Final Report

Megan Tran, Gabriel Goldstein, Yibo Li, and Jaeseong Lee

Design of ER diagram:

From the project description, the main task is to allow different users to make different transactions for bitcoin.

For the user, we have the entities “client”, “trader” and “manager” which belong to the entity “user”, we use ISA hierarchy to illustrate this relation, and different user types have different attributes, but they all have the unique userID and password. “Request” connects the trader and client, this relationship is used to assign clients with a trader to make transactions.

For the transaction, we have four types of transaction: sell, buy, transfer money and cancel. Both client and trader could buy or sell bitcoin, client need to transfer money to trader if she/he wants to buy/sell bitcoin through a trader, and trader could cancel any type of transactions. “Transfer” connects “client”, “trader” and the “USD transaction”. “Client_buysell” connects the “client” and “bitcoin transaction”, which indicates clients will buy/sell bitcoin by themselves. “Trader_buysell” connects the “trader” and “bitcoin transaction”, which indicates clients will ask the assigned trader to make a transaction for them. “Cancel” connects “trader”, “USD transaction” and “bitcoin transaction”, this relationship shows the trader could cancel any transactions and return the money to the client.

Finally, we have the entity “Log”, all information for any type of transactions (sell, buy, transfer money or cancel) will be stored in Log table.

Frontend:

For the frontend, we used React to create all of our pages. React is built on the infrastructure of javascript, CSS,, and HTML, so it behaves very similarly in nature. That is, similar to tags in HTML, React has what is called “components” that behave very similarly to tags. The only difference is that with React, one can create these components directly in the javascript code, making the integration of javascript, html, and css much easier for users.

In addition, we used React bootstrap to help style our components. Specifically, we used bootstrap “Card” components to give a consistent feel and style to each page. This allows user to learn how to interact with our frontend very quickly, since each page is so similar to the last. We

also used React bootstrap forms for each of our pages in order to very easily and efficiently submit data to the backend. We used a bootstrap form on our pages such as the login page, signup page, transaction page, etc. In conjunction with our forms, we used Axios to send requests/data to the backend. Axios is a very straightforward code package that allows coders to send HTTP requests.

We divided our front end into a series of pages, and each of those pages corresponds tpa javascript file in our source code. When the application first begins, it directs users to a login page (which corresponds to Login.js). This login page has two entry points (routes), either through "/", or "/login". A user is required to specify their username and password in order to continue. A third option available is to create a new account, which the user can select by pressing the "Create Account" button. If the user presses this button, it will direct users to a signin page (which corresponds to Signup.js). In this page, users first need to specify which type they are: namely, a client, trader, or manager. If they are a client, they must fill out all required information detailed in our ER diagram. If they are a trader or a manager, the only information required is a username, password, firstname, and lastname.

Once a user successfully logins or creates an account, the application then directs them to the appropriate screen for their needs. For a client, this means they are directed to the transactions page (corresponding to btcTransaction.js). The contents of the transactions page are dependent on the selection of the first field, the transaction type. There are three transaction types: buy, sell, or transfer. If the user selects buy or sell, they are then given two fields to fill out. The first field is the bitcoin amount they want to buy or sell. The second field asks the user to specify how they want to pay the commission: either in fiat or bitcoin. After that, the user presses submit, and their transaction is subsequently submitted to the backend for approval.

If a user logs in as a trader, the screen directs them to a page where they have the option to either search for a specific client based on client id, or firstname and lastname of a client.

If a user is a manager, they are directed to a screen that allows them to fetch transaction histories (corresponding to ManagerSearch.js). This requires a manager to first specify how they want the data broken down (either by days, weeks, or months). The manager must then give a start date and an end date. This information will be sent to the backend, which will then return a full list of transactions in the specified format and in the specified time range. Based on the information returned, it will be displayed in a table for the manager to examine.

Backend:

For the backend, we have used Flask to make the connection between MySQL database and python. There were several python files to support this task, and the majority functions were implemented in 2 python files, which are app.py and database.py.

