CMPT 368
Professor Abu Mallouh
Rachel Asante, Emelia Hajdarovic, Natalie McGovern
May 3, 2021

Final Project: Grocery Blockchain

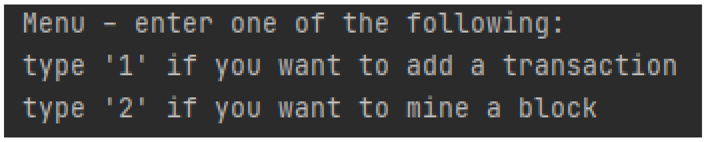
Project Idea

For our final project, our group decided to create a blockchain that can be accessed via a Google Cloud SQL database. In doing so, all of the nodes utilizing this system are able to gather information on their competing nodes in real time, provided they are all connected to the internet within this network. This blockchain is able to keep track of the basic information taken from clients purchasing various types of produce and other items from any provider of their choice. All the user needs to provide is the type of item they purchased, as well as where they purchased it from. This information is then stored in a grocery blockchain that also holds the timestamp of this transaction, as well as a randomly generated fee and transaction ID for this purchase. These transactions are then stored in a MEMPOOL kept on the Google Cloud database.

Through this randomly generated and unique transaction ID, miners can also utilize this software to sift through transactions in the MEMPOOL and automatically grab the transaction with the highest fee. As all nodes active in the network have access to the cloud database, the information they garner will always be up to date with one another to ensure that no miner encroaches on another to disrupt the grocery blockchain.

Implementation

MAIN MENU:

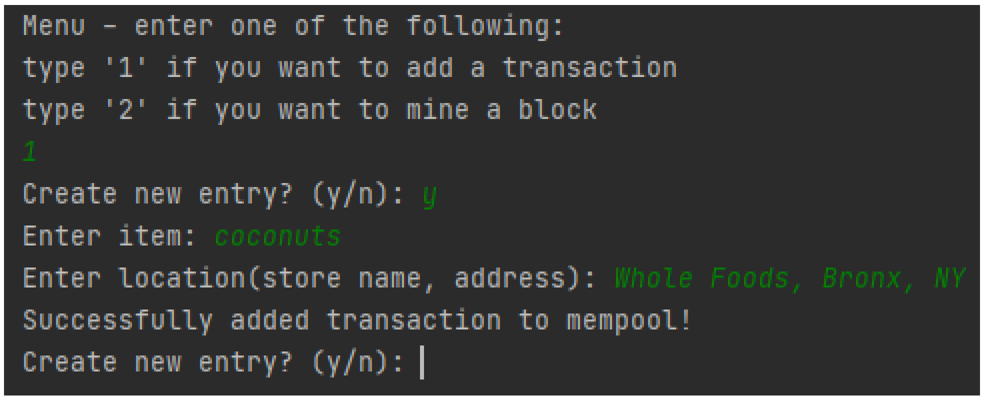


```
Menu - enter one of the following:
type '1' if you want to add a transaction
type '2' if you want to mine a block
```

Menu before the user makes any choices.

The program begins by displaying a simple menu in the console that gives the user two options: record a new transaction or mine a block. The system will then take the user's input of '1' or '2' and give them branching options from there.

OPTION 1 - TRANSACTIONS:



```
Menu - enter one of the following:
type '1' if you want to add a transaction
type '2' if you want to mine a block
1
Create new entry? (y/n): y
Enter item: coconuts
Enter location(store name, address): Whole Foods, Bronx, NY
Successfully added transaction to mempool!
Create new entry? (y/n): |
```

Menu when the user selects '1' to add a transaction.

