Consensus algorithm testing

By Alexander Winning

Contents

Contents

Section 1: Analysis

Problem definition

Currently, there is no way for people developing new consensus algorithms to test them without creating their own blockchains. This causes the following issues:

- Increased time of development for start-ups in blockchain development
- Increased initial cost for start-ups in blockchain development
- Increased risks for start-ups in blockchain development
- Additional risks for investors in start-ups in blockchain development

What is a consensus algorithm?

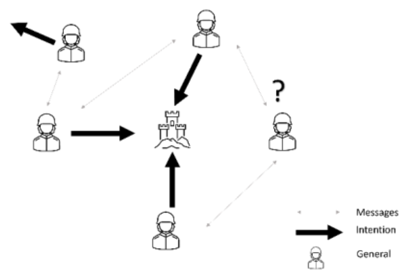
Consensus algorithms endeavour to solve a problem with distributed computer systems known as the Byzantine General's problem. Consensus algorithms are fundamental to any distributed network as they facilitate decision-making. E.g., in a blockchain environment they determine whether the order of transactions is valid.

Byzantine General's Problem

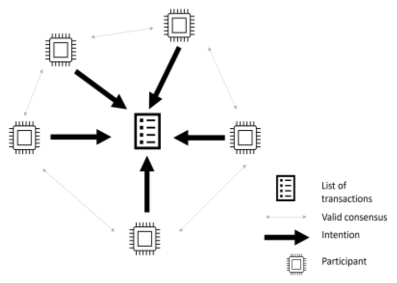
In the Byzantine general's problem, several generals are attacking a castle and must decide when to attack so that they can be successful. They can only communicate through messengers and these messengers may get captured on the way to their destination. This means there is a risk that they will not deliver the original message at all or deliver a changed message. This leads to the issue that the other generals don't know whether the message they receive is correct or truthful which can ultimately lead to the failure of the attack on the castle. In order to resolve the problem each general needs a system that gives more clarity on what to do e.g., they could ask for a secret code to authenticate the message.

How does the Byzantine General's Problem apply to blockchain?

Decentralised systems don't have a centralised entity telling them what is correct, e.g., there is no way of verifying that all participants have the same information. This means they are therefore at risk of experiencing the Byzantine General's Problem. A blockchain is a decentralised system that attempts to overcome this problem by using a consensus algorithm



The Byzantine Generals Problem



The Blockchain Application of BGP

The diagram shows the different participants in the network and their intention to verify their own version of the list of transactions. For someone to transact on a blockchain their transaction gets sent to the memory pool where a miner/participant will pick it out and include it in their version of the list of transactions. If they achieve valid consensus (i.e., they satisfy the consensus algorithm first) the transaction is included in the proposed block. The network now agrees on that transaction being valid.

How does a consensus algorithm solve this application of the Byzantine General's Problem?

A consensus algorithm is a set of objective rules that every node of the network can use to check that a block is valid. Any dishonest participant trying to broadcast false information would be stopped by the consensus algorithm as all nodes on the network would be able to verify the information was false due to the presence of the consensus algorithm which would check the block against a set of criteria.

Testing a consensus algorithm

Due to the fact that consensus algorithms provide security for the whole network, stringent testing of their functionality and possible attack vectors is required before they are entrusted to a network. To test a consensus algorithm a mock-up of a network is required. Currently this is done with public and private test networks (testnets) developed specifically for the network they are testing, meaning they are unusable for testing anything else. With my solution, I intend to create a versatile test network code that can be deployed by the developer specifically for their consensus algorithm.

Developing your own test network is costly and can be risky as you are investing in a consensus algorithm that may not work. My solution aims to solve that problem by having a ready-to-go test network for third parties to test their consensus algorithms with little upfront cost.

Why is the problem suited to a computational solution?

The problem lends itself to computational methods of discovery and development of a solution for many reasons. These are:

- The solution will be software that leverages networking and computation to test consensus algorithms. This will need to run on computers as the algorithm it will be testing will be run on computers and cannot be computed by a human in an appropriate time due to its complexity.
- The solution takes inputs from other nodes through networking in the form of chunks of data. This data will then be put in a sequence of instructions and branched out of the sequence depending on the input. A human could not understand the input in a reasonable amount of time, and this means it has to be run on computers as they can process the input extremely quickly.
- The program will send chunks of information to other nodes that it is aware of. A human could not effectively transfer this data due to its size and complexity, but computers can do this.
- The solution will always finish within a constant number of steps, making this suitable for programming.

The general requirement for this solution would be a computer that is capable of processing message hashing and verification which is not very high demand at all. In its most abstract form, the solution would take inputs of clumps of transactions, verify, save and propagate them to other computers it is aware of. This problem lends itself to a computational approach as the processing latency required invalidates everything but computers as the augmentation of the data required would take a human a large amount of time.

Computational methods relevant to the discovery of the solution

Problem recognition

The overall problem is the lack of ready-to-go test environments for consensus algorithm developers. The underlying problem will be setting up networking between nodes. After this is tackled the inputs will be formatted and passed to appropriate functions depending on the input. In the case of receiving a block, the inputs will be sent to a verification function that will check the block against consensus parameters and will verify all transactions. The verification function will then pass the block to a distribution function which will send it out to all known nodes.

Solution decomposition

The proposed solution can be broken down into segments that can be worked on and tested independently from the main program. My initial idea of what the modules would be are:

- Setting up a socket on the host computer and dealing with incoming connections
- Checking what is received against conditions
- Balance and transaction count lookup
- Block validation
- Transaction validation
- Creating a memory pool and adding valid transactions to it
- Executing all transactions in the valid and accepted block
- Persistent saving of blocks and address information
- Distribution of information for valid blocks and transactions

When all these sections are completed, I will have a fully functioning blockchain core program that can be used to test consensus algorithms and be distributed in testing to form a mini blockchain.

Divide and conquer

These smaller steps are easily attainable and can be implemented into the main program using subroutines. This will help with readability and is making use of the divide-and-conquer computational method.

My client

The clients of this software are blockchain developers, which would range from large companies to small one-man developers. I will be focussing on smaller scale blockchain developers and especially start-ups as they will have limited resources to create a version of my solution on their own.

I will be analysing my project using a representative sample from one-man development teams, start-ups and computer scientists not directly linked to blockchain. This will help validate my solutions' usefulness even to people in the wider space that I would not specifically target. I will conduct a series of interviews throughout the project to get their opinion on specific items that I will implement.

Initial client interviews

Questions for Blockchain Developers:

1. Are resources (money, time, and manpower) the limiting factor for development speed?
2. Does your Start-up depend on the software that you are creating?
3. What is your experience with testing?
4. Have you ever used a test environment that was not created in-house?
5. How many people are on your development team?
6. Are you aware of the idea of a test network?
7. Would you be interested in using a testing framework created externally?

Questions for Computer scientists:

1. How aware of consensus algorithms are you?
2. Have you worked with distributed networks?
3. Are you aware of the Byzantine generals' problem?
4. What is your experience in application testing?
5. How often do you use test environments and for what?
6. Would you use a test environment created by someone else?

Response Blockchain Developer (Start-up):

1. Are resources the limiting factor for development speed?
    - In some circumstances, yes but in others, it is just the required time of creating software. The circumstances where it is, are usually for complex projects such as blockchains
2. Does your start-up depend on the software that you are creating?
    - Yes, the main service that the start-up is aiming to provide is a blockchain and so the success of the start-up is centred around our consensus algorithm being airtight.
3. What is your experience with testing?
    - In the past, I had worked for a different start-up that failed due to a bug in the smart contract. I believe that had the testing team made a program to specifically test the smart contract before deployment this could have been avoided as it was an unchecked function that was missing a limiter on use.
4. Have you ever used a test environment that was not created in-house?

  - I have not. The test environments I come across are all made in-house though I know of people who have, and they spend a lot of time testing that before getting on to testing the thing it was created to test.
5. How many people are on your development team?
   - Right now, 15 but there is a range from start-up to start-up usually in a range of 6 - 25
6. Are you aware of the idea of a test network?
   - Yes, I have been a part of creating one in the past
7. Would you be interested in using a testing framework created externally?
   - Yes, as long as the team and I could pick it up easily as to not waste time and resources learning it.

Response Blockchain Developer (One-man developer):

1. Are resources (money, and manpower) the limiting factor for development speed? •
   Most definitely, I need to develop these key areas and do not have the recourses to hire someone to help develop faster.
2. Does your Start-up depend on the software that you are creating?
   - Yes, I am creating a blockchain and its success is based on whether the consensus algorithm works.
3. What is your experience with testing?
   - I have been testing applications ever since I learned to code.
4. Have you ever used a test environment that was not created in-house?
   - In more general computer science, yes, though never in blockchain as there aren't any that I have seen.
5. How many people are on your development team?
   - One
6. Are you aware of the idea of a test network?
   - Yes, such as the Ethereum test networks and the Bitcoin signet and testnet
7. Would you be interested in using a testing network created externally?
   - Yes, if it had enough documentation and it operated well. As I wouldn't want to discard my consensus algorithm if the errors were produced by the software.

Responses from Computer scientists:

1. How aware of consensus algorithms are you?
   - I am aware of them through hearing about them being used in datacentres, but I am by no means an expert.
2. Have you worked with distributed networks?
   - I have implemented Kubernetes once with a couple of old computers but have not done much more than that
3. Are you aware of the Byzantine generals' problem?
   - No
4. What is your experience in application testing?

  - I am somewhat experienced with application testing and have formally it twice. Though ever since I learned to code, I have been testing informally.
5. How often do you use test environments and if so for what?
  - I do not use test environments very often but would only use them if it was a critical application.
6. Would you use a test environment created by someone else?

- If the test environment worked properly and was easily serviceable then of course, but in the past, I have found them to be clunky and focus on a UI too much compared to making it the best it could be

My clients:

- 1-to-25-man development teams
- Limited budget ~ (£2,000-to £50,000)
- Their success relies on their consensus algorithm

My client's needs are:

- A solution that can be easily learned and used ○ I will provide documentation on all parts of the solution which will assist the client in using the solution effectively and save time from creating it inhouse / learning a poorly documented solution
- A solution with reduced cost compared to making it inhouse ○ My solution will be available for a minimal cost which will fit within most start-ups budget
- A solution that can be developed upon ○ I will make it clear where every part of the blockchain code is and what that part does and how to include their consensus algorithm

My client will use my blockchain as a base to test their consensus algorithm and they also will be able to build upon it to create their blockchain.

Existing solutions

Private Test Network Methodology

Overview

A private test network is a version of a main network, with all the same capabilities as well as proposed future capabilities, that the public cannot access. This is why private test networks are so valuable to development teams as they are a version of the main network that can be used to test for vulnerabilities without compromising the security of the live network. As a result of this changes can be made and tested rapidly with limited risk.

Within Blockchain, private test networks are usually created in-house and are the starting point for many development teams. These test networks usually start out as little more than an initial proof of concept and aim to be built up to a viable main network through a series of upgrades and testing.

This is a time-consuming and costly process which may be unsuccessful in creating a functioning main network, for example if the consensus algorithm is found to have a security weakness.

Once the network under development has been tested successfully the code from the private test network can be deployed as a main network and made public, whilst the private test network runs in parallel. The private test network will then act as the first port of call for developing and testing proposed network upgrades. The private test network will be active throughout the life of the main network and will cycle through the following stages: implementing a change, testing the change, evaluating the change, and discarding or keeping the change. This allows for a sheltered place to design and develop which will improve the quality of the solutions.

There are a number of problems with this approach:

- There is a lot of risk with the initial private test network due to the expense it will incur during its development which may be wasted if the project is unsuccessful i.e., the project will not generate a return.
- This approach may take a long time before it produces a deployable main network which will impact the speed of the product to market.
- Testing proposed upgrades for security cannot be done on a large scale due to the privacy of the private test network. **Example:**

As these are private there is little information or examples available in the public domain, but it is obvious that they are used and required in the blockchain development lifecycle. In a published research paper by the University of Barcelona (Ribera, E.G. (2018) *Design and Implementation of a Proofof-Stake Consensus Algorithm for Blockchain*. thesis.) reference is made to the use of a “Beta Network” in the testing of a Consensus algorithm. This Beta Network is not available to the public therefore I believe it to be an example of a private test network.

