EMU-Computer Department

CMSE353

Term project

Crypto currency using Block chain

Course supervisor:

Prof. Dr. Alexander Chefranov

Assistant:

Felix Babalola

Group Number:

1

Team Leader:

Abdalla Sherif 19700465

Other Team Members:

Jihad Mawlawi 19700475

Mohamedalkamel Omar 19700976

Parsa Fouladi 19700039

Date:

9/1/22

Outline

Table of Contents

Problem definition

We have developed a cryptocurrency system, using a block chain database system. In the normal databases in which we use daily in our lives, there's only 1 server computer who is responsible for storing the database contents, checking integrity of the columns and values. A cryptocurrency is a digital or vital currency that is secured by cryptography, which makes it nearly impossible to counterfeit or double-spend. Many cryptocurrencies are decentralized networks based on blockchain technology. Blockchain technology is the heart and soul of cryptocurrencies, blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. A blockchain is known for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party. In order to implement this, we have used C# language to create a decentralized server that processes the transaction conducted using our cryptocurrency. Moreover, we have used socket programming in order to connect our server to our users processing their transactions.

Description of our work

For this project, every one of us did their research to get all the information about block chain and crypto currency and how to implement it. After visiting several sites and gathering different information about the topic, we broke down our options and found the right algorithm which is the system we have that allows us to send and receive currency through the block chain. Then we explained it to each other in the team and we divided the report between all team members. We used MS Teams platform for the meetings we conducted and WhatsApp to communicate with each other via chat.

Description of algorithm for solving your problem:

In order to meeting the requirements in this project, we have created several classes. Transaction class which is responsible for creating transactions. Block class which is for saving the information of a block and creating hash of the block (by SHA256) by combining all the information of the block (such as list of

transactions (it will convert it to Json), Nonce (Number of tries for achieving the desired hash in mining), timestamp and hash of the previous block). Then we have BlockChain class which is responsible for the chain, rewarding, validity and getting balance of a user (detailed information will be given below). In addition, we have two more classes which are designed (by socket programming methods) to run a server and a client. By using these classes, we have achieved peer to peer network in the blockchain. The first class which is for the client side is called P2PClient, which is for connecting and sending information to sever (in Json format) as well as being able to receive from it. The other class which is for the server side, is called P2PServer. This one is responsible for creating a server and a behavior and also being able to receive and send data (in Json format) to the client. At the end, there is Program class which is for running the whole project. The user interactive parts are here.

Description of database structure designed:

In our project (which is a console C# application), we have 6 classes. The first one is the Program class, which is designed for running our project and the Main function is there. Then we have Transaction class, which is supposed to be called when we want to make a transaction. It has 3 properties called amount, FromAddress, ToAddress and a constructive method (it will assign the passing parameters to the defined properties). You can see the class below:

```
14 references
public class Transaction
{
    //Payer
    2 references
    public string FromAddress { get; set; }
    //Receiver
    2 references
    public string ToAddress { get; set; }
    3 references
    public int Amount { get; set; }

    2 references
    public Transaction(string fromAddress, string toAddress, int amount)
    {
        FromAddress = fromAddress;
        ToAddress = toAddress;
        Amount = amount;

    }
}
```

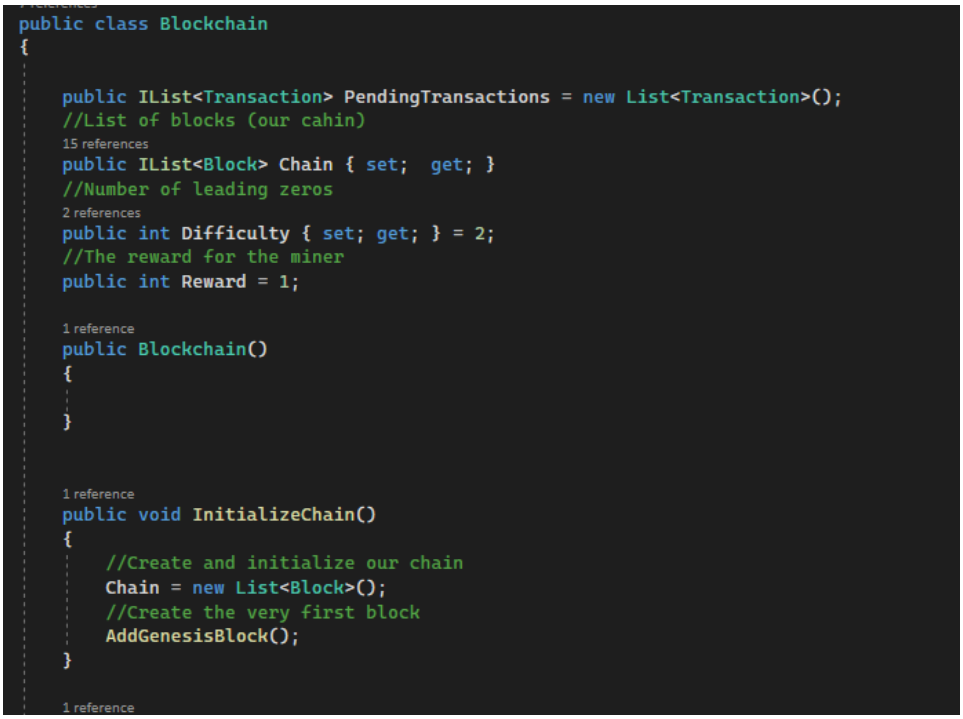
In addition, we have Block class that is implemented for saving information of a block and having some useful functions. The properties and their explanation (in form of comment) are in the picture below:

```
public class Block
{
    //Which block it is
    3 references
    public int Index { get; set; }
    //Time of the transaction
    2 references
    public DateTime TimeStamp { get; set; }
    //Hash of the previous block
    4 references
    public string PreviousHash { get; set; }
    //Hash of the current block
    7 references
    public string Hash { get; set; }

    //Transaction data
    //Rememeber that in a block we can have more than one transaction
    4 references
    public IList<Transaction> Transactions { get; set; }

    //verifying the source data that matches with the generated hash is trivial.
    //Because a specific piece of data can only get a specific hash, the source data must be changed to generate a different hash.
    //This is solved by introducing "nonce" in the data structure.
    //The nonce is an integer. By increasing the nonce, the hash algorithm can generate a different hash.
    //This process will be ended until the generated hash meets the requirement, we call it difficulty.
    2 references
    public int Nonce { get; set; } = 0;
```