If the user chooses to record a new transaction, the system will confirm the user's choice to create a new entry. Once confirmed, the system prompts the user to enter the name of the item and the name and location of the store where the item was purchased. To create the transaction, this system stores the information given by the user as well as system-generated information such as the current time, a randomly generated miner fee, and a randomly generated transaction ID. It is also important to note that the system will run a series of checks to ensure that the transaction ID does not already exist within the MEMPOOL. This is to ensure that every transaction is kept track of in a unique fashion within the Google Cloud SQL database. This also displays that with each new transaction created from a node, all nodes connected to the network via this database are able to communicate in real time. Once the transaction has been created, the system will add it to the MEMPOOL in the Google Cloud.

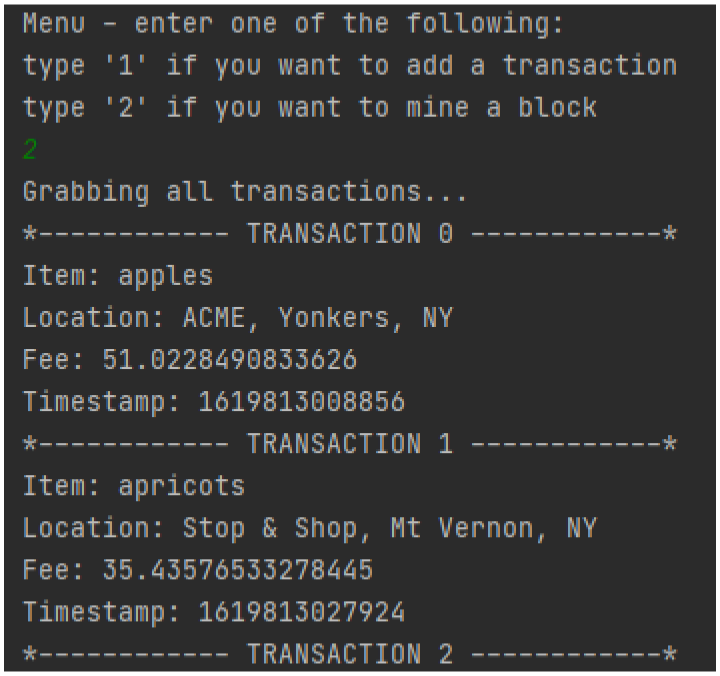
```
public Transactions(String item, String location) throws SQLException,
ClassNotFoundException {
        this.item = item;
        this.location = location;
        this.timeStamp = createTimeStamp();
        this.fee = createFee();
        this.tID = makeID();
    }
```

Constructor of Transactions class which holds information about: item purchased, location of purchase, time transaction was created by user, and a randomly generated fee and transaction ID.

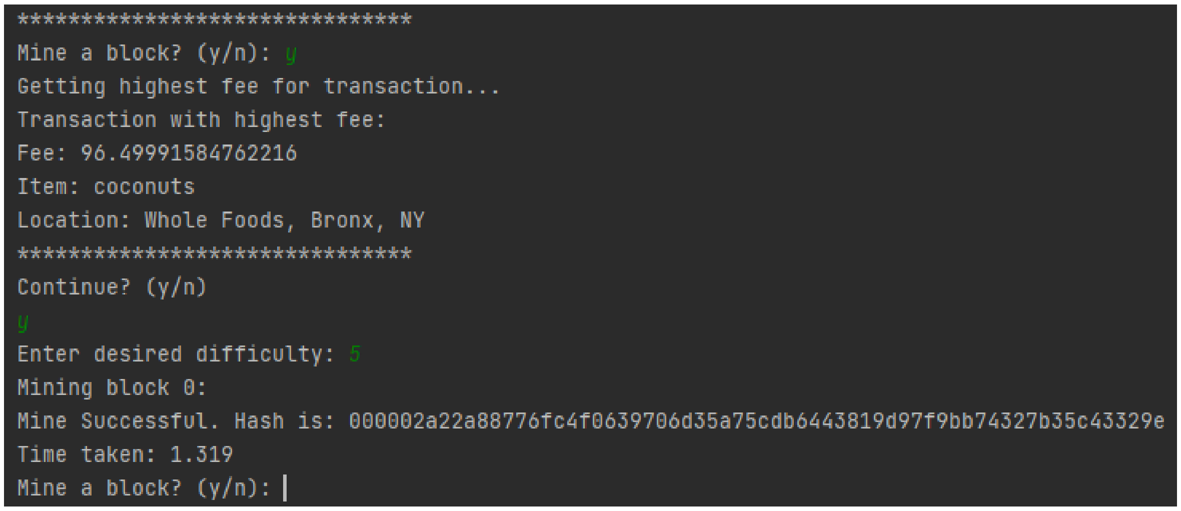
```
public String makeID() throws ClassNotFoundException, SQLException {
        String tID = UUID.randomUUID().toString();

        //check if it exists in the db
        Class.forName("org.postgresql.Driver");
        Connection con =
DriverManager.getConnection("jdbc:postgresql://34.67.145.192:5432/postgres"
, "postgres", "xCqm5JFEJH2mrwuj");
        Statement s = con.createStatement();

        String sql = "select trid, count(*) from mempool where trid = '" +
tID + "' group by trid;";
        ResultSet rs = s.executeQuery(sql);
        int check = 0;

        while(rs.next()){
            check = rs.getInt("count");
        }

        if(check > 0)
            tID = makeID();

        return tID;
    }
```