Features to incorporate into my solution

The features of the Private Test Network approach that I intend to incorporate into my design are:

- The made in house feel through a deep understanding of the workings and code
- The thoroughness of the documentation is necessary to be able to understand and use the testing framework easily and efficiently

I will provide the same quality and developer-friendly approach as the private test network approach but will make it available for less cost than creating one from scratch.

Many companies in the blockchain space are formed around the idea of their network and thus rely on the success of their consensus algorithm and main network and will not have the recourses to create multiple consensus algorithms and test networks.

My solution will come with all of the necessary documentation and resources so that it will take minimal time for the client’s developers to start utilising my solution.

Tried and Tested Methodology

Overview

The tried and tested method is when a blockchain uses an already-established consensus algorithm instead of creating a new one. The tried and tested method removes any risk that a consensus algorithm does not work as the consensus algorithm has been in use for an extended period of time. This removes a lot of the risk a new blockchain holds and can lead to greater trust between consumers and the blockchain as it is perceived as safer due to its widespread use.

Disadvantages of this method are:

- A Proof Of Work consensus algorithm can be at a heightened risk of 51% attacks if it is using an already established hashing algorithm. A 51% attack is when a malicious party can control the production of blocks and as such can censor specific transactions and can transact on the same coins multiple times.
- If the consensus algorithm is someone's intellectual property, there is a risk of copyright infringement if not properly licensed from them.

Example:

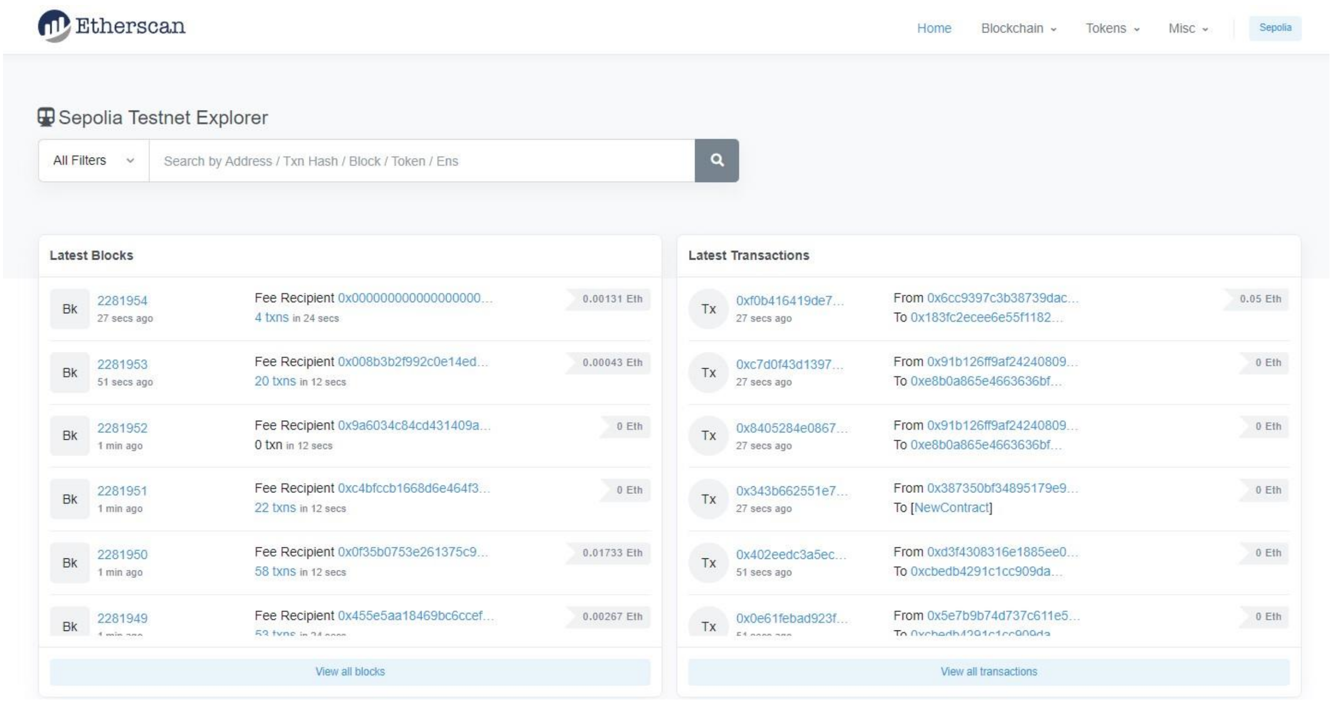
There are many companies providing a service that offers 3rd parties the ability to use private blockchains. Private blockchains are a copy of an established blockchain that’s use is restricted to only authorised parties. They are operated by the service provider who handles all of the backend and hardware. They are used for blockchain developers to test blockchain applications and in some cases where the service provider only manages the hardware can be used to test blockchain projects and consensus algorithms, for example, Amazon Web Services could enable you to run a copy of an existing blockchain as they provide hardware that any software that you submit can run on.

Features to incorporate into my solution

The features of the Tried and Tested approach that I intend to incorporate into my design are:

- Provision of a stable trusted development framework to test a consensus algorithm
- A stable base to test on is required as any faults may lead to the scrapping of the consensus algorithm - Ease of use.
- As the service provider already has a framework that you can use to test a blockchain application this leads to quicker and cheaper development as you do not need to create one yourself.

Public Test network Methodology



Sepolia Testnet Explorer

All Filters | Search by Address / Txn Hash / Block / Token / Ens

Latest Blocks

Bk	2281954 27 secs ago	Fee Recipient 0x000000000000000000... 4 txns in 24 secs	0.00131 Eth
Bk	2281953 51 secs ago	Fee Recipient 0x008b3b2f992c0e14ed... 20 txns in 12 secs	0.00043 Eth
Bk	2281952 1 min ago	Fee Recipient 0x9a6034c84cd431409a... 0 txn in 12 secs	0 Eth
Bk	2281951 1 min ago	Fee Recipient 0xc4bfccb1668d6e464f3... 22 txns in 12 secs	0 Eth
Bk	2281950 1 min ago	Fee Recipient 0x0f35b0753e261375c9... 58 txns in 12 secs	0.01733 Eth
Bk	2281949 1 min ago	Fee Recipient 0x455e5aa18469bc6ccef... 53 txns in 24 secs	0.00267 Eth

View all blocks

Latest Transactions

Tx	0xf0b416419de7... 27 secs ago	From 0x6cc9397c3b38739dac... To 0x183fc2ecee6e55f1182...	0.05 Eth
Tx	0xc7d0f43d1397... 27 secs ago	From 0x91b126ff9af24240809... To 0xe8b0a865e4663636bf...	0 Eth
Tx	0x8405284e0867... 27 secs ago	From 0x91b126ff9af24240809... To 0xe8b0a865e4663636bf...	0 Eth
Tx	0x343b662551e7... 27 secs ago	From 0x387350bf34895179e9... To [NewContract]	0 Eth
Tx	0x402eedc3a5ec... 51 secs ago	From 0xd3f4308316e1885ee0... To 0xcbedb4291c1cc909da...	0 Eth
Tx	0x0e61febad923f... 51 secs ago	From 0x5e7b9b74d737c611e5... To 0xcbedb4291c1cc909da	0 Eth

View all transactions

Overview

Public test networks are used as an intermediary step between a private test network and a main network. They allow a Blockchain project to launch an upgrade into the public domain without risk to their main network. The most common uses of public test networks are for trials of proposed network upgrades such as ***Bellatrix and Paris (The Merge) on Ethereum***. Their use for this purpose is limited to whoever owns and operates the test network. They are public in the fact that they allow the community time to try and attack the test network and its upgrades to expose weaknesses before the upgrade is pushed onto the main network. As all of the test network assets are worthless there is no financial benefit from an attack on a public test network.

A downside to public test networks is the cost of running the network.

Example:

Sepolia Test network owned by the Ethereum Foundation. This is a copy of the Ethereum Network used to test EIPs (Ethereum Improvement Proposals - upgrades suggested by members of the community and vetted by the Ethereum Foundation). The Base currency (SEP) is freely available to anyone that wants to use the network and is therefore worthless in the case that a vulnerability is discovered in the network. Anyone can access it by using the publicly available RPC URL and Chain ID.

Features to incorporate into my solution

The features of the Public Test Network approach that I intend to incorporate into my design are:

- The ability to test the network on a large scale
- Testing a wide range of scenarios is necessary to ensure the safety of the network and trust of the consumer
- The ability to have a large amount of people testing it at once

- This allows more people to check for errors and discover vulnerabilities in the consensus algorithms and is extremely valuable to the furthering and testing of any blockchain. My solution could be utilised as a quickly deployable public test network only used when upgrades are tested and then shut down afterwards. This will help the client save money as it will decrease the money spent on operating the test network as it is not working 24/7.

Features of the proposed solution:

The initial concept for my design considering my research

My solution will be a regular node which will consist of an executable file, supporting documentation and a locally hosted website.

Once you open the executable file the node will start running and it will display any important information in a popup, such as block number and node status. I decided to show the information this way so that if possible, the client could monitor multiple nodes at once as it is very easy to spot the information as long as any errors

The way to interact with your node will be through the website. Using the website, you can add and remove known nodes, add transactions, and check balances. I decided to use a website as the interface as it can be operated from a computer other than the one running the node software and also allows the node software to output useful information only.

All of the explanations of the workings of the network and the node will be available in the supporting documentation so that anyone can run and understand it. This will include theoretical topographies of the network, flow diagrams of the logic and validation, guidance on how to include your consensus algorithm, explanations of the code as a whole and a brief explanation of hashing and signature verification. I decided to add this as the feedback from the client interviews was positive towards Using a testing framework as long as they were quick to understand it. I believe that documentation will allow me to do this as it will be detailed along with comments in the code itself to the point where the code can be understood in a minimal amount of time.

Limitations of my solution

My solution focuses on consensus algorithm development with one blockchain. The limitations of my solution will increase the further the proposed network is away from a traditional blockchain. My solution would not be helpful with a consensus algorithm that controls multiple interwoven blockchains, though my code could be a helpful starting point for each of the single blockchains.

This is also not a finished blockchain and I believe to make it production ready some modification would have to be made. This does mean that development time is not fully wiped out though I believe that to be unattainable as all consensus algorithms vary in different ways and so the software could not be fully optimised for future algorithms. Though the effort needed to optimise the provided blockchain is nothing compared to creating one from scratch.

Requirements

Software and hardware requirements

Hardware

- **A computer capable of running low-demand software**
  - The software will hash data to verify blocks.
  - This will require a fast-enough processor, though most devices in use today will be capable of doing this the requirement would be to use a computer produced in the last 5 years.
  - This computer does not need monitoring and can have all of the peripherals removed post-setup.
- **Networking**
  - A connection to the internet that is faster than four hundred bytes per second.
  - This computer requires networking to be able to send and receive blocks.
  - Though not data-intensive, if a large number of requests are made to a node the amount of data throughput could be substantial.
  - All modern computers have a throughput high enough to sustain the node (~four hundred bytes per second).

Software

- **Windows operating system** ○ This can only be used in the Windows operating system as it will use Winsock2, a Windows-only library that handles networking.

Success Criteria

Qualitative

1. **Easy To Implement.** This encompasses all areas of the node software but especially comments outlining the purpose of functions, where the consensus algorithm starts, where the consensus algorithm ends, flow diagrams of the node and examples of it functioning correctly. This will be tested by asking someone completely foreign or with little insight into blockchain to place a provided consensus algorithm into the software in the correct position using the provided documentation and get detailed feedback on ease of use, areas for improvement and documentation that could be improved. This is necessary to satisfy my client's needs for a high quality and quicker solution.
2. **Test Feedback.** On the arrival of each block a report will be produced. This report will give reasons why a block was denied if this was the case or if the block is accepted will display the blocks information e.g., block number, block hash etc. The report will be tested for its usefulness by asking my clients for feedback on completeness of information provided, format and file type. **Quantitative**
3. **Validates Blocks Correctly.** This is integral to each node's ability to function properly and will be tested with an existing consensus algorithm by feeding the node valid and invalid blocks and recording the results.
4. **Validates Transactions Correctly.** This is fundamental to the security of the blockchain and uses the secp256k1 algorithm. This can be tested by submitting a block with incorrect transactions and a block with correct transactions and checking that the node rejects or accepts the block appropriately.

