

Introduction to Blockchain (CSL7490)

Assignment-1

With the implementation of a blockchain network, the following issue has been resolved which are usually faced in the centralized banking system.

- Intermediation: There is no third party system, the entire amount sent by the user is transferred to the user's wallet.
- Failure of centralized server: Since all the miners are connected through peer 2 peer network, failure of any one of the miner nodes won't affect the entire process as the same hyperledger is maintained by all the miners, and mining of block is done through a proper leader selection and consensus algorithm (Proof of work used by me)
- Transaction alteration: Since each of the transaction's hash is recorded in the body of the block and hash of previous block is also used to connect the next block, so there is no chance of alternation possible as changing even one slight entry will result in creating a different hash altogether and it will create flag to all the miners and users that some malpractice has been committed.
- Fake entry in the transactions: Since each block's transaction has been recorded in the UTXO format and also the hash of each transaction is stored with proper digital signing and verification, possibility of fake transactions are minimal.

Create at least 10 nodes as miners who are connected to each other. Each miner is connected to at least 2 users.

- Created 10 miners running at 10 different ports from 5001 and 5010.

```
In [1]: runfile('C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain/BC1.py', wdir='C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain')
* Serving Flask app "BC1" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
```

Miner node 1 at 5001

```
In [1]: runfile('C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain/BC10.py', wdir='C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain')
* Serving Flask app "BC10" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5010/ (Press CTRL+C to quit)
```

Miner node 10 at port 5010

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

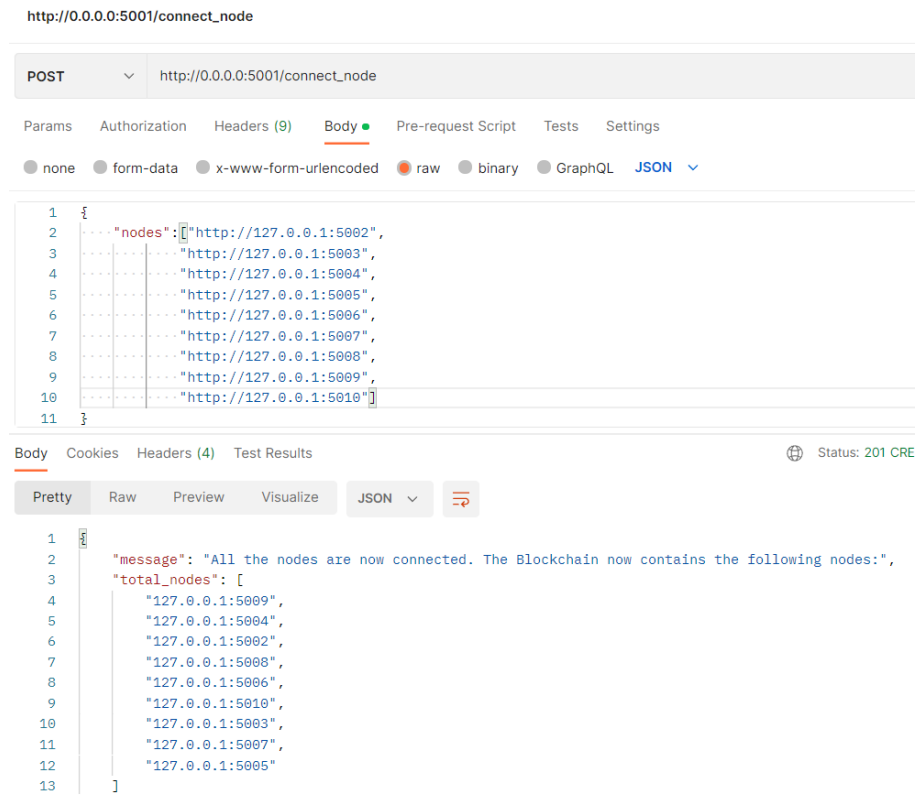
In [1]: runfile('C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain/BC9.py', wdir='C:/Users/pujag/OneDrive/Desktop/Semester 3/Blockchain')
* Serving Flask app "BC9" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5009/ (Press CTRL+C to quit)
```

10 miner nodes running at port 5001 to 5010

- Connected each of these miners through the “connect_node” function.
- API: [POST] method

http://0.0.0.0:5001/connect_node (For Miner at node 5001)

Connected each of these miner nodes with the rest of the 9 remaining miner nodes.



Miner at node 5001 port connected to all others 9 miner nodes