makeID function within Transactions that pulls data from the database and confirms whether the randomly generated ID exists on the MEMPOOL already or not.

OPTION 2 - MINER:


```
Menu - enter one of the following:
type '1' if you want to add a transaction
type '2' if you want to mine a block
2
Grabbing all transactions...
*------------ TRANSACTION 0 ------------*
Item: apples
Location: ACME, Yonkers, NY
Fee: 51.0228490833626
Timestamp: 1619813008856
*------------ TRANSACTION 1 ------------*
Item: apricots
Location: Stop & Shop, Mt Vernon, NY
Fee: 35.43576533278445
Timestamp: 1619813027924
*------------ TRANSACTION 2 ------------*
```
Menu when user selects '2' to mine a block.

If the user chooses to mine a block, the system will generate a unique ID for the miner behind the scenes, following the same checks as the transaction ID to confirm this node is unique. From there, the system will retrieve all the transactions currently stored in the MEMPOOL. This retrieval is then stored as a copy in a separate MEMPOOL that is unique to this node, that way any changes that the miner makes to this MEMPOOL will not permanently affect the general MEMPOOL shared between the other nodes until their consensus protocol is successful. After displaying the current transactions available, the miner is given the option to continue. Using this miner's unique MEMPOOL, the system will then locate the transaction with the highest transaction fee and ask the user once more if they wish to continue, and if yes, the transaction will be removed from this unique MEMPOOL. They will then be prompted to enter the desired difficulty. This is to provide each node with varying hash rates based on how intricate the miner wishes their block hash value to be. The system will then attempt to mine the block and, if successful, will display the time elapsed as well as their current hash value.



```
*******************************
Mine a block? (y/n): y
Getting highest fee for transaction...
Transaction with highest fee:
Fee: 96.49991584762216
Item: coconuts
Location: Whole Foods, Bronx, NY
*******************************
Continue? (y/n)
y
Enter desired difficulty: 5
Mining block 0:
Mine Successful. Hash is: 000002a22a88776fc4f0639706d35a75cdb6443819d97f9bb74327b35c43329e
Time taken: 1.319
Mine a block? (y/n): |
```