In this class we have a function called CalculateHash which is supposed calculate hash of the block based on the information that the block has by SHA256 algorithm. Also, we have a Mine function that will do the mining process based on the difficulty (number of leading zeros) that we pass to it.

```
public Block(DateTime timeStamp, string previousHash, IList<Transaction> transactions)
{
    Index = 0;
    TimeStamp = timeStamp;
    PreviousHash = previousHash;
    Transactions = transactions;
}

//Calculate hash of the currunt block
2 references
public string CalculateHash()
{
    SHA256 sha256 = SHA256.Create();

    //Combine all the needed for creating hash of the current block
    byte[] inputBytes = Encoding.ASCII.GetBytes($"{TimeStamp}-{PreviousHash ?? ""}-{JsonConvert.SerializeObject(Transactions)}-{Nonce}");
    byte[] outputBytes = sha256.ComputeHash(inputBytes);

    //Convert into a string
    return Convert.ToBase64String(outputBytes);
}

2 references
public void Mine(int difficulty)
{
    //Number of leading zeros (called 'difficulty' technically)
    var leadingZeros = new string('0', difficulty);

    //Brute-Forcing for finding the desired hash
    while (this.Hash == null || this.Hash.Substring(0, difficulty) != leadingZeros)
    {
        this.Nonce++;
        this.Hash = this.CalculateHash();
    }
```

One of our important classes is the Blockchain class, which is responsible for creating, initializing and updating the chain of blocks. Here we have Reward property, which is the default amount that will be given to the miner for mining (remember that in our system the miner is the payer). Also, we have a property called Difficulty for specifying number of leading zeros in the program for mining. And we have two lists, PendingTransactions and Chain. The first one basically works as a waiting list for transactions so they would be able to added to the next block. Then we have a function called Initialize chain for initializing and creating a chain



```
public class Blockchain
{

    public IList<Transaction> PendingTransactions = new List<Transaction>();
    //List of blocks (our cahin)
    15 references
    public IList<Block> Chain { set;  get; }
    //Number of leading zeros
    2 references
    public int Difficulty { set; get; } = 2;
    //The reward for the miner
    public int Reward = 1;

    1 reference
    public Blockchain()
    {

    }


    1 reference
    public void InitializeChain()
    {
        //Create and initialize our chain
        Chain = new List<Block>();
        //Create the very first block
        AddGenesisBlock();
    }

    1 reference
```

Moreover, we have created a function called CreateGenensisBlock in order to create the very first block of the chain and then we have a function called AddGenesisBlock which is for adding the first created block to the chain (which is already empty). Then we have two other functions called GetLatestBlock and CreateTransaction which are understandable from their names.

```
public Block CreateGenesisBlock()
{
    //give the block its necessary inputs (for time we give the current time, for previous hash we give null because it is the first block)
    //(and for transactions list we give PendingTransactions which is empty at the begining)
    Block block = new Block(DateTime.Now, null, PendingTransactions);
    //Start mining for our current transaction
    block.Mine(Difficulty);
    PendingTransactions = new List<Transaction>();
    return block;
}

1 reference
public void AddGenesisBlock()
{
    //Add the very first block to our chain
    Chain.Add(CreateGenesisBlock());
}

2 references
public Block GetLatestBlock()
{
    //Get the last block
    return Chain[Chain.Count - 1];
}

2 references
public void CreateTransaction(Transaction transaction)
{
    //Add the transaction to the list of transactions of the block
    PendingTransactions.Add(transaction);
}
1 reference
```