```
http://0.0.0.0:5010/connect_node

POST http://0.0.0.0:5010/connect_node

Body
[{"nodes": [{"http://127.0.0.1:5002",
"http://127.0.0.1:5003",
"http://127.0.0.1:5004",
"http://127.0.0.1:5005",
"http://127.0.0.1:5006",
"http://127.0.0.1:5007",
"http://127.0.0.1:5008",
"http://127.0.0.1:5009",
"http://127.0.0.1:5001"}]}]

Body
{"message": "All the nodes are now connected. The Blockchain now contains the following nodes:",
"total_nodes": [{"127.0.0.1:5007",
"127.0.0.1:5004",
"127.0.0.1:5001",
"127.0.0.1:5005",
"127.0.0.1:5002",
"127.0.0.1:5009",
"127.0.0.1:5006",
"127.0.0.1:5003",
"127.0.0.1:5008"}]}
```

Miner at node 5001 port connected to all others 9 miner nodes

Same way all other 8 miner nodes are connected to each other to form Peer 2 Peer Connection.

- Connected two users to each of the miner nodes using the “user_connect” function.
- API: [POST] method: http://0.0.0.0:5001/user_connect (For connecting user to miner 1)
Connected each of these miner nodes with the rest of the 9 remaining miner nodes.
- On connecting the user, a user’s wallet will be initialized.

```
POST http://0.0.0.0:5001/user_connect

Body
{"user_name": "user1"}

Body
{"message": "User Successfully connected to the blockchain network",
"user_name": "user1",
"user_public_key": "Public RSA key at 0x1B528AA7820"}
```

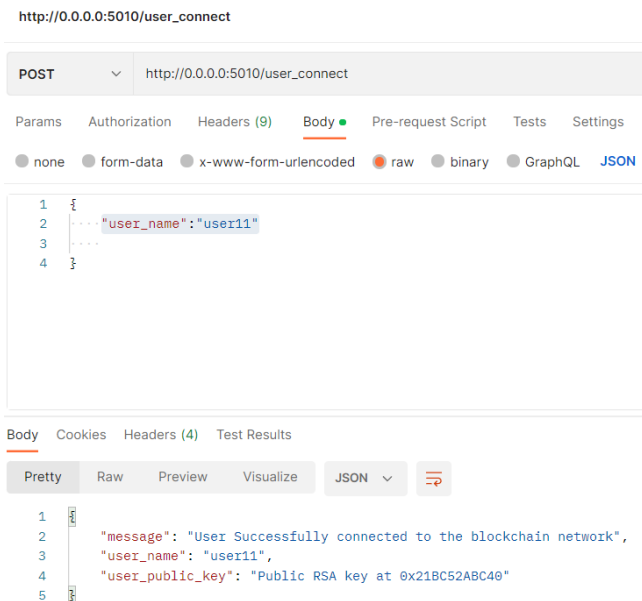
User1 connected to Miner 1