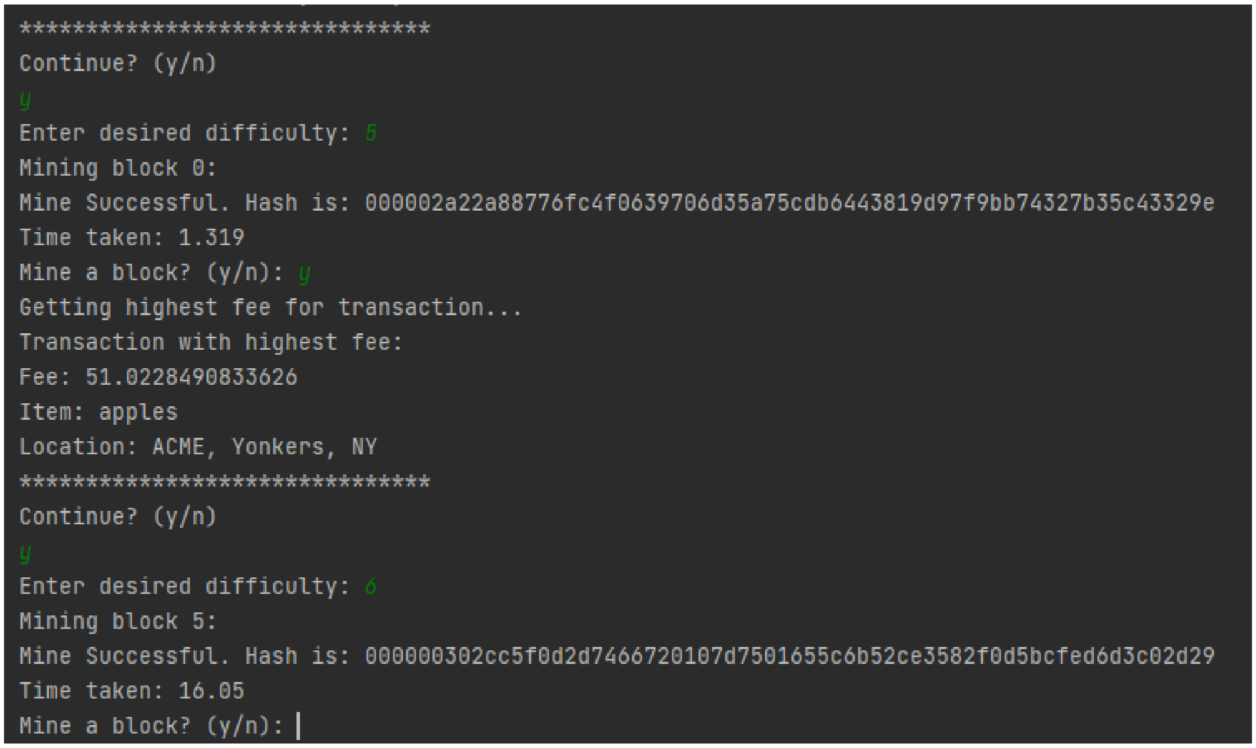
Menu when user selects ‘2’ to mine a block (cont’d)

From there, the system will handle the rest of the information required to create a full block for the grocery blockchain. Information will be pulled from the SQL database to get only the block number and hash value of the previous block in the chain, that way the new block can be appended without interrupting the flow of the grocery chain. From there, the transaction data selected for mining will be stored in the block’s data fields, as well as the nonce value needed to generate the hash value for this miner’s block. This information will not be made readily available to the miner in this instant, and instead they will only be able to see the time elapsed as well as the hash value of their new block. Once this information has been stored in a temporary chain unique to this miner, they will be given the option to continue mining for more blocks or to stop.

```
// update node's MEMPOOL to remove recently mined block
    public LinkedList<Transactions> updateTransactions(Transactions t,
LinkedList<Transactions> transactions){
        // search node's MEMPOOL for id
        for(int i = 0; i < transactions.size(); i++){
            if((transactions.get(i).tID).equals(t.tID)){
                transactions.remove(i);
            }
        }

        return transactions;
    }
```

