CSL7490: Introduction to Blockchain
Assignment - 2

Description of the code and assumptions made:

General Transaction Life cycle:

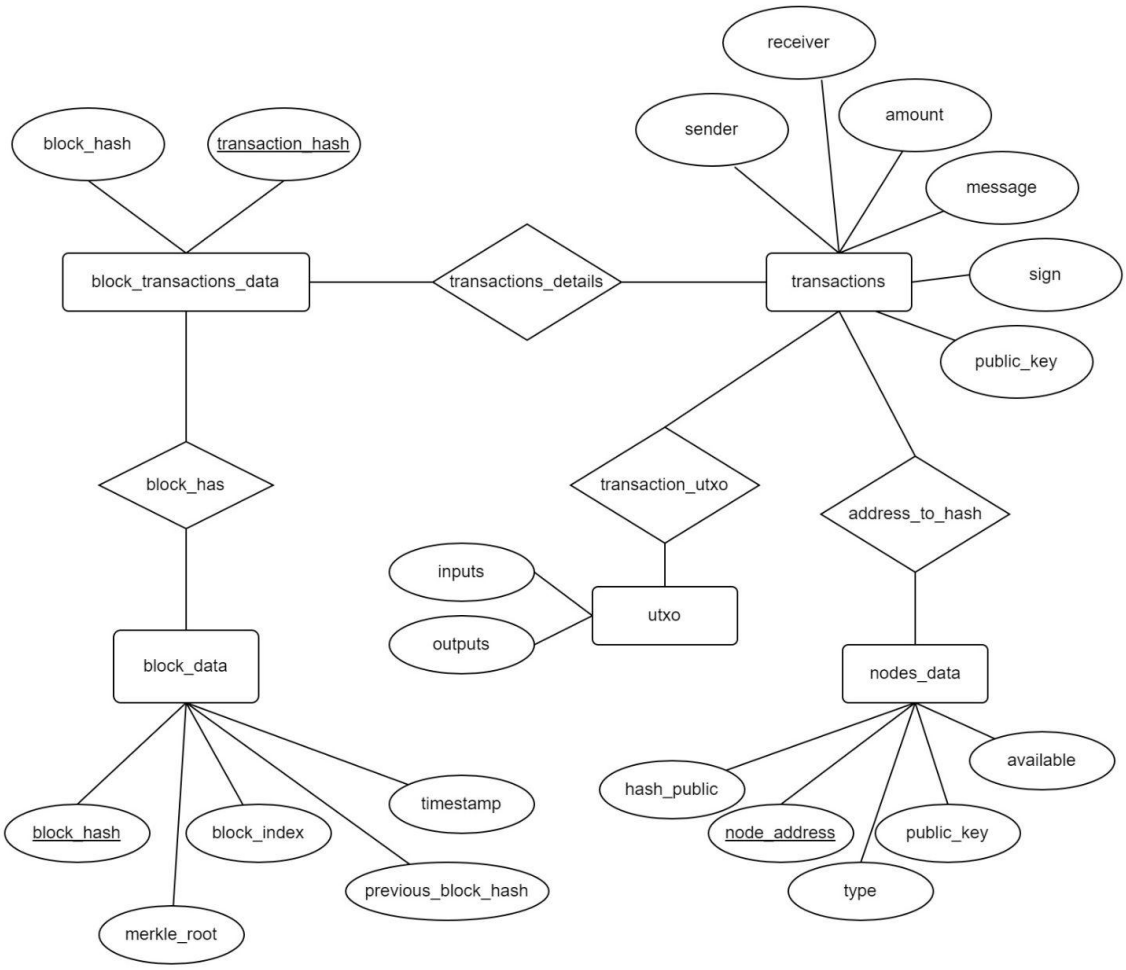
1. The user creates a transaction using a wallet and signs the transaction using a private key and then sends it to the memory pool.
2. The transaction is saved in the memory pool (transactions.csv).
3. The miner selected at random and connected to the user fetches the transaction (at max 3) and adds that to the block after digital signature verification.
4. Miner starts mining which is solving a proof of work and creates the block, with header and transaction hash in the body.
5. Miner sends the block to other miners for verification (both of block and transactions), if the block is verified a confirmation is generated and after 6 conformations (recommended confirmations to stop double spending attack) block is added to the blockchain.