```
POST http://0.0.0.0:5001/user_connect

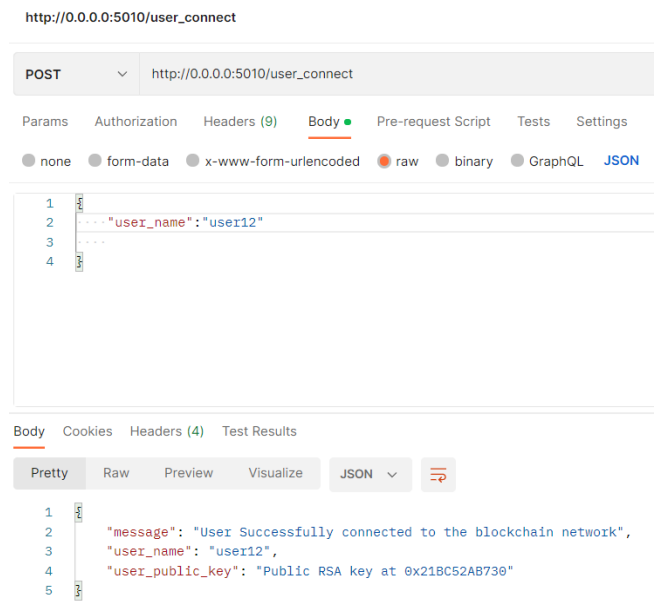
Body
{"user_name": "user2"}

Body
{"message": "User Successfully connected to the blockchain network",
"user_name": "user2",
"user_public_key": "Public RSA key at 0x1B528AABC10"}
```

User2 connected to Miner 1



User11 connected to Miner 10



User12 connected to Miner 10

Likewise we can connect 2 or more users to each of the different miner nodes.

Create a wallet which contains private and public keys. Use the hash of the public key as the address of each user's wallet.

Whenever a new user is connected to a miner node, created user's wallet having following fields by using function "initialize_Wallet_Credentials":

- Private_key : used for digitally sign the user transactions
- Public_key: used for verifying the transactions which were digitally signed by the private key of the sender.
- Address: created using the hash of the user's public key representing the address of each user's wallet.
- User_name
- Wallet_amount: While wallet initiation for new users, wallet is credited with 100 bitcoins for transactions.

```

#Initialize Mew private key, public key and address for new users
def initialize_Wallet_Credentials(user_name:str=""):
    random_gen = Crypto.Random.new().read
    private_key = RSA.generate(1024, random_gen)
    public_key = private_key.publickey()
    bitcoin_address = PKCS1_v1_5.new(private_key)
    wallet = User_Credentials(private_key, public_key, bitcoin_address, user_name)
    return wallet

```

Code for creating a user's wallet

Since I connected two users each to node1 and node10 (likewise we can connect users to other nodes as well).

```

* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Sep/2022 20:34:51] "POST /connect_node HTTP/1.1" 201 -
{'127.0.0.1:5009', '127.0.0.1:5004', '127.0.0.1:5002', '127.0.0.1:5008', '127.0.0.1:5006',
'127.0.0.1:5010', '127.0.0.1:5003', '127.0.0.1:5007', '127.0.0.1:5005'}
127.0.0.1 - - [21/Sep/2022 20:41:27] "POST /user_connect HTTP/1.1" 200 -
User's user name: user1
User Public Key: Public RSA key at 0x1B528AA7820
User Private Key: Private RSA key at 0x1B528AAB610
User's wallet address: <Crypto.Signature.pkcs1_15.PKCS115_SigScheme object at 0x000001B528AA7CD0>
User's Wallet amount: 100
127.0.0.1 - - [21/Sep/2022 20:43:11] "POST /user_connect HTTP/1.1" 200 -
User's user name: user2
User Public Key: Public RSA key at 0x1B528AABC10
User Private Key: Private RSA key at 0x1B528AAB9A0
User's wallet address: <Crypto.Signature.pkcs1_15.PKCS115_SigScheme object at 0x000001B528AABE50>
User's Wallet amount: 100

```

Wallet details for user1 and user2 connected to node 1