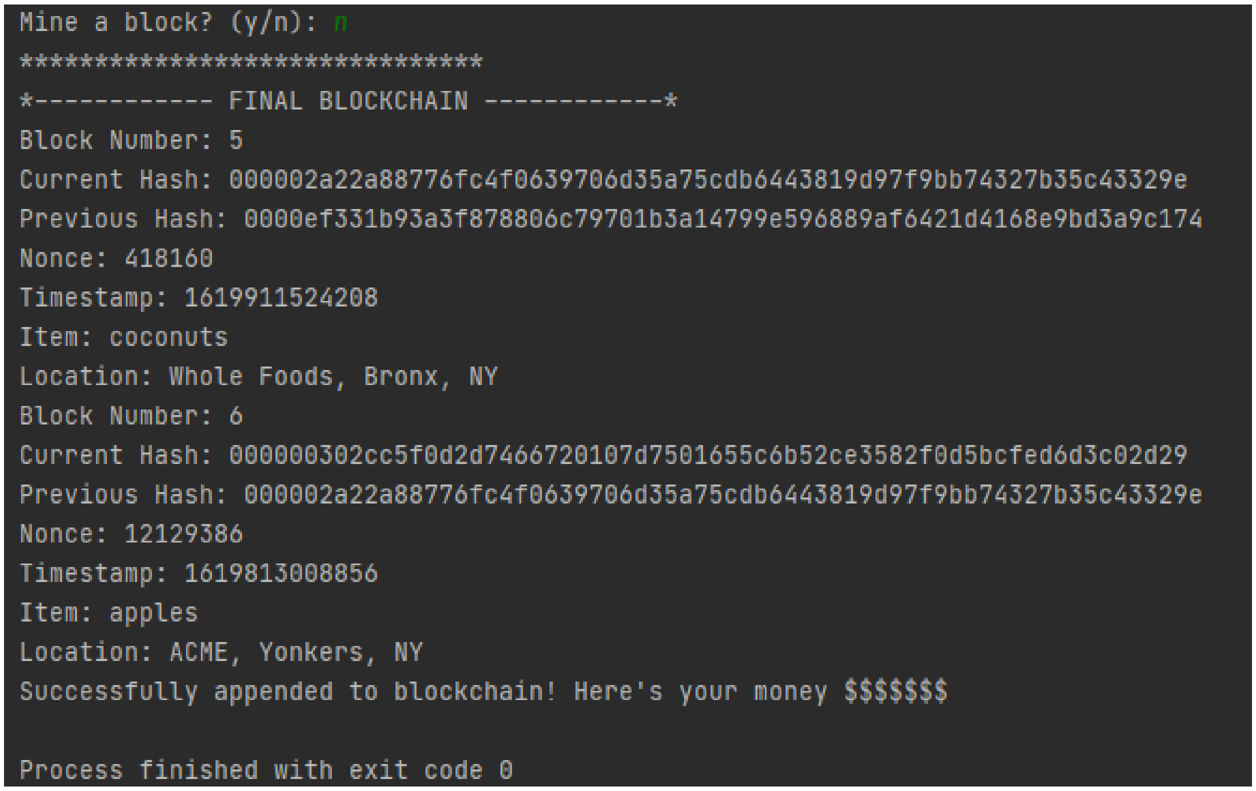
updateTranscations function that updates the node’s current MEMPOOL (in a LinkedList) by removing the block that was just recently mined using the previous highest transaction.

Should the user choose to keep mining, the system will run through the list of transactions available to them within this current node's MEMPOOL much like before. The loop that automatically shows the miner the next transaction with the greatest fee, as well as the question of what difficulty they would like, will continue to come up until they state that they are done mining. At the end of this loop, their final blockchain will be displayed. At the same time, the consensus protocol will be called to check through their final chain to ensure that: 1) No other miner has mined the same transaction as them and 2) If there are any competing chains, this miner has the longest of the competitors. If these two conditions are met, their chain can then be appended to the official grocery block chain, and the transactions they mined will be updated and deleted from the general MEMPOOL shared between the nodes.



```
*******************************
Continue? (y/n)
y
Enter desired difficulty: 5
Mining block 0:
Mine Successful. Hash is: 000002a22a88776fc4f0639706d35a75cdb6443819d97f9bb74327b35c43329e
Time taken: 1.319
Mine a block? (y/n): y
Getting highest fee for transaction...
Transaction with highest fee:
Fee: 51.0228490833626
Item: apples
Location: ACME, Yonkers, NY
*******************************
Continue? (y/n)
y
Enter desired difficulty: 6
Mining block 5:
Mine Successful. Hash is: 000000302cc5f0d2d7466720107d7501655c6b52ce3582f0d5bcfed6d3c02d29
Time taken: 16.05
Mine a block? (y/n):
```