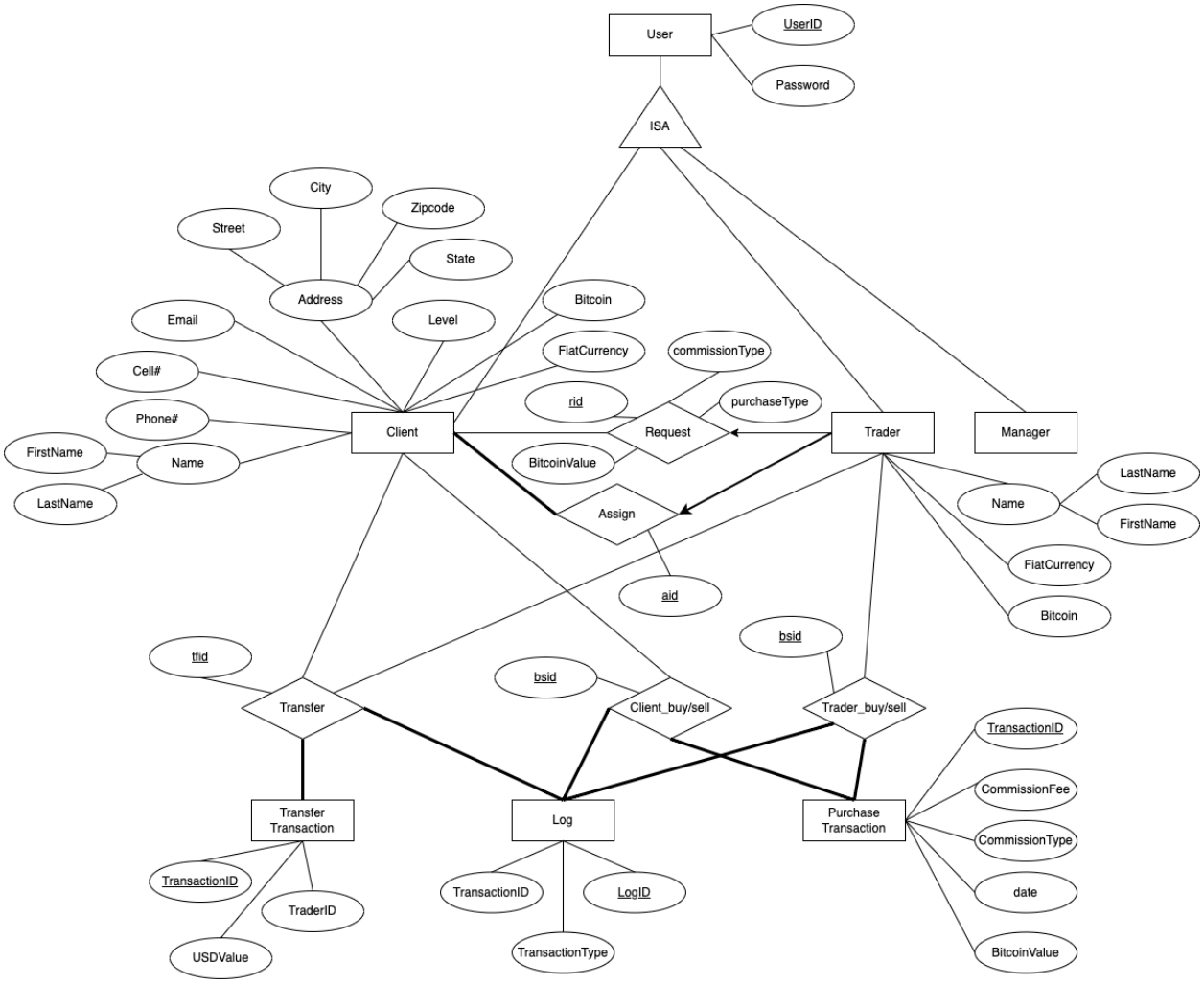
The app.py file is the root of the application, and this is where all Flask applications will go and create an environment variable that points to that file.

The database.py was used to implement the most SQL queries of this project. With the functions in app.py and corresponding SQL in database.py, we could realize all the actions that need to perform from the user, also the necessary modification and updates to the database, including and not limited to insert, cancel, update, display, search and transactions for bitcoin business.

Functions in app.py: for “login”, we took the input from the form of the frontend, which is the userid and the password, and then used the function of user_exist_in_db from database.py to check if the user already existed. If the username and password matched the record in the database, we have the variable session to determine the user login status. “update_level” is used to update levels, we have 2 levels gold and silver for this project. “Trader_assigned” and “client_assigned” are used to capture the trader and client. “Assign” is used to assign a trader for each client. “Transaction_history” is used to display all transaction history for a certain user. “Request_history” is used to find which trader was assigned to a client. “Manager_transction_history” is used to show clients and traders their transaction histories. “Request_bitcoin” is used for clients to request buy/sell bitcoin to a trader. “Buysell_bitcoin” is used for clients to buy/sell bitcoin themselves. “Transfer_money” is used for client to transfer money to the trader. “Cancel_transaction” is used for traders to cancel the transaction.

Important methods in database.py: “user_exists_in_db” is used to check if a user exists in database. “Insert_user_records” is used to insert the record of information for different user types(client, trader and manager”. When a trader is assigned to the client, “set_bitcoin_requests” is used to insert the information of the client and his/her assigned trader. “Get_bitcoin_requests” is used to display the information of a client and his/her assigned trader. “Buysell_bitcoin” is used to buy/sell bitcoin for client or trader, if the user is client, the commission _rate is based on the level, and we calculate the commission fee as bitcoin_value*commission_rate if the commission type is bitcoin, the commission fee as flat_value*commission_rate if the commission type is fiat. If the user is a trader, the commission_rate is based on the corresponding client’s level, and the calculation is the same. “Transfer_money” is used for clients to transfer money to the trader. “Cancel_transaction” is used to cancel transactions, such as sell, buy and transfer_money. For all transactions (sell, buy, transfer and cancel), the information is added to the log table.

ER diagram



Relational schema

1. user (<u>userID</u>, user_password)
2. Name(<u>firstname</u>, <u>lastname</u>)
3. Address(<u>address1</u>, <u>address2</u>, <u>city</u>, <u>zipcode</u>, <u>state</u>)
4. Client(<u>clientID</u>, password, reg_date, Name, PhoneNum, CellNum, email, address, level, bitcoin, flatcurrency)

5. Trader(traderID, password, reg_date, firstname, lastname, bitcoin, flatcurrency)
6. Manager(mangerID, password)
7. TransferTransaction(transferTr-ID, date, time, USD_value)
8. PurchaseTransaction(purchaseID, date, time, commissionType, commissionRate, bitcoinValue, flatValue, purchaseType)
9. Transaction(transactionID, tranferID, purchaseID)
10. Log(logID, logType)
11. Assign(assignID, clientID, traderID)
12. Request(requestID, clientID, traderID, bitcoinValue, commissionType, purchaseType)
13. Transfer(transferID,transferTr-ID, clientID, traderID)
14. Client_buysell(bsid, userID, purchaseID)
15. Trader_buysell(bsid, userID, purchaseID)