```

* Running on http://0.0.0.0:5010/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Sep/2022 20:35:49] "POST /connect_node HTTP/1.1" 201 -
{'127.0.0.1:5007', '127.0.0.1:5004', '127.0.0.1:5001', '127.0.0.1:5005', '127.0.0.1:5002',
'127.0.0.1:5009', '127.0.0.1:5006', '127.0.0.1:5003', '127.0.0.1:5008'}
127.0.0.1 - - [21/Sep/2022 20:45:29] "POST /user_connect HTTP/1.1" 200 -
User's user name: user11
User Public Key: Public RSA key at 0x21BC52ABC40
User Private Key: Private RSA key at 0x21BC52F9970
User's wallet address: <Crypto.Signature.pkcs1_15.PKCS115_SigScheme object at 0x0000021BC52ABBE0>
User's Wallet amount: 100
127.0.0.1 - - [21/Sep/2022 20:46:22] "POST /user_connect HTTP/1.1" 200 -
User's user name: user12
User Public Key: Public RSA key at 0x21BC52AB730
User Private Key: Private RSA key at 0x21BC5300A60
User's wallet address: <Crypto.Signature.pkcs1_15.PKCS115_SigScheme object at 0x0000021BC52ABB50>
User's Wallet amount: 100

```

Wallet details for user11 and user12 connected to node 10

The header of a block in the blockchain should contain: block index, timestamp, previous block hash, and merkle root.

- If there is no existing blocks in the blockchain i.e. the blockchain is empty, Created a “Genesis Block” with index 1 and previous_hash as 0

Here in the below attached screenshot, we can see that since all the nodes are connected through the P2P network, the genesis block will be the same for the entire blockchain. Also in the header section of the block all the specified fields, i.e. index, merkle_root, previous_hash, timestamp as specified in the question is maintained.

http://0.0.0.0:5001/get_chain

GET http://0.0.0.0:5001/get_chain

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

| KEY | VALUE |
|-----|-------|
|-----|-------|

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2    "chain": [
3      {
4        "body": {
5          "transactions": []
6        },
7        "header": {
8          "index": 1,
9          "merkle_root": 0,
10         "previous_hash": "0",
11         "proof": 1,
12         "timestamp": "2022-09-21 21:56:12.917086"
13       }
14     },
15   ],
16   "length": 1
17 }
```

get_chain() at node1

http://0.0.0.0:5010/get_chain

GET http://0.0.0.0:5010/get_chain

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

| KEY | VALUE |
|-----|-------|
|-----|-------|

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2    "chain": [
3      {
4        "body": {
5          "transactions": []
6        },
7        "header": {
8          "index": 1,
9          "merkle_root": 0,
10         "previous_hash": "0",
11         "proof": 1,
12         "timestamp": "2022-09-21 21:56:12.917086"
13       }
14     },
15   ],
16   "length": 1
17 }
```

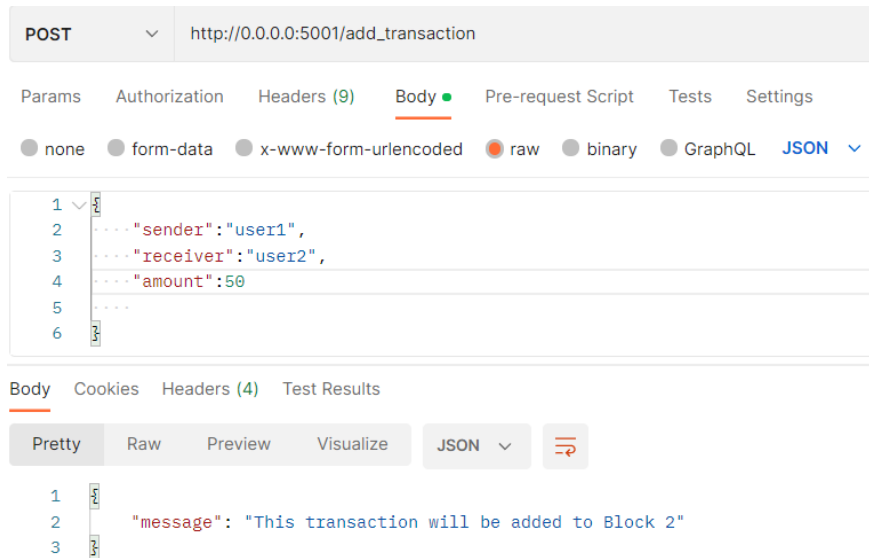
get_chain() at node10

The body of the block should contain the hash of each transaction. Using digital signatures verify the sender and receiver.

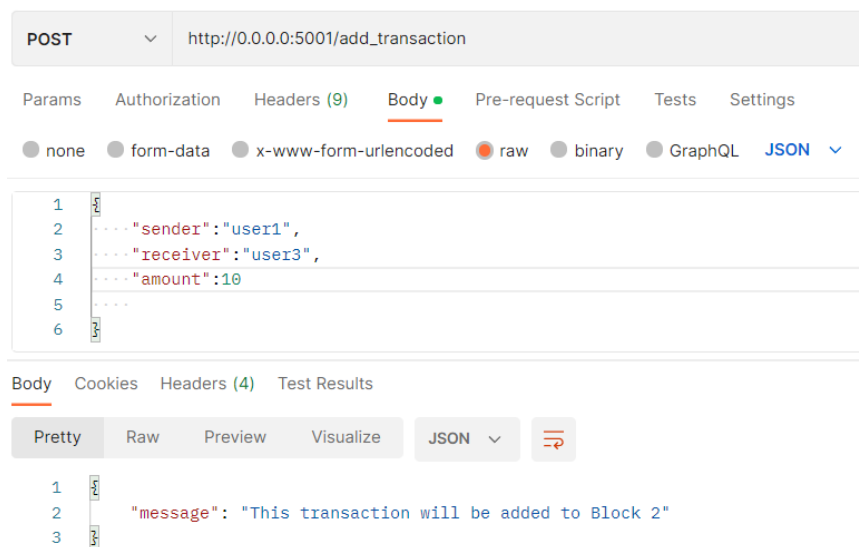
We have restricted that the transactions are possible only between the connected users to the miner nodes as we need user's wallet credentials for digitally signing and verifying the transactions.

- Steps for transaction creation:

1. Whenever a transaction is initiated between two users i.e. sender and receiver, firstly that transaction is digitally signed by the sender's private key and that digital signature is a part of the transaction.



Initiating a transaction between user1 and user2



Initiating a transaction between user1 and user3

Likewise initiated another two transactions between user2 and user1 and user2 and user3.

2. Each transaction contains a hash of each transaction of the form.