Menu displaying the next highest transaction after a block was previously mined by the current node.



```
Mine a block? (y/n): n
*******************************
*------------ FINAL BLOCKCHAIN ------------*
Block Number: 5
Current Hash: 000002a22a88776fc4f0639706d35a75cdb6443819d97f9bb74327b35c43329e
Previous Hash: 0000ef331b93a3f878806c79701b3a14799e596889af6421d4168e9bd3a9c174
Nonce: 418160
Timestamp: 1619911524208
Item: coconuts
Location: Whole Foods, Bronx, NY
Block Number: 6
Current Hash: 000000302cc5f0d2d7466720107d7501655c6b52ce3582f0d5bcfed6d3c02d29
Previous Hash: 000002a22a88776fc4f0639706d35a75cdb6443819d97f9bb74327b35c43329e
Nonce: 12129386
Timestamp: 1619813008856
Item: apples
Location: ACME, Yonkers, NY
Successfully appended to blockchain! Here's your money $$$$$$$

Process finished with exit code 0
```

Menu displaying the completed blockchain of this miner, as well as their success in uploading this completed chain to the final grocery blockchain

In the off chance that the miner failed the consensus protocol, their chain will simply be lost and they will have to start fresh with a newly updated MEMPOOL, free of any overlap in mined blocks.

Through using a Google Cloud SQL Database, this provides a common platform for all the nodes to communicate with each other in real time- so long as they are connected both to the internet and to the database. They are able to pull data from the database in almost every step of the software, making it so all of the information gathered is consistent across all nodes. In order to prevent overlap and to enforce the consensus protocol, it is especially important for each node to update their competitors of the blocks they have mined. In doing so, when the miner is done creating their chain, both the unique miner IDs and transaction IDs can be compared across all nodes to determine if there was any clashing, and who won in this scenario.

```
// add miner's mined blocks to a temporary table
    public void addToMinedBlocks(Transactions t) throws
ClassNotFoundException, SQLException {
        Class.forName("org.postgresql.Driver");
        Connection con = 
DriverManager.getConnection("jdbc:postgresql://34.67.145.192:5432/postgres"
, "postgres", "xCqm5JFEJH2mrwuj");
        Statement s = con.createStatement();
        String sql = "insert into minedblocks values ('" + mID + "', '" + 
t.tID + "');";

        s.executeUpdate(sql);

        con.close();
    }
```

addToMinedBlocks function that appends the most recently mined transaction ID and the corresponding miner ID to a table that all nodes can access via the Google Cloud SQL database at the same time

Conclusion

After completing our software, it was interesting to note how our program could provide clients with a plethora of ways to improve their businesses. In particular, grocery stores would be able to utilize the information stored in this grocery blockchain to not only keep track of what products sell the most frequently in various locations, but also how well competing stores are doing in comparison to them. Our system has a very simple user interface via the console that is easy to work through, so clients of any technical ability would be able to make use of our grocery food block chain. By the same token, since this Google Cloud SQL Database is updated in real time, all of the information being provided is as up-to-date as possible. Our software is filled with several checks to ensure that every transaction is unique and anonymous, just as each node has a different miner ID each time the program is run. This creates a level of security between our software and the user so they are guaranteed their privacy while getting as much information from our database as possible.