5. **Gossips Blocks Throughout Known Nodes.** This is how blocks propagate the network and become accepted by all other nodes, without this the network cannot function. This can be tested by sending a valid block to node A which is aware of nodes B, C and D and checking that they all received the valid block.
6. **Block Requests.** This is used by nodes that are not level with the network's block height to become up to date by requesting blocks from peers and validating them. This can be tested by turning off node B whilst node A still accepts blocks and then, after a while, restarting node B. As the next block is accepted by Node A it will send it to node B and if all is working correctly node B will request the missed blocks in order to validate the latest block.
7. **Website Queries.** The self-hosted website is used to interact with the node. This is how the node owner will interact with the network; it allows him to check his balance and an important variable called the transaction counter to formulate transactions from his addresses. The parameter being requested along with the account in question are transmitted to the node, the node then responds with the answer shortly after. This will be tested by querying different addresses real and fake and recording the results.
8. **Transaction memory pool.** The nodes need to have a memory pool of all transactions waiting to be included in blocks. The nodes transmit transactions they receive to all other nodes. This helps people send transactions by allowing them to reach more nodes which decide to include the transaction in a block. The website will allow anyone with access to input transactions and they will be sent to the local node. To test this, I will input a valid transaction to the website connected to node A and will check the memory pool of node B to make sure that it has been propagated.

Evaluation

How will you know if your solution works?

The solution will be evaluated by conducting a series of interviews with clients that have used my solution to test their consensus algorithms. The interviews will ask about the ease of use, the quality of the documentation and the effectiveness of the software.

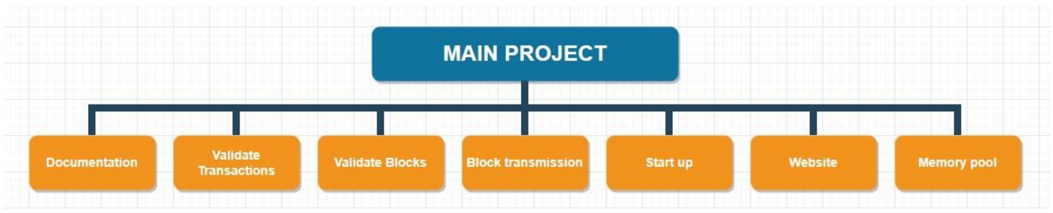
How will you know if your solution is good enough?

The solution will be considered good enough if the clients that have used it report that it is easy to use, the documentation is of good quality and the software is effective.

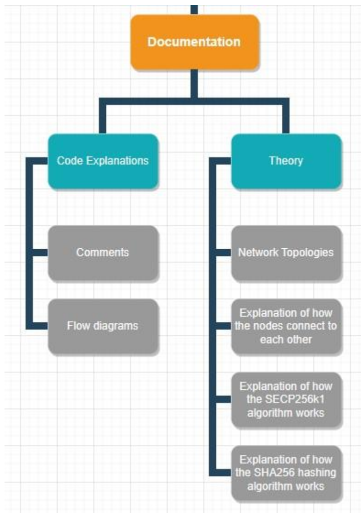
Section 2: Decomposition

Decomposition

The diagram bellow is the decomposition of the main project into 7 sub-problems. These are then broken down into points shown in the structure diagram accompanying each written breakdown. I broke the project down in this way as those are the smallest reasonable pieces of the problem. All of the input validation is carried out in Validate Transactions and Validate Blocks sections. All of the usability features are mentioned in the Documentation and Website sections.



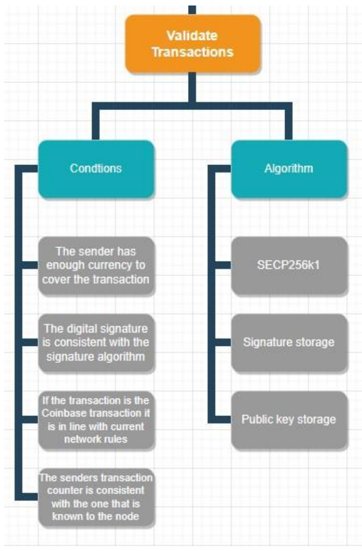
Documentation


Documentation
Code Explanations
Theory
Comments
Flow diagrams
Network Topologies
Explanation of how the nodes connect to each other
Explanation of how the SECP256k1 algorithm works
Explanation of how the SHA256 hashing algorithm works

These segments are a representation of the solutions that fix the overall problem of the node's documentation.

The Documentation will allow anyone to easily understand the code and be able to test their consensus algorithm with minimum time figuring out the code.

Validate Transactions


Validate Transactions
Condtions
Algorithm
The sender has enough currency to cover the transaction
The digital signature is consistent with the signature algorithm
If the transaction is the Coinbase transaction it is in line with current network rules
The senders transaction counter is consistent with the one that is known to the node
SECP256k1
Signature storage
Public key storage

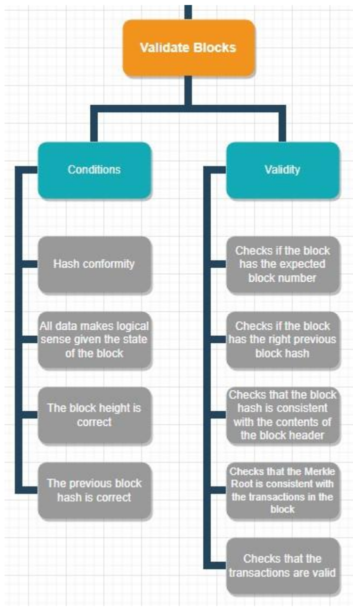
These segments are a representation of the solutions that fix the overall problem of the node's transaction validation.

Transaction Validation conditions:

- Does the sending account exist?
  - To protect against database errors for senders who don't exist - Is the signature correct?
  - Validates that the account labelled sender actually sent the transaction
- Does the sending account have a sufficient balance? ○ Validates the ability of the sender to send the funds - Is the Transaction count of the sending account correct?
  - This stops old signatures being reused by an attacking party to send a transaction twice

The algorithm I will be using is the SECP256k1 algorithm it is part of the family of signatures that uses an elliptical curve to validate signatures and is defined in the *Standards for efficient cryptography page 9* (http://www.secg.org/sec2-v2.pdf) The name secp256k1 refers to the parameters of the elliptic curve used by the algorithm. The algorithm used is not like other commonly used curves as it was constructed in a non-random way allowing for extremely efficient computation. Secp256k1s constants were selected in a predictable way which reduces the possibility of a backdoor unlike NIST curves. The library that I will use is the same one deployed in bitcoin and handles signature storage and key storage.

Validate Blocks

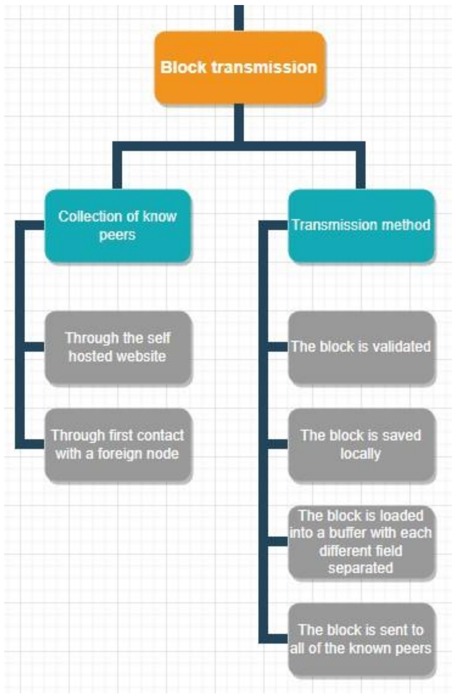

Validate Blocks
Conditions
Validity
Hash conformity
All data makes logical sense given the state of the block
The block height is correct
The previous block hash is correct
Checks if the block has the expected block number
Checks if the block has the right previous block hash
Checks that the block hash is consistent with the contents of the block header
Checks that the Merkle Root is consistent with the transactions in the block
Checks that the transactions are valid

These segments are a representation of the solutions that fix the overall problem of the node's block validation.

Block Validation conditions:

- The hash conforms with what is expected.
  - difficulty (number of 0s)
  - Matches the contents of the block header (ensures the block has not been tampered with)
- All data inside the block matches what the block declares in the relevant fields.
- The block height is correct
- The previous block hash is correct

Block Transmission


Block transmission
Collection of know peers
Through the self hosted website
Through first contact with a foreign node
Transmission method
The block is validated
The block is saved locally
The block is loaded into a buffer with each different field separated
The block is sent to all of the known peers

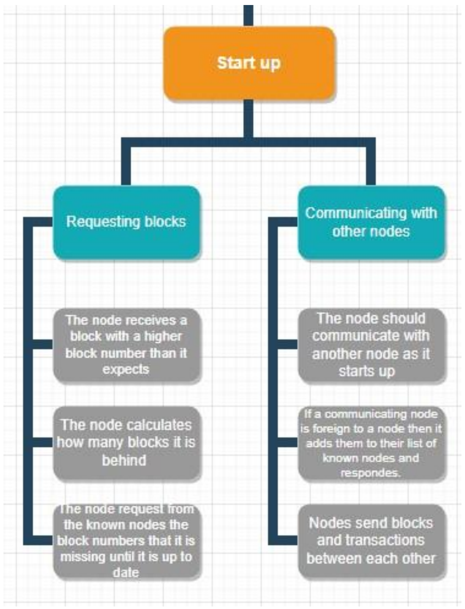
These segments are a representation of the solutions that fix the overall problem of the node's start-up and procedure.

The Transmission of blocks is a linear process that makes sure that only valid blocks get passed on to other nodes. The process is: Receipt of block, block validation, block is saved, the block is loaded into the buffer, the block is sent to all known peers. This allows any invalid block to not clog up network bandwidth and node time.

Collection of known peers:

- Through the self-hosted website ○ This allows the first known nodes to be added
- Through first contact with a foreign node ○ Nodes will be aware of who is connected to them and add a node to their list of known nodes if an unknown node provides them with a valid block.

Start-up and Procedure

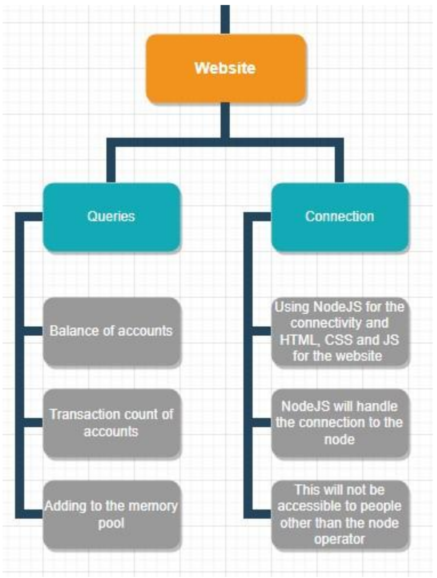

Start up
Requesting blocks
The node receives a block with a higher block number than it expects
The node calculates how many blocks it is behind
The node request from the known nodes the block numbers that it is missing until it is up to date
Communicating with other nodes
The node should communicate with another node as it starts up
If a communicating node is foreign to a node then it adds them to their list of known nodes and respondes.
Nodes send blocks and transactions between each other

These segments are a representation of the solutions that fix the overall problem of the node's start-up and procedure.

As the node comes online it should communicate with another node. This forms the purpose of gathering all of the missed blocks allowing a node to reach the correct block height and therefore be up to date. Once the node attains the current block height it will request all of the missed blocks from its most valuable node (AKA the most trusted node) and validate each of them until it has a full copy of the blockchain.

Nodes will be aware of who is connected to them and add a node to their list of known nodes if an unknown node provides them with a valid block. The list of known nodes is important because Nodes transmit blocks and transactions between each other to function and the more nodes that connect to each other will tighten the security of the network and make it easier to test.

