The hybrid approach seeks to minimize the cons of the two different approaches, and keep the `TransactionHistory` table as a "single source of truth" for all changes in a customer's balance.

The `Amount` column in the `TransactionHistory` table will be ***positive*** for all transfers where the customer was a *recipient* (since their balance increases in this case), and ***negative*** for all transfers where the customer was a *sender* (since their balance decreases in this case). Two new `TransactionHistory` records will need to be populated for every transfer (one for the sender, one for the recipient).

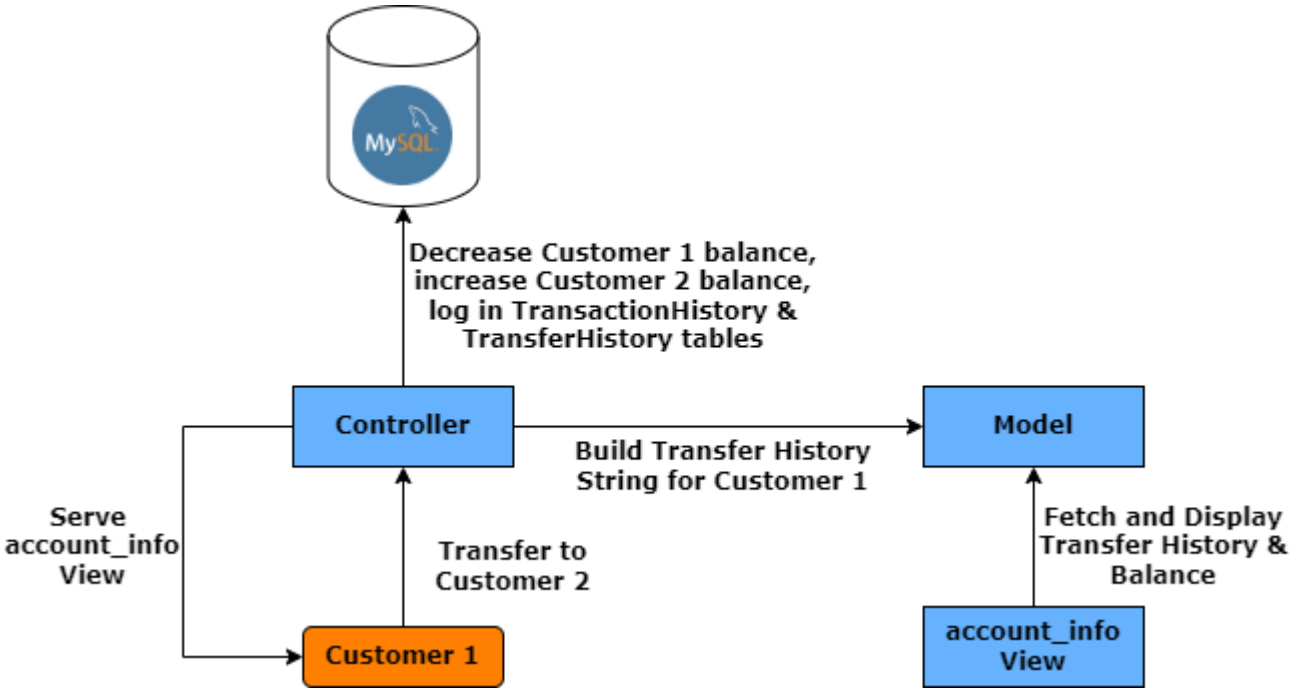
Only 1 new record will be added to the new `TransferHistory` table for every Transfer, since this table's schema can store information about both the sender and receiver in a single record.

The Timestamp used in both `TransactionHistory` and `TransferHistory` to record each Transfer will be the same.