```
{ "sender": sender_username,  
  "receiver": receiver_username,  
  "amount": amount_sent }
```

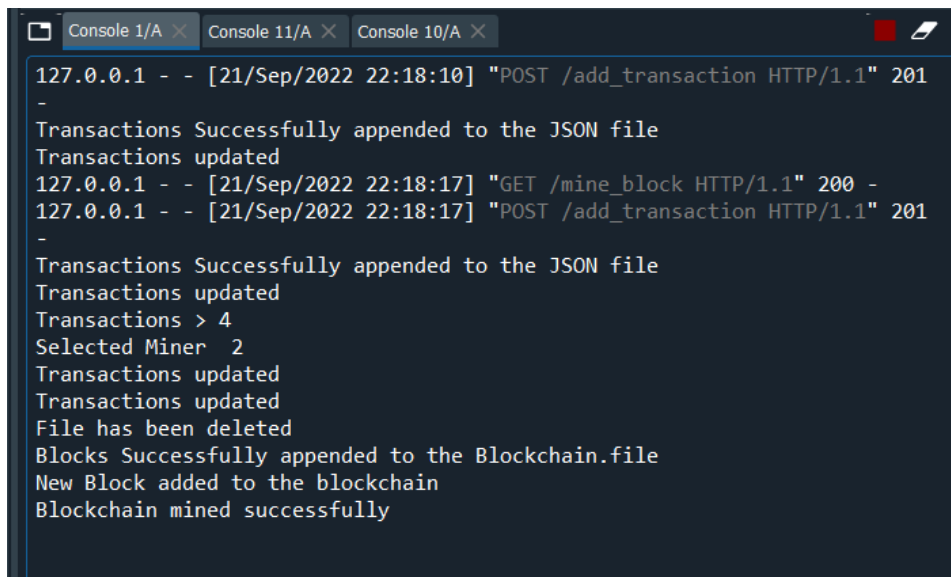
3. Transactions are stored in the UTXO format as specified in the assignment.

4. Once the transaction is created, it is stored in the transaction memory pool, which is broadcasted to all the connected miner nodes (in our case miner nodes at node1 (port: 5001) to node10 (port: 5010) for verification using sender's public key, while mining the block.

As per my logic whenever the number of transactions exceed or become equal to 4 in the transaction memory pool:

- 1.1. Miner node is randomly selected from node 1 to 10 through the leader selection mechanism (random selection).
- 1.2. The selected miner node mines the new block and the consensus algorithm used is Proof of Work.
- 1.3. While mining the digital signature of each transaction is verified before adding the new block by the selected miner node.
- 1.4. Once the block is added to the blockchain (it is same for all the nodes as all are connected through P2P network)

In my example the users connected to node1 created the transaction, which was broadcasted to all the miner nodes and through leader selection node2 (port: 5002) was selected for mining the block. While mining a new block I cleared the transactions from the memory pool.