Also, we have another function is this class called ProcessPendingTransactions which is basically responsible for clearing the waiting list of transactions and add them to a new block. Also, at the end of this block, the reward of the miner will be transferred to his/her wallet. There is a function in this class specialized for checking the validity of the chain called IsValid. This function will basically check whether the previous hash property of the current block equals to the hash of the previous block (Just to check that the hash is not altered by somebody else) and it will also compare the saved hash value of the current block and recalculated hash of the current just in case of a malicious alteration. At the end of this class, we have a function for calculating the balance of a user called GetBalance. This function traces every transaction in the list and it will do calculations on the user balance. It is worth to mention that every user at the beginning would have 100 in his/her balance.

```
public void ProcessPendingTransactions(string minerAddress)
{
    //Update the information for the new block (add the transactions related to block to it)
    Block block = new Block(DateTime.Now, GetLatestBlock().Hash, PendingTransactions);
    AddBlock(block);

    //Renew the transactions list
    PendingTransactions = new List<Transaction>();
    //Create a new transaction for rewarding the miner (this transaction will be added to the next block)
    CreateTransaction(new Transaction(null, minerAddress, Reward));
}

1 reference
public void AddBlock(Block block)
{
    //Get the current last block
    Block latestBlock = GetLatestBlock();
    //Set the index of the new block
    block.Index = latestBlock.Index + 1;
    block.PreviousHash = latestBlock.Hash;
    //block.Hash = block.CalculateHash();
    //Mine for the new block
    //PROOF OF WORK
    block.Mine(this.Difficulty);
    //Add the block to the chain
    Chain.Add(block);
}

//Check if anything has been modified by a malware or anything
2 references
public bool IsValid()
{
    for (int i = 1; i < Chain.Count; i++)
    {
        Block currentBlock = Chain[i];
        Block previousBlock = Chain[i - 1];

        //check if the current block's hash is altered by recalculating hash of the block
        if (currentBlock.Hash != currentBlock.CalculateHash())
        {
            //If they are not equal, the block and the chain have a problem and is not valid
            return false;
        }

        //Check whether the previous block's hash is changed in anyway
        if (currentBlock.PreviousHash != previousBlock.Hash)
```

```
            if (currentBlock.Hash != currentBlock.CalculateHash())
            {
                //If they are not equal, the block and the chain have a problem and is not valid
                return false;
            }

            //Check whether the previous block's hash is changed in anyway
            if (currentBlock.PreviousHash != previousBlock.Hash)
            {
                return false;
            }
        }

        //If everything is ok, the chain is valid
        return true;
    }

    //Get the current balance of an user
    5 references
    public int GetBalance(string address)
    {
        //Initial balance for everyone
        int balance = 100;

        for (int i = 0; i < Chain.Count; i++)
        {
            for (int j = 0; j < Chain[i].Transactions.Count; j++)
            {
                var transaction = Chain[i].Transactions[j];

                //if he is paying, the amount will be removed from his account
                if (transaction.FromAddress == address)
                {
                    balance -= transaction.Amount;
                }


                //if he is receiving money, his balance will be increased
                if (transaction.ToAddress == address)
                {
                    balance += transaction.Amount;
                }
            }
        }

        return balance;
    }
```

P2PClient class is actually vital for our project. Because in this and the next class synchronization will be achieved (by socket server and client). In this class we have several methods, properties and event handlers by getting help from WebSocketSharp library in C#. Firstly, we have wsDict dictionary which is responsible for saving the URL of the sever. Then we have Connect function which is able to connect, receive, send data to the server. At the beginning of this function, when the given server sends us anything, the onMessage event handler will be called to handle that. if the server that we passed its URL to the function, send us a string (in very beginning of the connection) which its first 9 characters is “Hi Client” (this process is known as hand shacking). After making sure that connection is established (in the else part), we will decode the data (in Json form) that server sent to us. In this part there is a condition for checking that which chain is longer between the one that server sent and the one that client already has (this is a famous protocol for validity in Bitcoin) so that we can sync the data between two users (client and the server). It is worth to mention that, in this function in line 54 and 55 we will connect to the server and send a message for hand shacking. The important screenshots of this class is shown below:

```
public class P2PClient
{
    //A dictionary(like in python) for saving server information, such as link
    IDictionary<string, WebSocket> wsDict = new Dictionary<string, WebSocket>();

    //The function for connecting the server
    //the url is the link of the server that the client will be connected to
    1 reference
    public void Connect(string url)
    {
        //Check whether the client is already connected to a server or not
        if (!wsDict.ContainsKey(url))
        {
            //Create a new websocket object for connecting to the server
            WebSocket ws = new WebSocket(url);

            //By onMessage we can handle message events from the server
            ws.OnMessage += (sender, e) =>
            {
                //Check if the server sent 'Hi Client' (for handshacking at the beginning)
                if (e.Data.Substring(0,9) == "Hi Client")
                {
                    //Write the message in the console
                    Console.WriteLine(e.Data);
                    //Set nameOtherSide
                    Program.nameOtherSide=e.Data.Substring(16);

                }
```

```
                //This will be executed for interactions after handshacking
                else
                {
                    Blockchain newChain = JsonConvert.DeserializeObject<Blockchain>(e.Data);
                    if (newChain.IsValid() && newChain.Chain.Count > Program.PhillyCoin.Chain.Count)
                    {
                        List<Transaction> newTransactions = new List<Transaction>();
                        newTransactions.AddRange(newChain.PendingTransactions);
                        newTransactions.AddRange(Program.PhillyCoin.PendingTransactions);

                        newChain.PendingTransactions = newTransactions;
                        Program.PhillyCoin = newChain;
                    }
                }
            };
            ws.Connect();
            ws.Send($"Hi Server, I'am {Program.name}");
            ws.Send(JsonConvert.SerializeObject(Program.PhillyCoin));
            wsDict.Add(url, ws);
        }
    }
```

