Project Description

This project is a digital wallet system which allows users, once they have been registered into the system as a member with a login (username and password), to have their own digital bank account with some amount of money in it. This money can be added to their digital wallet from some payment method such as a credit card, debit card or bank account. They can also send and receive payments to other members of the system, request payments, cancel payments so long as they have not been accepted by the recipient at the other end yet (this means they are still pending), and even request refunds from someone they have made a payment to in the past so long as this payment is fully completed and not pending. This system should also be implemented in such a way that each member is assured that their money is kept safe under the supervision of a trusted third party and all transactions that a member can do under this system will go through this third party.

General Architecture

The digital wallet system works by having a centralized server which clients communicate with rather than each other directly. The server keeps a record of all members and their respective username and password for logging in to the system, as well as a ledger of all past payments for verifying refund requests (you cannot request a refund, for example, for an amount that you have not paid to someone in the past; this is simply a payment request and not a refund request). For user convenience, the server should also keep a record of payment information so that a member can quickly add funds to their digital wallet without having to type their entire credit card number or bank routing number in every time (though this information should be encrypted for safety). Every operation that a client can do goes through the server, such as with becoming a member; the server verifies that a username someone is trying to register under is not taken, and if successful the user and their login information is added to a file or database. Similarly, for something like making a payment, it is the server which subtracts funds from one user's wallet and adds them to another user's wallet. This is how we ensure that each member can trust the digital wallet system: so long as they trust the server to be a third party middleman for all transactions, they can trust the system as a whole.

Scenario 1: Join

The join scenario is only between one client and the server. When a user connects to the server, they will normally be prompted for a username and password, but ideally there should also be a "create an account" option. At this point, the server has no idea whether or not the client that has connected to them is a member. If they enter a username and password but they are not a member, the server will recognize this once it checks its database and notices that the username the client entered is not present. The server will inform the user of this, and ask them if they want to make an account or try again (if they simply made a type in their login information). Once the user decides they want to make an account and they tell the server this, they will be prompted for a username and password for their new account, and they may get an error back if the username they chose was already present in the database or their password was deemed to be not strong enough. Once they have entered valid login information that the server likes, though, they will be added to the database as a member and from now on they can simply login to the system with the information they chose.

Scenario 2: Pay

The pay scenario involves two clients and the server. Once successfully logged in to the system by entering their username and password, a client can tell the server that they want to make a payment. It is likely more efficient if the client also includes who they want to make the payment to, as well as the amount, that way it can all be done in one go rather than having the server prompt them for this information after it was informed of the client's intention to make a payment. When the server gets the information it needs, it will check to make sure that the amount of money in the sending client's digital wallet is greater than or equal to the amount they want to pay, and if it is not then it tell the user that they will have to add more funds to their account before making a payment of that amount. The server will also check if the user that the client is trying to make the payment to exists in the system, and inform the user if they do not. Once all this information that the client sent to the server is verified, the server then deducts the specified amount from the paying client's wallet and makes an entry in the ledger for this

transaction, marking it as “pending.” So long as it is still pending, this transaction is able to be canceled by the sending party. The next time the receiving party logs in to the system, they will be notified by the server that they have been sent money and, if they accept it, the specified amount will be added to their wallet and this transaction in the server’s ledger will be marked as “completed.”

Scenario 3: Refund

The refund scenario is also between two clients and the server. In order for this scenario to occur between two clients, there has to be a past transaction present in the ledger between these two clients for the correct amount that one of the clients wants to refund, and it has to be marked as “completed.” When a client sends a refund request to the server by specifying who they are requesting the refund from and the amount, they will get an error back if the server cannot find a completed transaction in the ledger which details that this same amount of money was paid from the client who is requesting a refund to the other client whom they are requesting a refund from. If this transaction does exist in the ledger, though, the server will then inform the client who needs to pay this refund, the next time they login to the system, that they have a refund request with regard to a past transaction. They can choose to either decline this refund request, in which case the requesting user will be informed that they denied it, or pay this refund request in which case a normal pay scenario of the requested refund amount and to the correct party will take place.

Scenario 4: Cancel

This final scenario is also between two clients and the server. In order for this scenario to occur, there has to be a pending transaction in the server’s ledger which can be canceled. A client will send a cancel request to the server indicating which transaction they would like to cancel (each transaction can be identified by a transaction ID, for example), and the server will check if it exists in the ledger. If this transaction ID is not in the ledger at all, the client will be informed of this mistake, but if it is in the ledger and is marked as “completed” the user will be informed that they cannot cancel this payment since the receiving user has logged in since they first made the payment and accepted the amount into their wallet. At

this point, the user will have to make a refund request instead of a cancellation request. If the transaction exists in the ledger and it is still marked as “pending,” the server will simply delete this transaction from the ledger and add the pending funds back into the paying client’s wallet.