```
Console 1/A X Console 11/A X Console 10/A X
127.0.0.1 - - [21/Sep/2022 22:18:10] "POST /add_transaction HTTP/1.1" 201
-
Transactions Successfully appended to the JSON file
Transactions updated
127.0.0.1 - - [21/Sep/2022 22:18:17] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [21/Sep/2022 22:18:17] "POST /add_transaction HTTP/1.1" 201
-
Transactions Successfully appended to the JSON file
Transactions updated
Transactions > 4
Selected Miner 2
Transactions updated
Transactions updated
File has been deleted
Blocks Successfully appended to the Blockchain.file
New Block added to the blockchain
Blockchain mined successfully
```


http://0.0.0.0:5010/get_chain

Save



GET

http://0.0.0.0:5010/get_chain

Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Cookies

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 21 ms Size: 2.92 KB Save Response

Pretty

Raw

Preview

Visualize

JSON



```
15  {
16    "body": {
17      "transactions": [
18        {
19          "input": {
20            "sender": "user1",
21            "sender_public_key": "67257af8c20817456bbfd402f331dbff34ca3ea10e20d754324ef41fb1cd98af",
22            "signature":
23              "2389f4249b0ff540f1c5c659ac5278bc91ac19ee08627fc31fbb04e51c6e29a096f437b8031c25ace2319a0f60cef536b24fbf5e5ad:
24              ca9d0aa7c7a5e973d2fdab3dee05a1a9f3a8a83b7cdd4f692cb37fa4d2518c1757701abef5b1688069cea42b88a8661ac755e19f5e56:
25              4d086334188e93d718641360ebb8d18fde9b9bf",
26            "transaction_hash": "1d41fb7c7e0baa98861fdb2f6c3591138c4ef978f94276b3a222ccf890108375"
27          },
28          "output": {
29            "amount": 50,
30            "receiver": "user2",
31            "receiver_public_key": "71d550608e0c91c324ce2a56b7cd2de02e49d9543c752e7f11584e9809cb508e"
```

```
31  {
32    "input": {
33      "sender": "user1",
34      "sender_public_key": "67257af8c20817456bbfd402f331dbff34ca3ea10e20d754324ef41fb1cd98af",
35      "signature":
36        "970fdc34b80aeb6a47aa63be08b28347f3b3ba48f7ac6b53ab5afcc33ed011b6d1a54b32509aea320b8f28838998ca8f7ed32492f:
37        bb8c5cbe22eba0cd56b14024559e0f05bb62d471a31ae5115ceb9d84e6d84d556dbc29e09ef874d6456e53e3dd91493ef45b6c9b2f:
38        7bed9f5eecd7e8670b224c4a44dc65aa707827",
39      "transaction_hash": "dcf157aee7c545da9e366ad13ca54055c9e01df1f2be1fea27c08aaeb14d5839"
40    },
41    "output": {
42      "amount": 10,
43      "receiver": "user3",
44      "receiver_public_key": "1102b247bbc908acc983f56223ab81159a6f1399c5b716cefb5dcb1ecf4eb12a"
```