Website


Website
Queries
Connection
Balance of accounts
Transaction count of accounts
Adding to the memory pool
Using NodeJS for the connectivity and HTML, CSS and JS for the website
NodeJS will handle the connection to the node
This will not be accessible to people other than the node operator

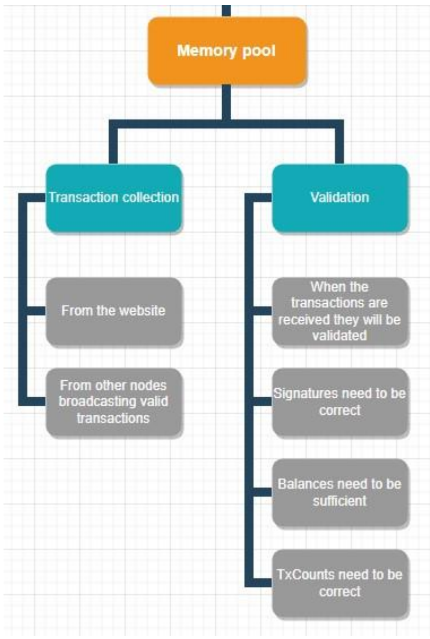
These segments are a representation of the solutions that fix the overall problem of the node's website

Connections to the node will be short as to allow the node to still operate due to accepting a connection may not allow another connection to initiate. NodeJS is my platform of choice to facilitate these connections as it is an evolution of JS which will be used in the website and is familiar to me. This website will be hosted locally and will only be available to the owner through being hosted locally by the node.

The ability of the website:

- Balance of an account ○ This allows operators to check balances locally allowing for test validation
- Transaction count of an account ○ This allow the operators to check the Transaction count of addresses allowing them to formulate transactions to test
- Adding transactions to the memory pool ○ Allows the operator to submit test transactions to be mined

Memory pool


Memory pool
Transaction collection
Validation
From the website
From other nodes broadcasting valid transactions
When the transactions are received they will be validated
Signatures need to be correct
Balances need to be sufficient
TxCounts need to be correct

These segments are a representation of the solutions that fix the overall problem of the node's Memory pool

Transactions need to be collected from the website and from other nodes. This is essential as unless a transaction is entered directly into the node through the website, the node will be unaware of the transaction and therefore will not include it in the blocks it attempts to create.

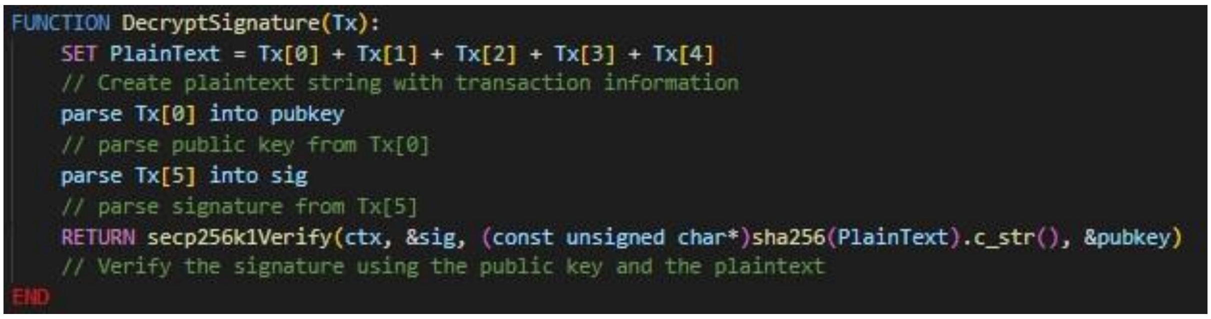
The transactions need to be validated through 4 main criteria:

- Does the sending account exist?
  - To protect against database errors for senders who don't exist - Is the signature correct?
  - Validates that the account labelled sender actually sent the transaction
- Does the sending account have a sufficient balance? ○ Validates the ability of the sender to send the funds - Is the Transaction count of the sending account correct?
  - This stops old signatures being reused by an attacking party to send a transaction twice

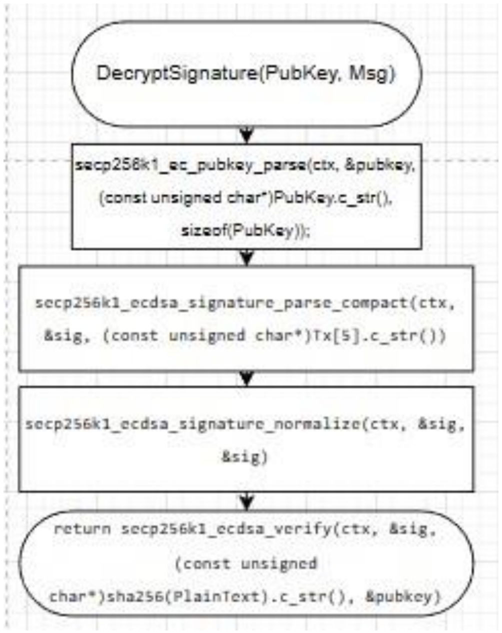
Pseudocode algorithms and flow diagrams

This section contains all of the pseudocode algorithms for my solution. The flow diagrams are included in the submission but are a collection of images named after the function they correspond to.

Decrypt Signature

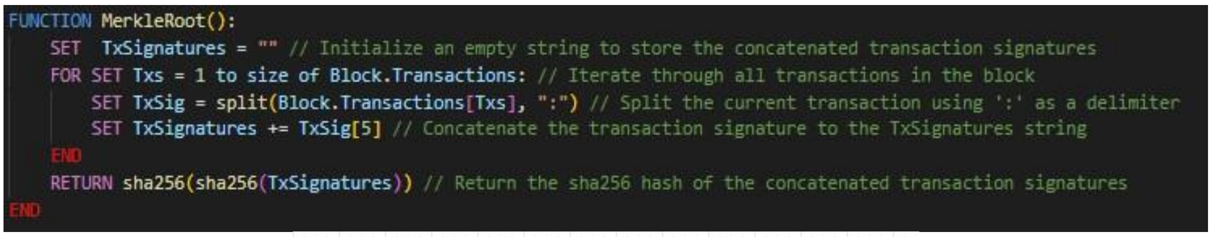


```
FUNCTION DecryptSignature(Tx):
    SET PlainText = Tx[0] + Tx[1] + Tx[2] + Tx[3] + Tx[4]
    // Create plaintext string with transaction information
    parse Tx[0] into pubkey
    // parse public key from Tx[0]
    parse Tx[5] into sig
    // parse signature from Tx[5]
    RETURN secp256k1Verify(ctx, &sig, (const unsigned char*)sha256(PlainText).c_str(), &pubkey)
    // Verify the signature using the public key and the plaintext
END
```

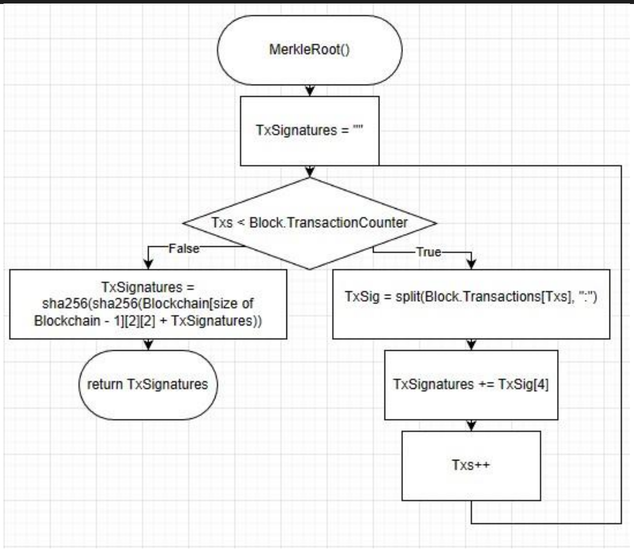

DecryptSignature(PubKey, Msg)
secp256k1_ec_pubkey_parse(ctx, &pubkey, (const unsigned char*)PubKey.c_str(), sizeof(PubKey));
secp256k1_ecdsa_signature_parse_compact(ctx, &sig, (const unsigned char*)Tx[5].c_str())
secp256k1_ecdsa_signature_normalize(ctx, &sig, &sig)
return secp256k1_ecdsa_verify(ctx, &sig, (const unsigned char*)sha256(PlainText).c_str(), &pubkey)

`DecryptSignature` is a function that takes in the transaction and verifies the signature against the plaintext of the transaction. The SECP256k1 algorithm will be used to produce and verify signatures. The last line is written in C++ formatt as I wanted to make it clear that I would not be writing the function to decrypt the signature as that is unattainable for this timeframe due to complexity.

Merkle Root

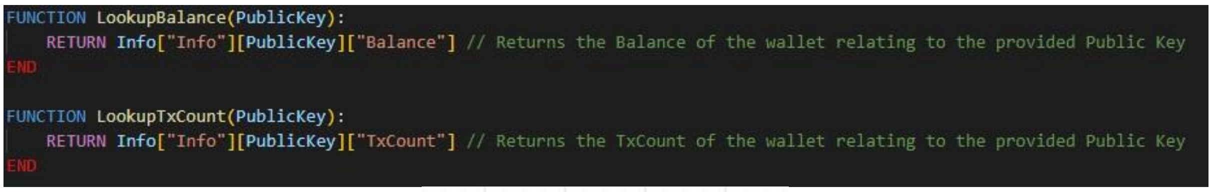


```
FUNCTION MerkleRoot():
    SET  TxSignatures = "" // Initialize an empty string to store the concatenated transaction signatures
    FOR SET Txs = 1 to size of Block.Transactions: // Iterate through all transactions in the block
        SET TxSig = split(Block.Transactions[Txs], ":") // Split the current transaction using ':' as a delimiter
        SET TxSignatures += TxSig[5] // Concatenate the transaction signature to the TxSignatures string
    END
    RETURN sha256(sha256(TxSignatures)) // Return the sha256 hash of the concatenated transaction signatures
END
```

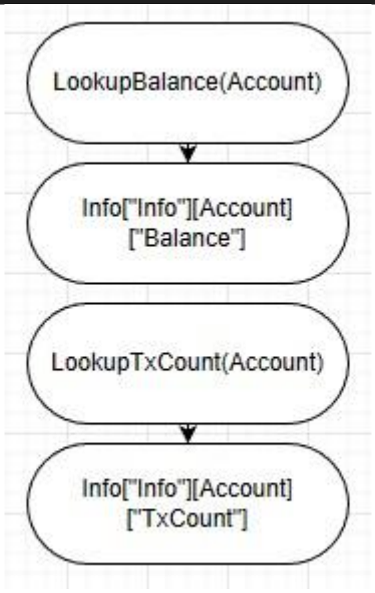

MerkleRoot()
TxSignatures = ""
Txs < Block.TransactionCounter
False
True
TxSignatures = sha256(sha256(Blockchain[size of Blockchain - 1][2][2] + TxSignatures))
TxSig = split(Block.Transactions[Txs], ":")
return TxSignatures
TxSignatures += TxSig[4]
Txs++

The `MerkleRoot` function calculates the Merkle root of the transactions in the block. It first creates a string called `TxSignatures` by concatenating the fifth element of each transaction (which is the signature), split by a colon. The function then returns the double hash of the concatenation of the previous block's Merkle root and `TxSignatures`.