For the server side, we have a class called P2PServer. This class extends WebSocketBehavior class (which is from WebSocketSharp). Here we have a

property which is kind of a flag, for checking if the data is synced between the client and the server. Then we have Start function which will initialize the server URL and its port (the port will be given by the user in the command line). Again here, we have onMessage event handler but in the form of a function (we have override it). Almost everything is same from here with P2PClient class. You can see important parts of the code below (I've written some comments in the code which may be useful for you):

```
//WebSocketBehaviour is a class for servicing clients in the server
3 references
public class P2PServer: WebSocketBehavior
{
    //For checking if everything is synced
    bool chainSynched = false;
    WebSocketServer wss = null;

    1 reference
    public void Start()
    {
        //Set the server
        wss = new WebSocketServer($"ws://127.0.0.1:{Program.Port}");
        //Set the service
        wss.AddWebSocketService<P2PServer>("/Blockchain");
        wss.Start();
        Console.WriteLine($"Started server at ws://127.0.0.1:{Program.Port}");
    }

    0 references
    protected override void OnMessage(MessageEventArgs e)
    {
        if (e.Data.Substring(0,9) == "Hi Server")
        {

            Console.WriteLine(e.Data);
            //Set nameOtherSide
            Program.nameOtherSide = e.Data.Substring(16);
            Send($"Hi Client, I'am {Program.name}");


        }
```

```
else
{
    //Get the data from client and decode it from json
    Blockchain newChain = JsonConvert.DeserializeObject<Blockchain>(e.Data);

    //Check if the chain is valid and choose the longest chain (we must choose longest chain for validity always)
    if (newChain.IsValid() && newChain.Chain.Count > Program.PhillyCoin.Chain.Count)
    {
        List<Transaction> newTransactions = new List<Transaction>();
        newTransactions.AddRange(newChain.PendingTransactions);
        newTransactions.AddRange(Program.PhillyCoin.PendingTransactions);

        newChain.PendingTransactions = newTransactions;
        Program.PhillyCoin = newChain;
    }

    if (!chainSynched)
    {
        Send(JsonConvert.SerializeObject(Program.PhillyCoin));
        chainSynched = true;
    }
}
```

At the end, in the program class, we will get name of the user and the port (will be explained later). Then we will print textual user interface. It is worth to mention that before doing a transaction, our system will check if the user has the needed balance or not.

Description of tools you used for implementation of your problem and the details of their installation and preparation for usage:

In this project we've used C# programming language and it's recommended that you have Visual Studio for Building and compiling the program. Also, for running the code you may need .NET Core 2.0 Runtime which you can download from here: https://dotnet.microsoft.com/en-us/download/dotnet/2.0/runtime.

In the program we have used two external packages called Newtonsoft.Json (which is for working with Json format) and WebSocketSharp (which is for socket programming). You can download them from "Manage NuGet Packages". The following links can be useful:

https://youtu.be/69AVDgp9NSc

https://youtu.be/ThiAQAB5Dp4

Also it is recommended that you use Windows.

Description of developed program

Our developed application is about cryptocurrency system using blockchain technology. For this system the intended users are buyers and sellers of crypto currencies. It's purpose is to allow users to easily have these transaction with no worries of any harm since it is secured using the block chain. To use the system the user must interact using visual studio and command line interface.It can support many users but they should be connected to the same server in the command line (described below in the user guide how to connect them to the server using socket programming). The features the system includes: transferring money to other users, receiving money from other users, displaying the blockchain in different accounts, mine coins and also checking the balance of the users.

User guide of the system:

First of all, you need to have Microsoft visual studio and have the C# extension with JSON and WebSocketSharp packages installed in it. Secondly, you have to download and extract our file that contains the code named as (P2P Network). After downloading the file that contains the code, you have to run the code named as BlockchainDemo in Microsoft visual studio one time. After that you have to create a directory in command line whereas the directory should be inside the Blockchain Demo file and inside the bin and Debug files also, going to the netcore app file. Like this u will be inside the directory and then u will have to write inside this directory (dotnet BlockchainDemo.dll) and write beside it any port number (for example: 6000) and then write the name of the buyer. Last but not the least, after doing all these steps you will have to open another command line and do the same thing exactly except you have to type another port number and the name of the client (seller).

Finally, you have to connect the buyer with the seller by pressing 1 (connect to a server) and then copy the url of the buyer and paste it in the seller's command line. Like this you will be able to do transactions and mine coins and also display the blockchain with all transactions.

Description of conducted tests and their results with screenshots:

Figure 1, shows the user interface of the system and also shows the buyer(server) account