```
{
  "input": {
    "sender": "user2",
    "sender_public_key": "71d550608e0c91c324ce2a56b7cd2de02e49d9543c752e7f11584e9809cb508e",
    "signature":
      "5a8a539e5489b5258e3de3515c5c9f2b2f64f89a52a1eacd5c0af682fb37194451fade6421142ae51c7515f3e723e429dfaa5d7f:
      1c0fb67d83ac661c93469ee5efb9ff424fc3d1b5d5874b997f2e619ed9dc3d46968b4c444e6962a954011b6fe82e5e164569fb4af:
      b50307d37adf67c8534eb314d3ee0cf992776a8",
    "transaction_hash": "c11c6522571b0356d241651042c81c4411243ce7e8fb4fad18267f1debd50146"
  },
  "output": {
    "amount": 20,
    "receiver": "user1",
    "receiver_public_key": "67257af8c20817456bbfd402f331dbff34ca3ea10e20d754324ef41fb1cd98af"
  }
}
```

```
Raw Preview Visualize JSON ↗
{
  "input": {
    "sender": "user2",
    "sender_public_key": "71d550608e0c91c324ce2a56b7cd2de02e49d9543c752e7f11584e9809cb508e",
    "signature": "2ae51b25d224b2c2eaea365dcd68d2cc59d64d703ef990aedc2e1b14290313f253e000ee80d4a11d34fa94791d8390226916bd9172302cf9a09b26c7a5551c30c9126862353097b603dc71b0e612760656c8d554aa3f1041c5aff09044c3ef09fd4c1efabcac4b9:2b14a62369f0756fcf26fff14c8ccb5ae89b3f1",
    "transaction_hash": "9bdd80f48a1c39ea6937dd3c7892c8e31edd8e9cf4ccafbf7d076153af920a66"
  },
  "output": {
    "amount": 10,
    "receiver": "user3",
    "receiver_public_key": "1102b247bbc908acc983f56223ab81159a6f1399c5b716cefb5dcb1ecf4eb12a"
  }
},
{
  "header": {
    "index": 2,
    "merkle_root": "003cc9e6966209f3384ee9bbce7cb97c816b4ec76c9ca757f6cc2adcf941848a",
    "previous_hash": "33466db01f7e9c9fef9f8b98edfc3e5c6ae6674cc75967eeff4affbc3dbbd8d2",
    "proof": 533,
    "timestamp": "2022-09-21 22:18:17.144860"
  }
}
```

Blockchain accessed at node10: It contains 4 transactions in the body part and the header part contains index, merkle_root, previous_hash, proof and timestamp.

Store the transactions in UTXO format (w.r.t. Bitcoin Wallet)

As shown in the above screenshot, each transaction has been stored in the UTXO format. Each transaction has been stored in the following format:

- Transaction['input'] contains transaction hash, sender's public key and digital signature of the transaction.
- Transaction['output'] contains receiver's amount, username and wallet's address as shown in the below screenshot.

```
Raw Preview Visualize JSON ↗
{
  "input": {
    "sender": "user2",
    "sender_public_key": "71d550608e0c91c324ce2a56b7cd2de02e49d9543c752e7f11584e9809cb508e",
    "signature": "2ae51b25d224b2c2eaea365dcd68d2cc59d64d703ef990aedc2e1b14290313f253e000ee80d4a11d34fa94791d8390226916bd9172302cf9a09b26c7a5551c30c9126862353097b603dc71b0e612760656c8d554aa3f1041c5aff09044c3ef09fd4c1efabcac4b9:2b14a62369f0756fcf26fff14c8ccb5ae89b3f1",
    "transaction_hash": "9bdd80f48a1c39ea6937dd3c7892c8e31edd8e9cf4ccafbf7d076153af920a66"
  },
  "output": {
    "amount": 10,
    "receiver": "user3",
    "receiver_public_key": "1102b247bbc908acc983f56223ab81159a6f1399c5b716cefb5dcb1ecf4eb12a"
  }
},
{
  "header": {
    "index": 2,
    "merkle_root": "003cc9e6966209f3384ee9bbce7cb97c816b4ec76c9ca757f6cc2adcf941848a",
    "previous_hash": "33466db01f7e9c9fef9f8b98edfc3e5c6ae6674cc75967eeff4affbc3dbbd8d2",
    "proof": 533,
    "timestamp": "2022-09-21 22:18:17.144860"
  }
}
```