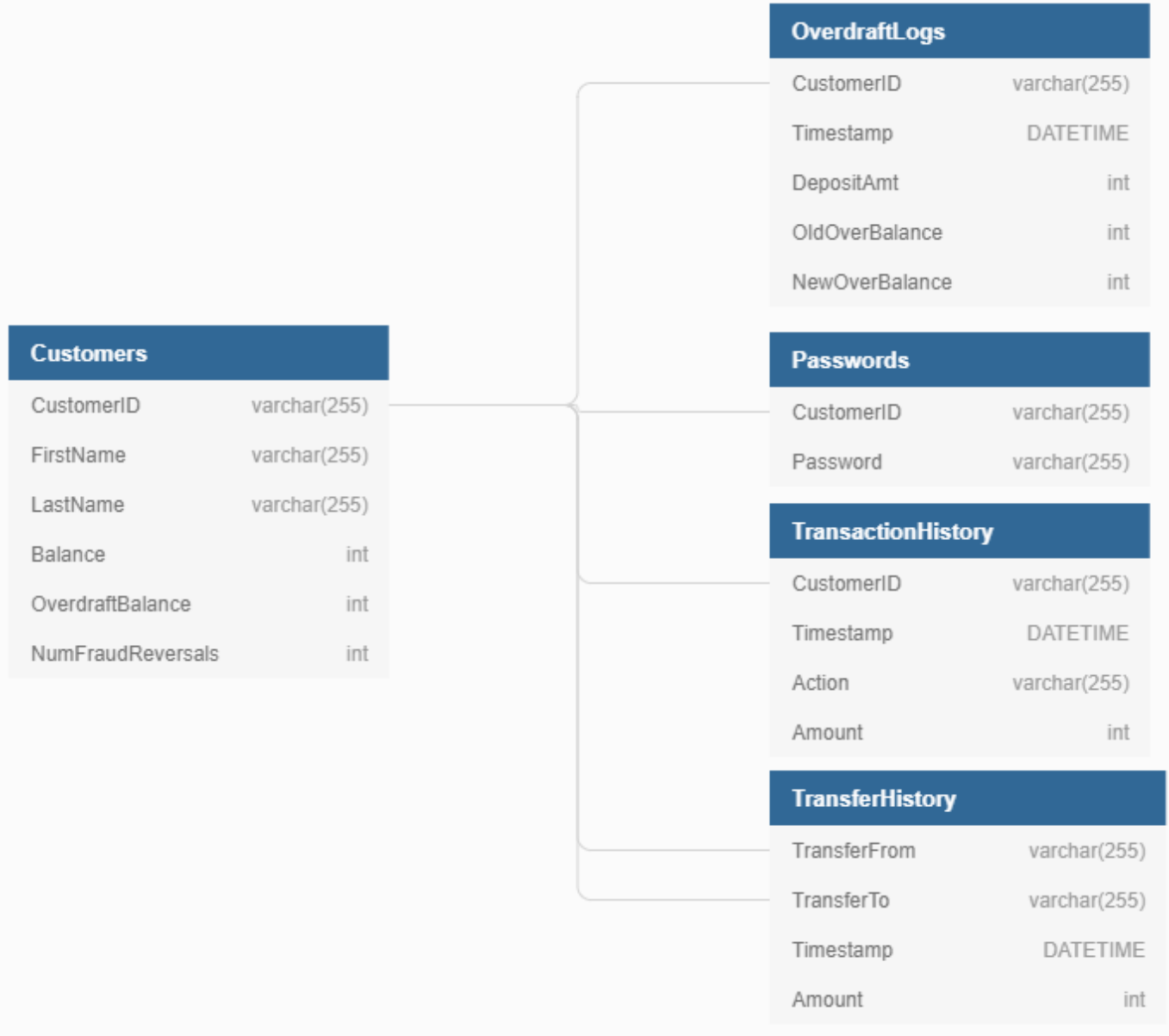
Technical Architecture

MVC Logic Diagrams

Simple Transfer Case (Recipient not in Overdraft, Sender has enough money in Main Balance)



MySQL DB Schema



DB Schema Notes

- Primary Key for new `TransferHistory` table is { `TransferFrom` , `TransferTo` , `Timestamp` }.
- Not shown in DB Schema Diagram, but `"Transfer"` is a new allowed value for `Action` in `TransactionHistory` table.
- Future developers can use the `CustomerID` and `Timestamp` from a Transfer record in `TransactionHistory` as a foreign key to fetch the corresponding Transfer record in the `TransferHistory` table. One inconvenience with this approach is that `CustomerID` from `TransactionHistory` would need to be compared with both `TransferFrom` and `TransferTo` to find a match. A better foreign key approach is to also use the ***sign*** (positive or negative) of the `Amount` column in a Transfer record in `TransactionHistory` table to search for the corresponding record in `TransferHistory` . A ***negative sign*** means the

customer was a Sender in this Transfer, so `CustomerID` from `TransactionHistory` table should be compared to `TransferFrom` in `TransferHistory` table. A ***positive sign*** means the customer was a recipient, so `CustomerID` should be compared to `TransferTo` .

Edge Cases

- If the recipient is in overdraft, then the transferred money should first go towards repaying the overdraft balance, and any leftover transfer money is added to the recipient's main balance.
    - The overdraft repayment should be logged in the `OverdraftLogs` table.
- If a sender initiates a transfer that is above their current main balance, then the amount exceeding their main balance is used to calculate an overdraft balance for the sender by applying the 2% interest rate.
    - Senders can not exceed the $1000 overdraft limit, so they can only transfer money up to $1000 more than their current balance.