Lookup Balance, Lookup TransactionCount



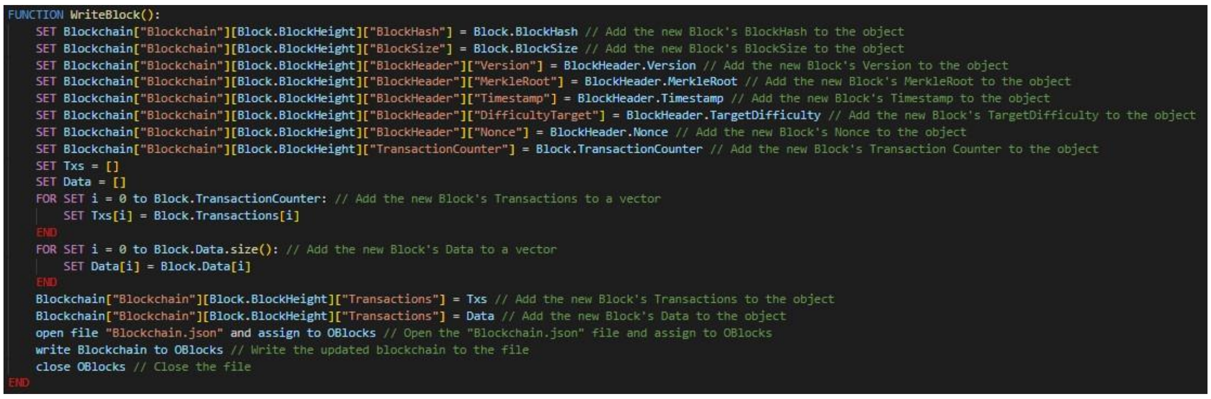
```
FUNCTION LookupBalance(PublicKey):
    RETURN Info["Info"][PublicKey]["Balance"] // Returns the Balance of the wallet relating to the provided Public Key
END

FUNCTION LookupTxCount(PublicKey):
    RETURN Info["Info"][PublicKey]["TxCount"] // Returns the TxCount of the wallet relating to the provided Public Key
END
```


LookupBalance(Account)
Info["Info"][Account]["Balance"]
LookupTxCount(Account)
Info["Info"][Account]["TxCount"]

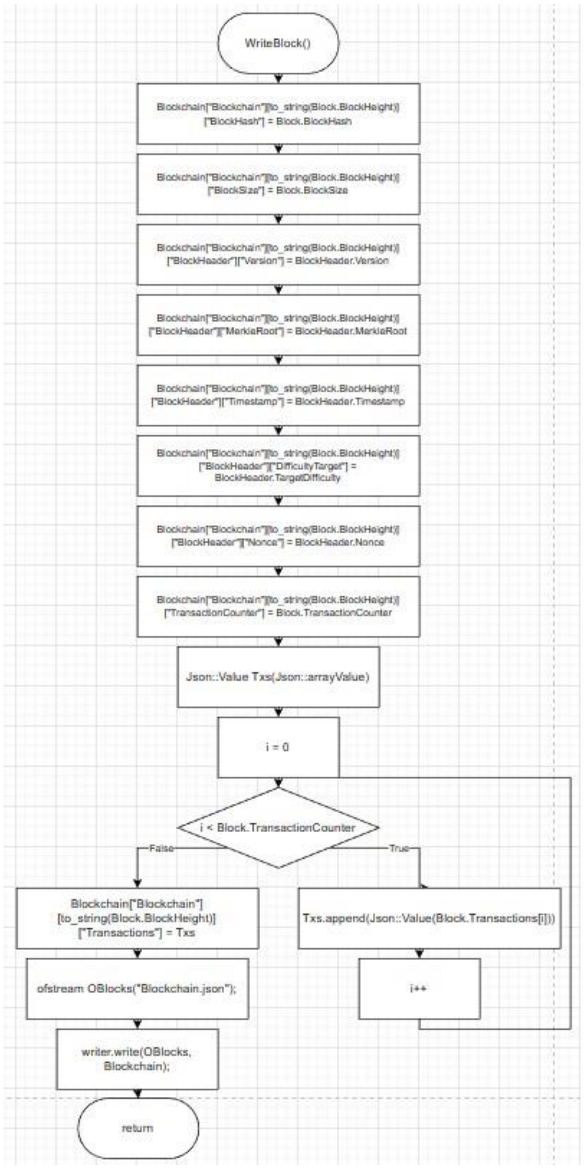
The `LookupBalance` function takes in an account and returns the balance of that account stored in the `Info` data structure.

The `LookupTransactionCount` function takes in an account and returns the number of transactions made by that account stored in the `Info` data structure.

WriteBlock

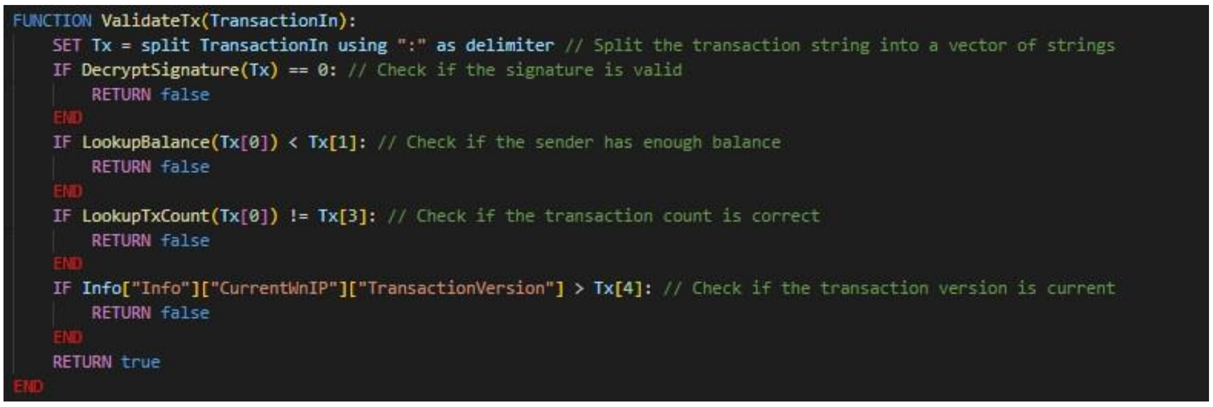


```
FUNCTION WriteBlock():
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHash"] = Block.BlockHash // Add the new Block's BlockHash to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockSize"] = Block.BlockSize // Add the new Block's BlockSize to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHeader"]["Version"] = BlockHeader.Version // Add the new Block's Version to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHeader"]["MerkleRoot"] = BlockHeader.MerkleRoot // Add the new Block's MerkleRoot to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHeader"]["Timestamp"] = BlockHeader.Timestamp // Add the new Block's Timestamp to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHeader"]["DifficultyTarget"] = BlockHeader.TargetDifficulty // Add the new Block's TargetDifficulty to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["BlockHeader"]["Nonce"] = BlockHeader.Nonce // Add the new Block's Nonce to the object
    SET Blockchain["Blockchain"][Block.BlockHeight]["TransactionCounter"] = Block.TransactionCounter // Add the new Block's Transaction Counter to the object
    SET Txs = []
    SET Data = []
    FOR SET i = 0 to Block.TransactionCounter: // Add the new Block's Transactions to a vector
        SET Txs[i] = Block.Transactions[i]
    END
    FOR SET i = 0 to Block.Data.size(): // Add the new Block's Data to a vector
        SET Data[i] = Block.Data[i]
    END
    Blockchain["Blockchain"][Block.BlockHeight]["Transactions"] = Txs // Add the new Block's Transactions to the object
    Blockchain["Blockchain"][Block.BlockHeight]["Transactions"] = Data // Add the new Block's Data to the object
    open file "Blockchain.json" and assign to OBlocks // Open the "Blockchain.json" file and assign to OBlocks
    write Blockchain to OBlocks // Write the updated blockchain to the file
    close OBlocks // Close the file
END
```

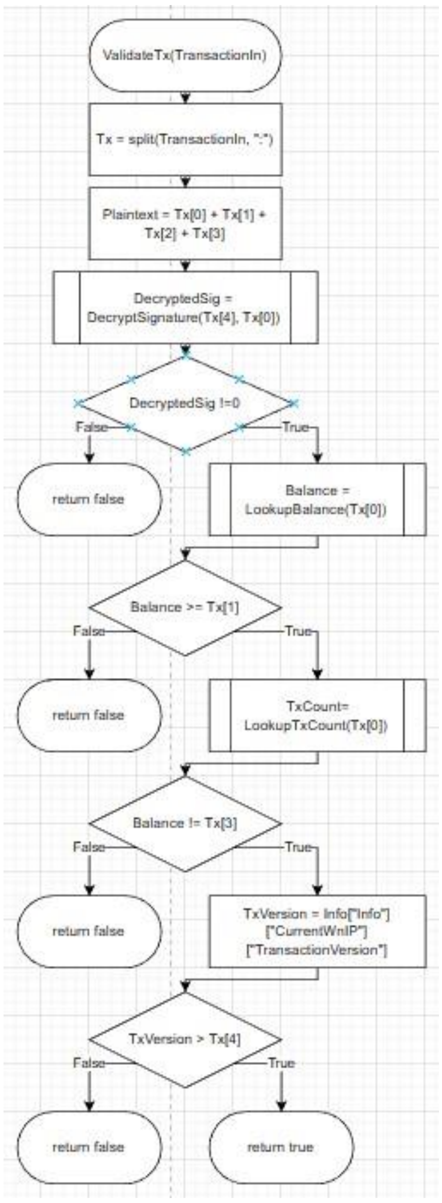

WriteBlock()
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHash"] = Block.BlockHash
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockSize"] = Block.BlockSize
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Version"] = BlockHeader.Version
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["MerkleRoot"] = BlockHeader.MerkleRoot
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Timestamp"] = BlockHeader.Timestamp
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["DifficultyTarget"] = BlockHeader.TargetDifficulty
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Nonce"] = BlockHeader.Nonce
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["TransactionCounter"] = Block.TransactionCounter
Json::Value Txs(Json::arrayValue)
i = 0
i < Block.TransactionCounter
False
True
Blockchain["Blockchain"][to_string(Block.BlockHeight)]["Transactions"] = Txs
Txs.append(Json::Value(Block.Transactions[i]))
ofstream OBlocks("Blockchain.json");
i++
writer.write(OBlocks, Blockchain);
return

The `WriteBlock` function writes the current block to the `Blockchain` data structure. It calls on two structures called `Block` and `BlockHeader`, both of which hold different parts of the block. It then stores the various fields of the block and block header in the appropriate locations in the object `Blockchain`. It also stores the transactions and data of the block in `Blockchain`. It then saves this to an external file.

Validate Transaction

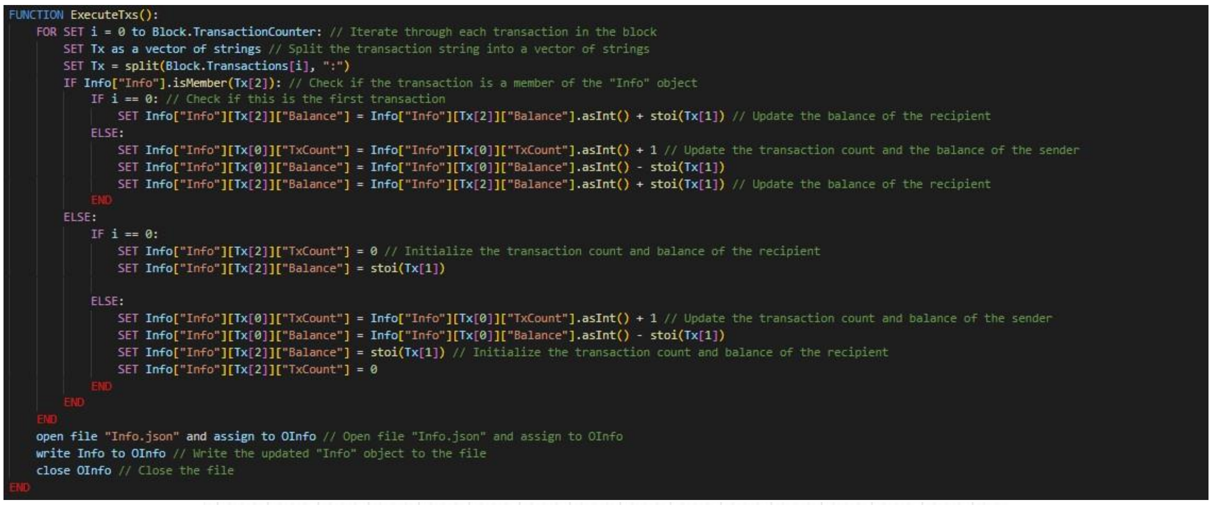


```
FUNCTION ValidateTx(TransactionIn):
    SET Tx = split TransactionIn using ":" as delimiter // Split the transaction string into a vector of strings
    IF DecryptSignature(Tx) == 0: // Check if the signature is valid
        RETURN false
    END
    IF LookupBalance(Tx[0]) < Tx[1]: // Check if the sender has enough balance
        RETURN false
    END
    IF LookupTxCount(Tx[0]) != Tx[3]: // Check if the transaction count is correct
        RETURN false
    END
    IF Info["Info"]["CurrentWnIP"]["TransactionVersion"] > Tx[4]: // Check if the transaction version is current
        RETURN false
    END
    RETURN true
END
```