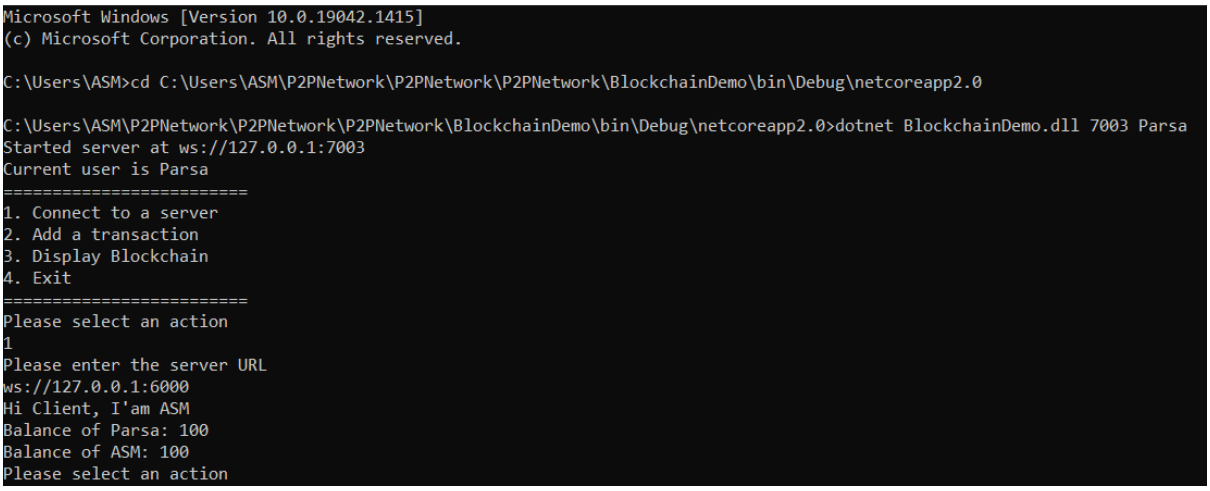


Figure 2, shows the seller's(client) account and also shows that it is connected to the buyer's account and showing the balance of both accounts

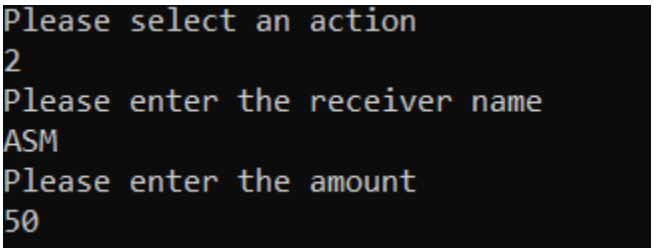


Figure 3, shows how the client sends the server some money.

```
Please select an action
3
Blockchain
{
  "PendingTransactions": [
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    }
  ],
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T17:32:53.5836078+02:00",
      "PreviousHash": null,
      "Hash": "00KKF6is2AzCql7Xhyj1hg2bmmYyUn9qc98x7sPyXtw=",
      "Transactions": [],
      "Nonce": 456
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T17:33:37.3455635+02:00",
      "PreviousHash": "00KKF6is2AzCql7Xhyj1hg2bmmYyUn9qc98x7sPyXtw=",
      "Hash": "00bss0S+LpNd90Wcu5xAJZTPj3F3ZAsxwhj9ShmP0TE=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
        }
      ],
      "Nonce": 14536
    }
  ],
  "Difficulty": 2
}
Balance of Parsa: 50
Balance of ASM: 150
Please select an action
```


Figure 4, displays the blockchain in the client's account, showing also the transactions and time for finding the hash and the previous and current hash value and the number of tries to find the hash(nonce). Showing also the number of leading zero's (difficulty).

```
Please select an action
Hi Server, I'am Parsa
3
Blockchain
{
  "PendingTransactions": [
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    }
  ],
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T17:32:53.5836078+02:00",
      "PreviousHash": null,
      "Hash": "00KKF6is2AzCql7Xhyj1hg2bmmYyUn9qc98x7sPyXtw=",
      "Transactions": [],
      "Nonce": 456
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T17:33:37.3455635+02:00",
      "PreviousHash": "00KKF6is2AzCql7Xhyj1hg2bmmYyUn9qc98x7sPyXtw=",
      "Hash": "00bss0S+LpNd90Wcu5xAJZTPj3F3ZAsxwhj9ShmP0TE=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
        }
      ],
      "Nonce": 14536
    }
  ],
  "Difficulty": 2
}
Balance of ASM: 150
Balance of Parsa: 50
Please select an action
```

Figure 5, checks that the blockchain of the server is same as the blockchain of the client and displaying all information.

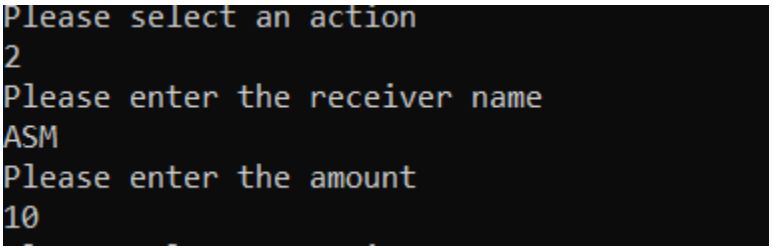


Figure 6, shows that the user Parsa sends money to the user ASM.

```
Please select an action
3
Blockchain
{
  "PendingTransactions": [
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    }
  ],
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T20:08:07.6966458+02:00",
      "PreviousHash": null,
      "Hash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Transactions": [],
      "Nonce": 31448
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T20:09:00.3208279+02:00",
      "PreviousHash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Hash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
        }
      ],
      "Nonce": 1050
    },
    {
      "Index": 2,
      "TimeStamp": "2022-01-09T20:09:23.4659821+02:00",
      "PreviousHash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Hash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
      "Transactions": [
        {
          "FromAddress": null,
          "ToAddress": "Parsa",
          "Amount": 1
        },
```