The features implemented in the blockchain are the same as that in Assignment - 1.

The database schema is defined as:

1. The database Blockchain consists of 5 tables, which are created using CSV files.
2. The first table **nodes_data.csv** consists of the information about the data of the nodes that as their address, public hash, availability, type, etc.
3. The second table **transactions.csv** is used to store all the information required by the transactions like sender, receiver, amount, signature, and the public key to verify the signature.
4. The next table **block_data.csv** contain the data of blocks like block_hash, block_index, previous_block_hash, merkle_root, timestamp, etc.
5. The next table **block_transactions_data.csv** is used to store the transactions hashes of the transactions contained by a block.
6. The final table utxo.csv contains the denials about the input and output of the transactions.
7. The tables are mapped as shown in the ER Diagram. That is block_data.csv is connected to block_transactions_data.csv with block_hash as primary key - foreign key pair, block_transactions_data.csv is connected to transaction.csv table with transations_hash as primary key - foreign key pair, transactions are connected to utxo.csv and nodes_data.csv with one to one mapping.
8. All the tables are interconnected and thus fetching any data is completely user-friendly.

ER -Diagram and Schema defination



General Instructions for running the program.
There are three files **nodes_data.py**, **Blockchain.py**, **queries.py,** and **user.py.**

Steps:

1. Run all the files in the flask environment as discussed in the lecture by Lokendra Sir.
2. First, run **nodes_data.py** to create a table **nodes_data.csv** file with the data of nodes. Use the request **/create_nodes** with the **GET** method. **Remember to change the localhost variable in the nodes_data.py file as your localhost address if you are running without wifi it should be http://127.0.0.1 else it should be a wifi address like http://172.31.50.86.**

3. Then run user.py for each user create a new file and copy and paste the content of the file **remember to change the port number in each file.** Port numbers lie in the range of **5011 to 5025** for users.
4. After changing the port numbers for respective users use **/get_key_pair** request with the **GET** method to get the public and private keys.
5. Use **/add_transaction** request with the **POST** method to add a transaction from a user to the table **transactions.csv**. The format of the transaction is like this.

```
{
        'receiver': '',
        'amount': ''
}
```

   Where the receiver is the URL of the receiver like we need to send 10 BTC from URL **http://172.31.50.86/5011** to **http://172.31.50.86/5012** then use /add_transaction in URL **http://172.31.50.86/5011** with receiver **http://172.31.50.86/5012** as and amount as "10".
6. After creating the transaction the transaction is added to the **transactions.csv** file which acts as a memory pool.
7. Then use Blockchain.py and for each miner use a separate file with the different port numbers which lie in the range 5001 to 5010.
8. In Blockchain.py first use **/get_key_pair** request with the **GET** method to get the public and private keys.
9. Then use the **/connect_node** request with the **GET** method to get the connected nodes.
10. Then use **/add_transactions** request with the **GET** method to get the transactions done by connected users. This function adds transactions to a block after verifying the signature.
11. Then use **/print_transactions** request with the **GET** method to print the transactions that are in the current block.
12. Then use the **/mine_block** request with the **GET** method to create a block with the current transactions.
13. During mining two tables are used to store data which are **block_data.csv** and **block_transactions_data.csv**.
14. For the UTXO format of transactions using input and output, a table named utxo.csv is created.
15. Then use the **/get_chain** request with the **GET** method to get detailed information about the blockchain.
16. Use the **/replace_chain** request with the **GET** method to check the validity of the new block and add that to the existing chain.
17. If the chain is changed then the block added previously is deleted/ removed from the existing chain and transactions in that block are added again to the memory pool.
18. Finally, use the file queries.py files with requests like **/get_genesis_directly** (for directly getting genesis block information), **/get_genesis_transaction** (for getting genesis block

information using transaction hash), **/get_transaction_details** (for getting the details of the transactions for all blocks), **/get_block_details** (for getting the details of blocks based on block hash), **/get_block_recent** (for getting the recently mined block details), **/get_recent** (for the height of blockchain), **/get_average** (for getting the average number of transactions per block), **/get_summary** (for getting the summary as asked in the question).

19. Also, more queries can be added to the queries.py file.