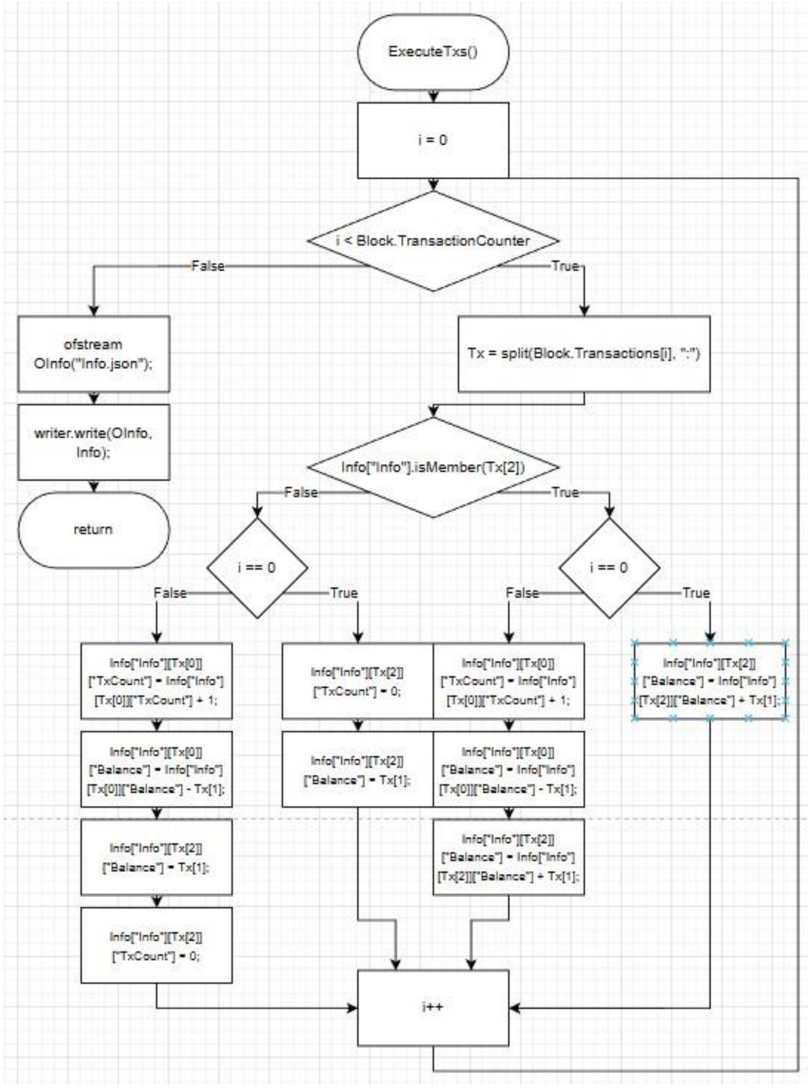

ValidateTx(TransactionIn)
Tx = split(TransactionIn, ":")
Plaintext = Tx[0] + Tx[1] + Tx[2] + Tx[3]
DecryptedSig = DecryptSignature(Tx[4], Tx[0])
DecryptedSig !=0
False
True
return false
Balance = LookupBalance(Tx[0])
Balance >= Tx[1]
False
True
return false
TxCount= LookupTxCount(Tx[0])
Balance != Tx[3]
False
True
return false
TxVersion = Info["Info"]["CurrentWnIP"]["TransactionVersion"]
TxVersion > Tx[4]
False
True
return false
return true

The `ValidateTx` function takes in a transaction and returns a Boolean indicating whether the transaction is valid or not. It does this by first splitting the transaction into its various fields and then checking if the signature is valid, if the sender has sufficient funds, and if the transaction count is correct.

Execute Transactions

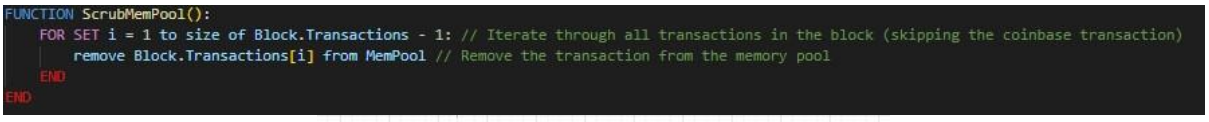


```
FUNCTION ExecuteTxs():
    FOR SET i = 0 to Block.TransactionCounter: // Iterate through each transaction in the block
        SET Tx as a vector of strings // Split the transaction string into a vector of strings
        SET Tx = split(Block.Transactions[i], ":")
        IF Info["Info"].isMember(Tx[2]): // Check if the transaction is a member of the "Info" object
            IF i == 0: // Check if this is the first transaction
                SET Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]) // Update the balance of the recipient
            ELSE:
                SET Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1 // Update the transaction count and the balance of the sender
                SET Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1])
                SET Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]) // Update the balance of the recipient
            END
        ELSE:
            IF i == 0:
                SET Info["Info"][Tx[2]]["TxCount"] = 0 // Initialize the transaction count and balance of the recipient
                SET Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1])

            ELSE:
                SET Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1 // Update the transaction count and balance of the sender
                SET Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1])
                SET Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1]) // Initialize the transaction count and balance of the recipient
                SET Info["Info"][Tx[2]]["TxCount"] = 0
            END
        END
    END
    open file "Info.json" and assign to OInfo // Open file "Info.json" and assign to OInfo
    write Info to OInfo // Write the updated "Info" object to the file
    close OInfo // Close the file
END
```

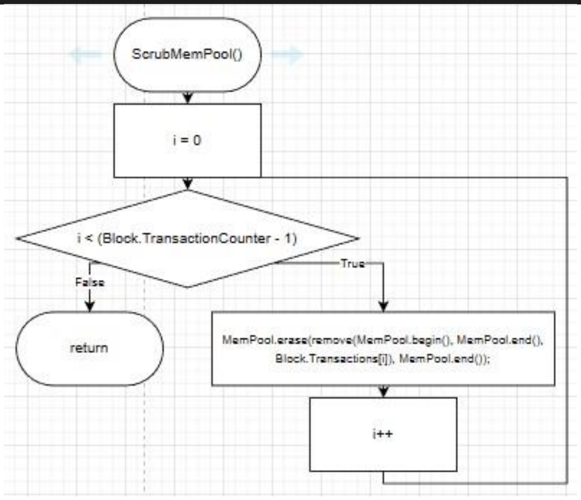

ExecuteTxs()
i = 0
i < Block.TransactionCounter
False
True
ofstream OInfo("Info.json");
writer.write(OInfo, Info);
return
Tx = split(Block.Transactions[i], ":")
Info["Info"].isMember(Tx[2])
False
True
i == 0
i == 0
False
True
False
True
Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"] + 1;
Info["Info"][Tx[2]]["TxCount"] = 0;
Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"] + 1;
Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"] + Tx[1];
Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"] - Tx[1];
Info["Info"][Tx[2]]["Balance"] = Tx[1];
Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"] - Tx[1];
Info["Info"][Tx[2]]["Balance"] = Tx[1];
Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"] + Tx[1];
Info["Info"][Tx[2]]["TxCount"] = 0;
i++

The `ExecuteTx` function executes the transactions in the current block. For the first transaction, it simply increments the balance of the recipient as the first transaction is the Coinbase (Tokens are created). For all other transactions, it decrements the balance of the sender and increments the balance of the recipient, and also increments the transaction count of the sender. If the recipient account has not been seen by the node it creates an account in the object `Info`.

Scrub Memory Pool

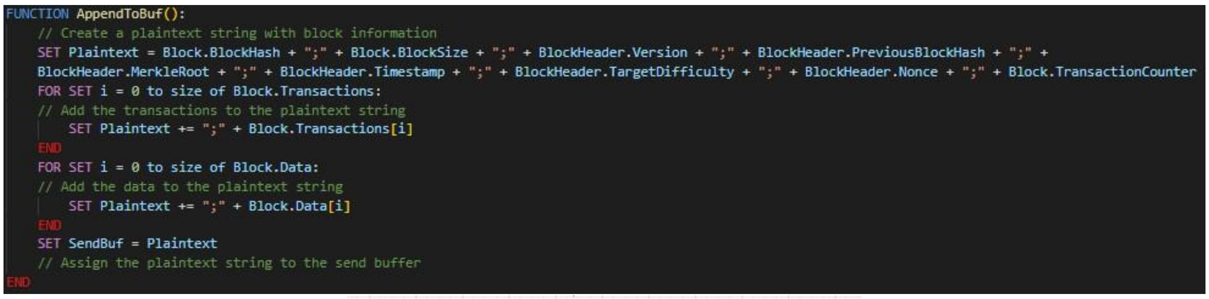


```
FUNCTION ScrubMemPool():
    FOR SET i = 1 to size of Block.Transactions - 1: // Iterate through all transactions in the block (skipping the coinbase transaction)
        remove Block.Transactions[i] from MemPool // Remove the transaction from the memory pool
    END
END
```

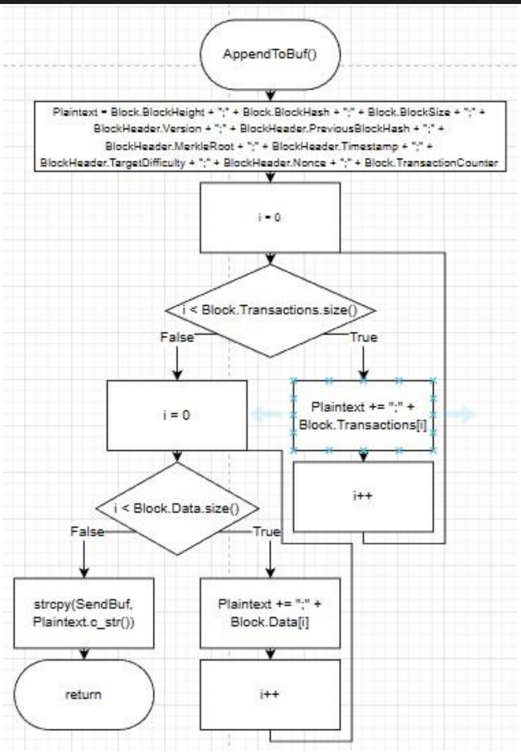

ScrubMemPool()
i = 0
i < (Block.TransactionCounter - 1)
True
False
return
MemPool.erase(remove(MemPool.begin(), MemPool.end(), Block.Transactions[i]), MemPool.end());
i++

The `ScrubMemPool` function removes the transactions from the Mempool that were included in the latest block.