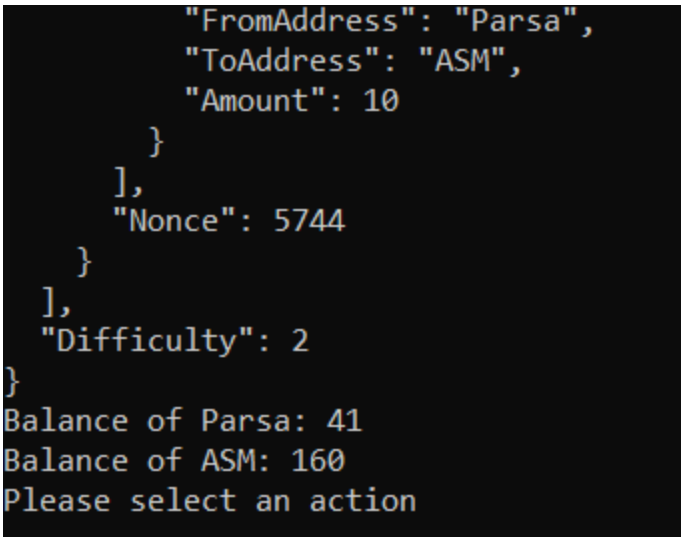


Figure 7, shows the blockchain after the second transaction and it shows that the user Parsa have mined one coin as a reward of doing the transaction and finding the hash and showing all other details.

```
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T20:08:07.6966458+02:00",
      "PreviousHash": null,
      "Hash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Transactions": [],
      "Nonce": 31448
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T20:09:00.3208279+02:00",
      "PreviousHash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Hash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
        }
      ],
      "Nonce": 1050
    },
    {
      "Index": 2,
      "TimeStamp": "2022-01-09T20:09:23.4659821+02:00",
      "PreviousHash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Hash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
      "Transactions": [
        {
          "FromAddress": null,
          "ToAddress": "Parsa",
          "Amount": 1
        },
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 10
        }
      ],
      "Nonce": 5744
    }
  ],
  "Difficulty": 2
}
Balance of ASM: 160
Balance of Parsa: 41
Please select an action
```


Figure 8, checks that the blockchain of the user ASM is same as the user Parsa, and as you can see it is exactly same.

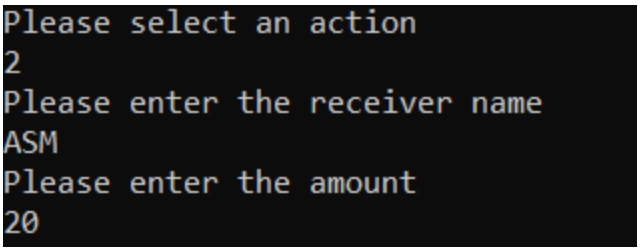


Figure 9, shows another transaction from Parsa to ASM.

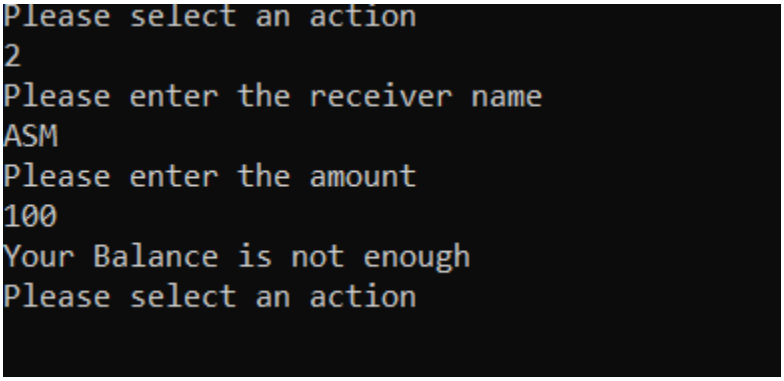


Figure 10, shows that Parsa wanted to send ASM more than what he have in his wallet so no transaction happened as his balance is not enough

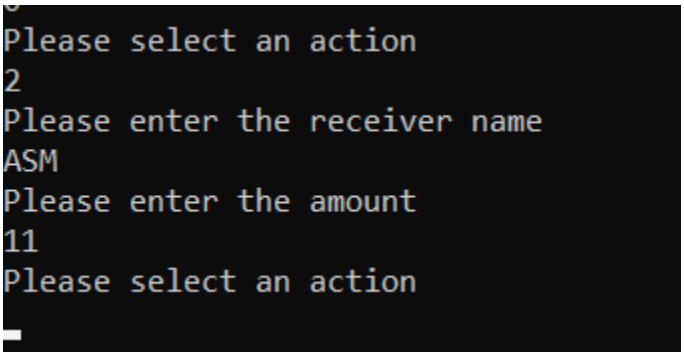


Figure 11, shows another transaction from Parsa to ASM.

```
Blockchain
{
  "PendingTransactions": [
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    }
  ],
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T20:08:07.6966458+02:00",
      "PreviousHash": null,
      "Hash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Transactions": [],
      "Nonce": 31448
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T20:09:00.3208279+02:00",
      "PreviousHash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Hash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
        }
      ],
      "Nonce": 1050
    },
    {
      "Index": 2,
      "TimeStamp": "2022-01-09T20:09:23.4659821+02:00",
      "PreviousHash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Hash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
      "Transactions": [
        {
          "FromAddress": null,
          "ToAddress": "Parsa",
          "Amount": 1
        },
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 10
        }
      ],
```

```
    "Nonce": 5744
  },
  {
    "Index": 3,
    "TimeStamp": "2022-01-09T20:15:25.3193611+02:00",
    "PreviousHash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
    "Hash": "003WA2eq7n1j9/xauIXtVYJ/N7lhoCecrVTQvo9ATJY=",
    "Transactions": [
      {
        "FromAddress": null,
        "ToAddress": "Parsa",
        "Amount": 1
      },
      {
        "FromAddress": "Parsa",
        "ToAddress": "ASM",
        "Amount": 20
      }
    ],
    "Nonce": 3737
  },
  {
    "Index": 4,
    "TimeStamp": "2022-01-09T20:18:17.6225589+02:00",
    "PreviousHash": "003WA2eq7n1j9/xauIXtVYJ/N7lhoCecrVTQvo9ATJY=",
    "Hash": "00TTockGIdIelPwEn+wpcx/sXE/EjEW4gHBXp+eRqm4=",
    "Transactions": [
      {
        "FromAddress": null,
        "ToAddress": "Parsa",
        "Amount": 1
      },
      {
        "FromAddress": "Parsa",
        "ToAddress": "11",
        "Amount": 0
      }
    ],
    "Nonce": 1044
  },
  {
    "Index": 5,
    "TimeStamp": "2022-01-09T20:18:25.3643231+02:00",
    "PreviousHash": "00TTockGIdIelPwEn+wpcx/sXE/EjEW4gHBXp+eRqm4=",
    "Hash": "00fSONAdfh9L7B5mHwVVQ6qX4Fcjpwim1aS8LU0o5+E=",
    "Transactions": [
      {
        "FromAddress": null,
```