Append to Buffer

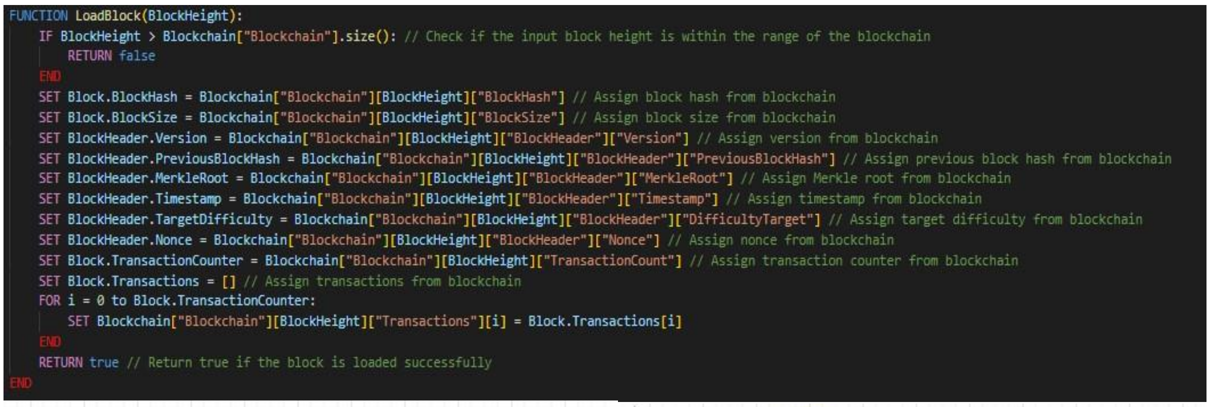


```
FUNCTION AppendToBuf():
    // Create a plaintext string with block information
    SET Plaintext = Block.BlockHash + ";" + Block.BlockSize + ";" + BlockHeader.Version + ";" + BlockHeader.PreviousBlockHash + ";" +
    BlockHeader.MerkleRoot + ";" + BlockHeader.Timestamp + ";" + BlockHeader.TargetDifficulty + ";" + BlockHeader.Nonce + ";" + Block.TransactionCounter
    FOR SET i = 0 to size of Block.Transactions:
    // Add the transactions to the plaintext string
        SET Plaintext += ";" + Block.Transactions[i]
    END
    FOR SET i = 0 to size of Block.Data:
    // Add the data to the plaintext string
        SET Plaintext += ";" + Block.Data[i]
    END
    SET SendBuf = Plaintext
    // Assign the plaintext string to the send buffer
END
```

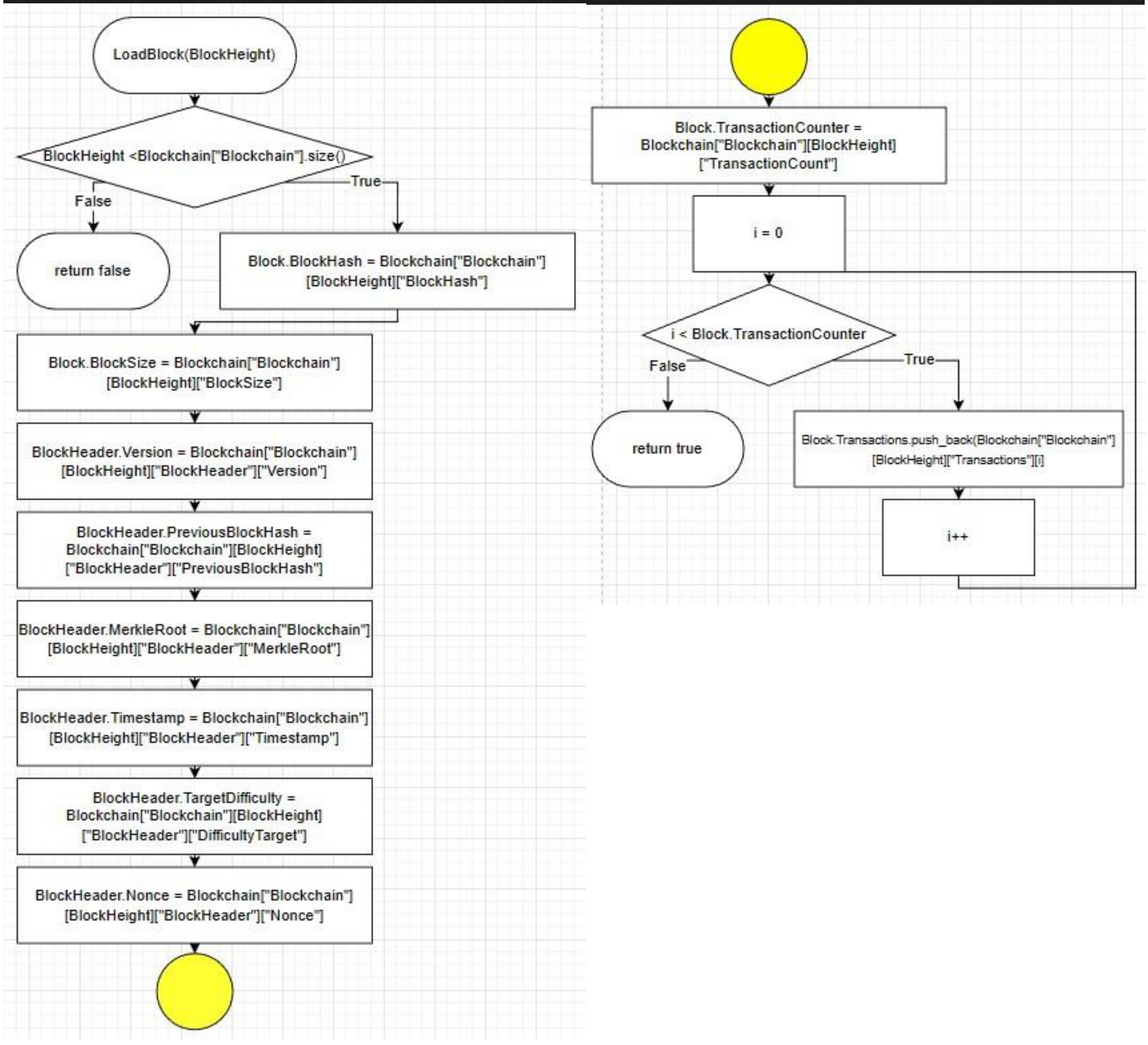

AppendToBuf()
Plaintext = Block.BlockHeight + ";" + Block.BlockHash + ";" + Block.BlockSize + ";" + BlockHeader.Version + ";" + BlockHeader.PreviousBlockHash + ";" + BlockHeader.MerkleRoot + ";" + BlockHeader.Timestamp + ";" + BlockHeader.TargetDifficulty + ";" + BlockHeader.Nonce + ";" + Block.TransactionCounter
i = 0
i < Block.Transactions.size()
False
True
i = 0
Plaintext += ";" + Block.Transactions[i]
i++
i < Block.Data.size()
False
True
strcpy(SendBuf, Plaintext.c_str())
Plaintext += ";" + Block.Data[i]
return
i++

The `AppendToBuf` function adds the contents of the structure `Block` into a single list of characters with a separator in between fields ready for transmission to another node. It then loads it into a char array that will be transmitted to the other node.

Load Block

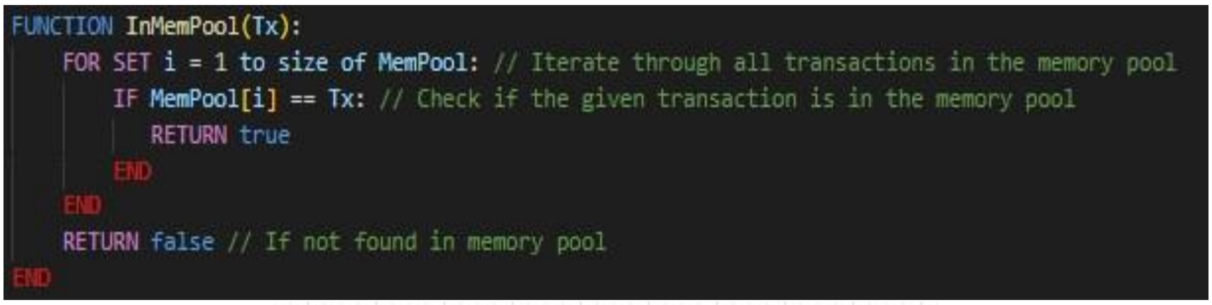


```
FUNCTION LoadBlock(BlockHeight):
    IF BlockHeight > Blockchain["Blockchain"].size(): // Check if the input block height is within the range of the blockchain
        RETURN false
    END
    SET Block.BlockHash = Blockchain["Blockchain"][BlockHeight]["BlockHash"] // Assign block hash from blockchain
    SET Block.BlockSize = Blockchain["Blockchain"][BlockHeight]["BlockSize"] // Assign block size from blockchain
    SET BlockHeader.Version = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Version"] // Assign version from blockchain
    SET BlockHeader.PreviousBlockHash = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["PreviousBlockHash"] // Assign previous block hash from blockchain
    SET BlockHeader.MerkleRoot = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["MerkleRoot"] // Assign Merkle root from blockchain
    SET BlockHeader.Timestamp = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Timestamp"] // Assign timestamp from blockchain
    SET BlockHeader.TargetDifficulty = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["DifficultyTarget"] // Assign target difficulty from blockchain
    SET BlockHeader.Nonce = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Nonce"] // Assign nonce from blockchain
    SET Block.TransactionCounter = Blockchain["Blockchain"][BlockHeight]["TransactionCount"] // Assign transaction counter from blockchain
    SET Block.Transactions = [] // Assign transactions from blockchain
    FOR i = 0 to Block.TransactionCounter:
        SET Blockchain["Blockchain"][BlockHeight]["Transactions"][i] = Block.Transactions[i]
    END
    RETURN true // Return true if the block is loaded successfully
END
```

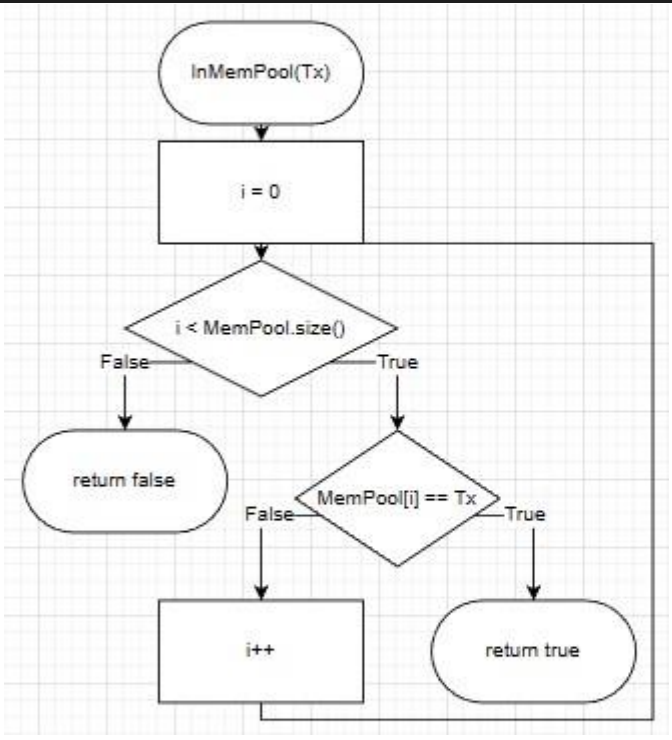

LoadBlock(BlockHeight)
BlockHeight <Blockchain["Blockchain"].size()
True
False
return false
Block.BlockHash = Blockchain["Blockchain"] [BlockHeight]["BlockHash"]
Block.BlockSize = Blockchain["Blockchain"] [BlockHeight]["BlockSize"]
BlockHeader.Version = Blockchain["Blockchain"] [BlockHeight]["BlockHeader"]["Version"]
BlockHeader.PreviousBlockHash = Blockchain["Blockchain"][BlockHeight] ["BlockHeader"]["PreviousBlockHash"]
BlockHeader.MerkleRoot = Blockchain["Blockchain"] [BlockHeight]["BlockHeader"]["MerkleRoot"]
BlockHeader.Timestamp = Blockchain["Blockchain"] [BlockHeight]["BlockHeader"]["Timestamp"]
BlockHeader.TargetDifficulty = Blockchain["Blockchain"][BlockHeight] ["BlockHeader"]["DifficultyTarget"]
BlockHeader.Nonce = Blockchain["Blockchain"] [BlockHeight]["BlockHeader"]["Nonce"]
Block.TransactionCounter = Blockchain["Blockchain"][BlockHeight] ["TransactionCount"]
i = 0
i < Block.TransactionCounter
True
False
return true
Block.Transactions.push_back(Blockchain["Blockchain"] [BlockHeight]["Transactions"][i])
i++

The `LoadBlock` function populates the structure `Block` with the contents from the JSON object that relate to the blockheight passed as a parameter. This is to be sent to other nodes to catch them back up if they have been inactive and missed blocks.