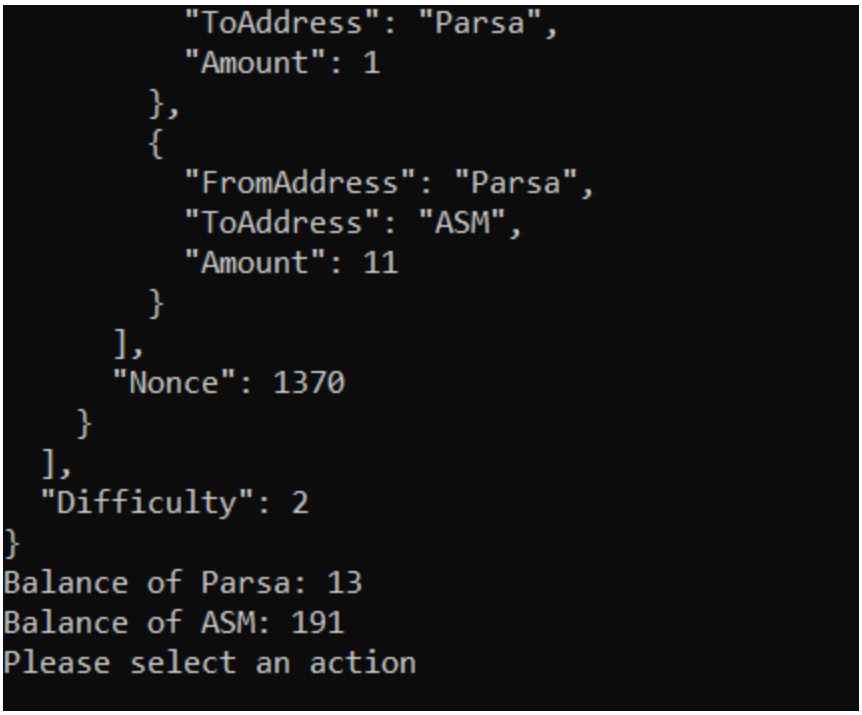


Figure 12, displays the blockchain of the user Parsa.

```
Blockchain
{
  "PendingTransactions": [
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    },
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    },
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    },
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    },
    {
      "FromAddress": null,
      "ToAddress": "Parsa",
      "Amount": 1
    }
  ],
  "Reward": 1,
  "Chain": [
    {
      "Index": 0,
      "TimeStamp": "2022-01-09T20:08:07.6966458+02:00",
      "PreviousHash": null,
      "Hash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Transactions": [],
      "Nonce": 31448
    },
    {
      "Index": 1,
      "TimeStamp": "2022-01-09T20:09:00.3208279+02:00",
      "PreviousHash": "00cL1byCsBCpgMUaeS/dmn7uIFo3P8n0X42TSIWfg/Q=",
      "Hash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
      "Transactions": [
        {
          "FromAddress": "Parsa",
          "ToAddress": "ASM",
          "Amount": 50
```

```
    ],
    "Nonce": 1050
  },
  {
    "Index": 2,
    "TimeStamp": "2022-01-09T20:09:23.4659821+02:00",
    "PreviousHash": "00bNWb01Ad1E3mc9sgbsLxPZG8ar1F9krcs3czhwAV4=",
    "Hash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
    "Transactions": [
      {
        "FromAddress": null,
        "ToAddress": "Parsa",
        "Amount": 1
      },
      {
        "FromAddress": "Parsa",
        "ToAddress": "ASM",
        "Amount": 10
      }
    ],
    "Nonce": 5744
  },
  {
    "Index": 3,
    "TimeStamp": "2022-01-09T20:15:25.3193611+02:00",
    "PreviousHash": "00Nh5YqS1Ki14J7x3TZmVAgiVMRMzO0JxEvcfx5T+qE=",
    "Hash": "003WA2eq7n1j9/xauIXtVYJ/N7lhoCecrVTQvo9ATJY=",
    "Transactions": [
      {
        "FromAddress": null,
        "ToAddress": "Parsa",
        "Amount": 1
      },
      {
        "FromAddress": "Parsa",
        "ToAddress": "ASM",
        "Amount": 20
      }
    ],
    "Nonce": 3737
  },
  {
    "Index": 4,
    "TimeStamp": "2022-01-09T20:18:17.6225589+02:00",
    "PreviousHash": "003WA2eq7n1j9/xauIXtVYJ/N7lhoCecrVTQvo9ATJY=",
    "Hash": "00TTockGIdIelPwEn+wpcx/sXE/EjEW4gHBXp+eRqm4=",
    "Transactions": [
      {
        "FromAddress": null,
        "ToAddress": "Parsa",
```

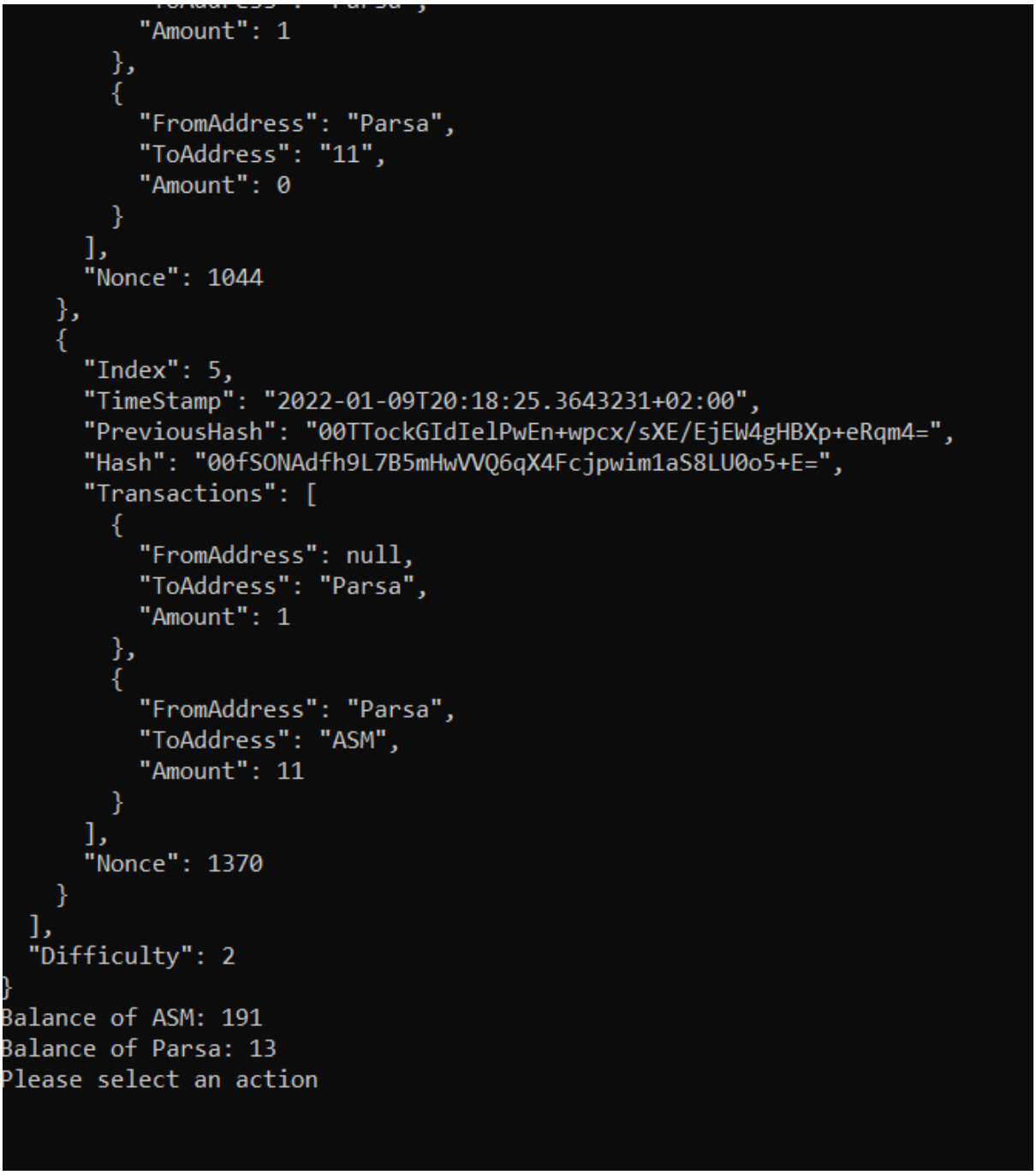


Figure 13, displays the blockchain of the user ASM, and as you can see it is totally same as the blockchain of the user Parsa.

Conclusion

In conclusion, the digital age has come upon us and is, therefore, digitalizing most of our lives. A cryptocurrency would soon replace the paper currency that we now use. Our simple cryptocurrency, that has been coded using C# and whose server has been connected to the users by means of socket programming techniques has implemented the blockchain technology in order to ensure the integrity and security of our project. Our project works using Microsoft Visual Studio having C# extension, JSON and .NET Core 2.0 packages installed, along with Newtonsoft.JSON and WebSocketSharp external packages. When a buyer and seller establish a connection using our server by using Visual Studio and the command line interface transactions such as: transferring money to other users, receiving money from other users, displaying the blockchain in different accounts, mine coins and also checking the balance of the users.

References

https://medium.com/coinmonks/python-tutorial-build-a-blockchain-713c706f6531

https://geekflare.com/create-a-blockchain-with-python/

https://youtu.be/69AVDgp9NSc

https://youtu.be/ThiAQAB5Dp4

https://dotnet.microsoft.com/en-us/download/dotnet/2.0/runtime.

https://www.c-sharpcorner.com/