In the memory pool

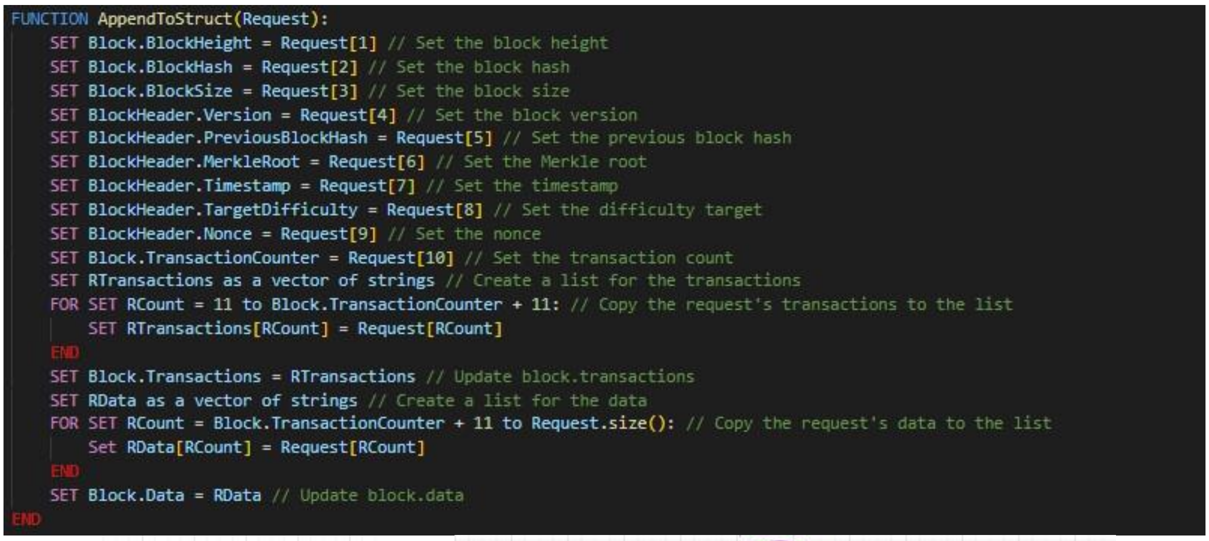


```
FUNCTION InMemPool(Tx):
    FOR SET i = 1 to size of MemPool: // Iterate through all transactions in the memory pool
        IF MemPool[i] == Tx: // Check if the given transaction is in the memory pool
            RETURN true
        END
    END
    RETURN false // If not found in memory pool
END
```

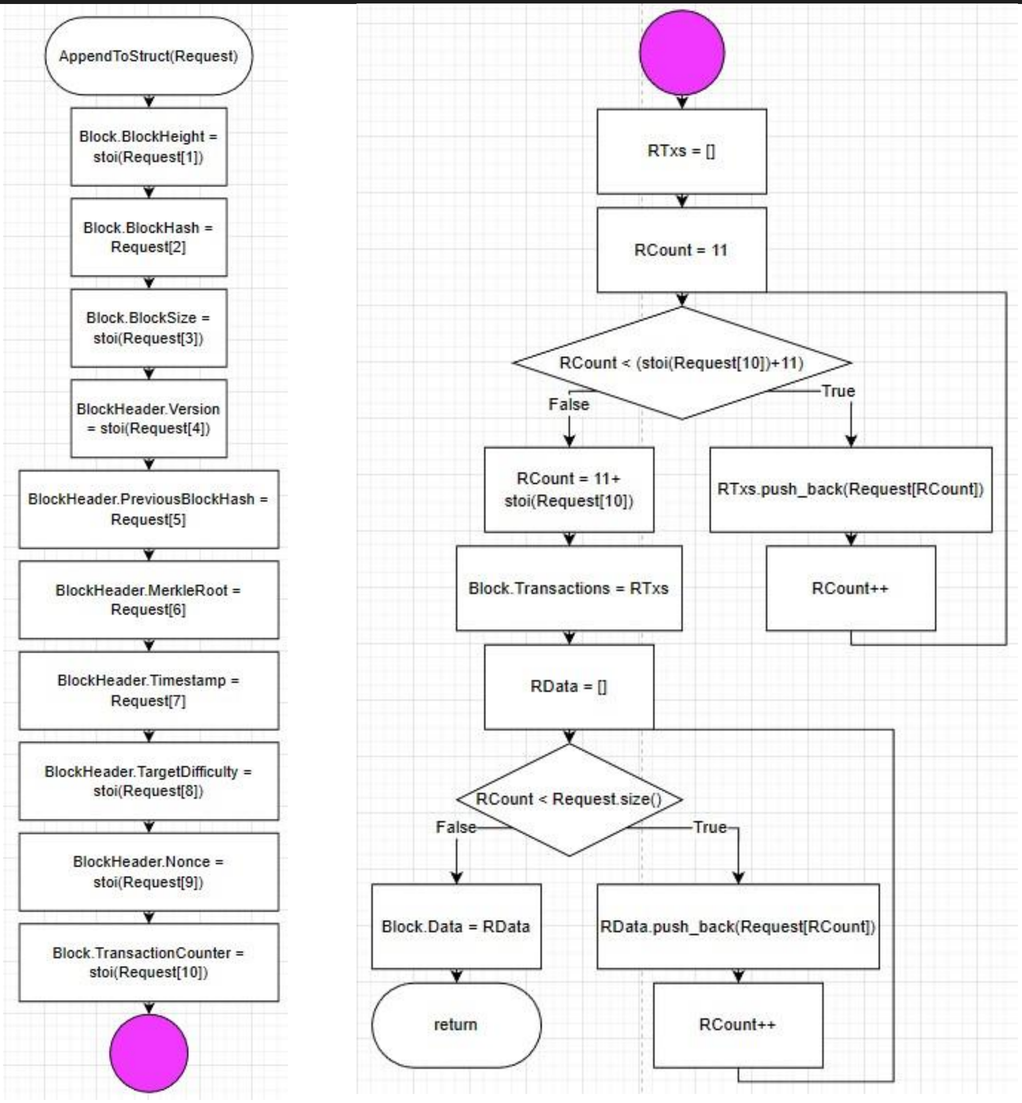


The `InMemPool` function is used to check if a received transaction is present in the memory pool to check if it needs to be added to the memory pool. It is used to stop duplicate transactions from clogging up the memory pool.

Append to Structure

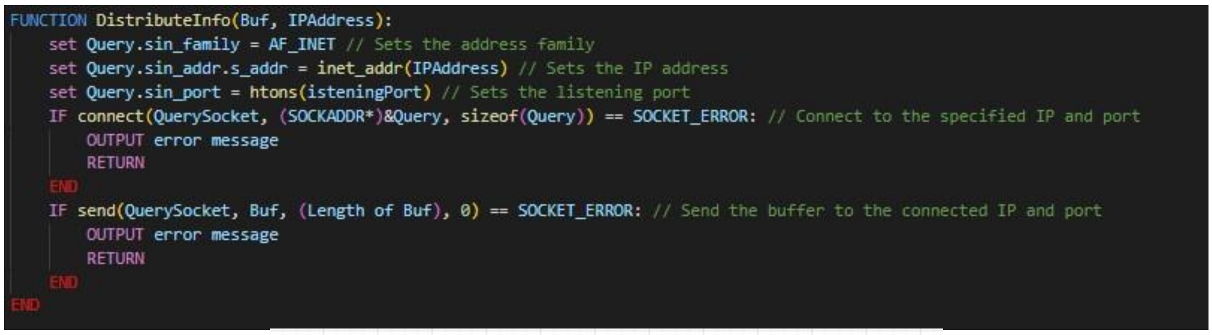


```
FUNCTION AppendToStruct(Request):
    SET Block.BlockHeight = Request[1] // Set the block height
    SET Block.BlockHash = Request[2] // Set the block hash
    SET Block.BlockSize = Request[3] // Set the block size
    SET BlockHeader.Version = Request[4] // Set the block version
    SET BlockHeader.PreviousBlockHash = Request[5] // Set the previous block hash
    SET BlockHeader.MerkleRoot = Request[6] // Set the Merkle root
    SET BlockHeader.Timestamp = Request[7] // Set the timestamp
    SET BlockHeader.TargetDifficulty = Request[8] // Set the difficulty target
    SET BlockHeader.Nonce = Request[9] // Set the nonce
    SET Block.TransactionCounter = Request[10] // Set the transaction count
    SET RTransactions as a vector of strings // Create a list for the transactions
    FOR SET RCount = 11 to Block.TransactionCounter + 11: // Copy the request's transactions to the list
        SET RTransactions[RCount] = Request[RCount]
    END
    SET Block.Transactions = RTransactions // Update block.transactions
    SET RData as a vector of strings // Create a list for the data
    FOR SET RCount = Block.TransactionCounter + 11 to Request.size(): // Copy the request's data to the list
        Set RData[RCount] = Request[RCount]
    END
    SET Block.Data = RData // Update block.data
END
```

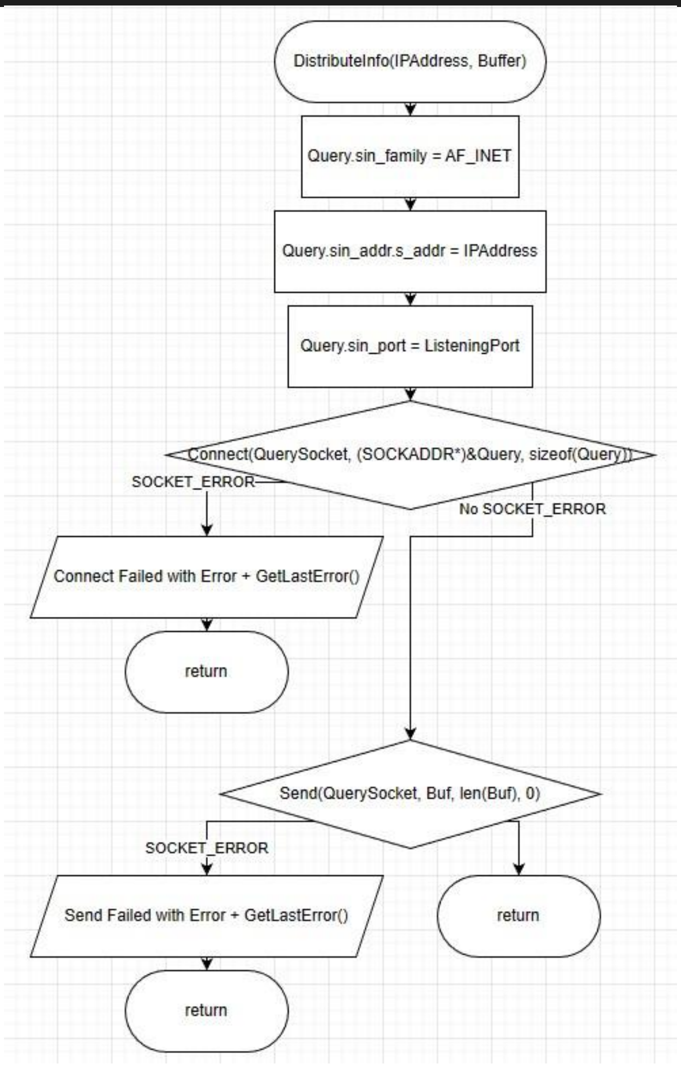

AppendToStruct(Request)
Block.BlockHeight = stoi(Request[1])
Block.BlockHash = Request[2]
Block.BlockSize = stoi(Request[3])
BlockHeader.Version = stoi(Request[4])
BlockHeader.PreviousBlockHash = Request[5]
BlockHeader.MerkleRoot = Request[6]
BlockHeader.Timestamp = Request[7]
BlockHeader.TargetDifficulty = stoi(Request[8])
BlockHeader.Nonce = stoi(Request[9])
Block.TransactionCounter = stoi(Request[10])
RTxs = []
RCount = 11
RCount < (stoi(Request[10])+11)
False
True
RCount = 11+ stoi(Request[10])
RTxs.push_back(Request[RCount])
Block.Transactions = RTxs
RCount++
RData = []
RCount < Request.size()
False
True
Block.Data = RData
RData.push_back(Request[RCount])
return
RCount++

The `AppendToStruct` function adds the contents of the vector: request into the structure `Block`. This is used to interpret incoming blocks ready for validation.

Distribute Info

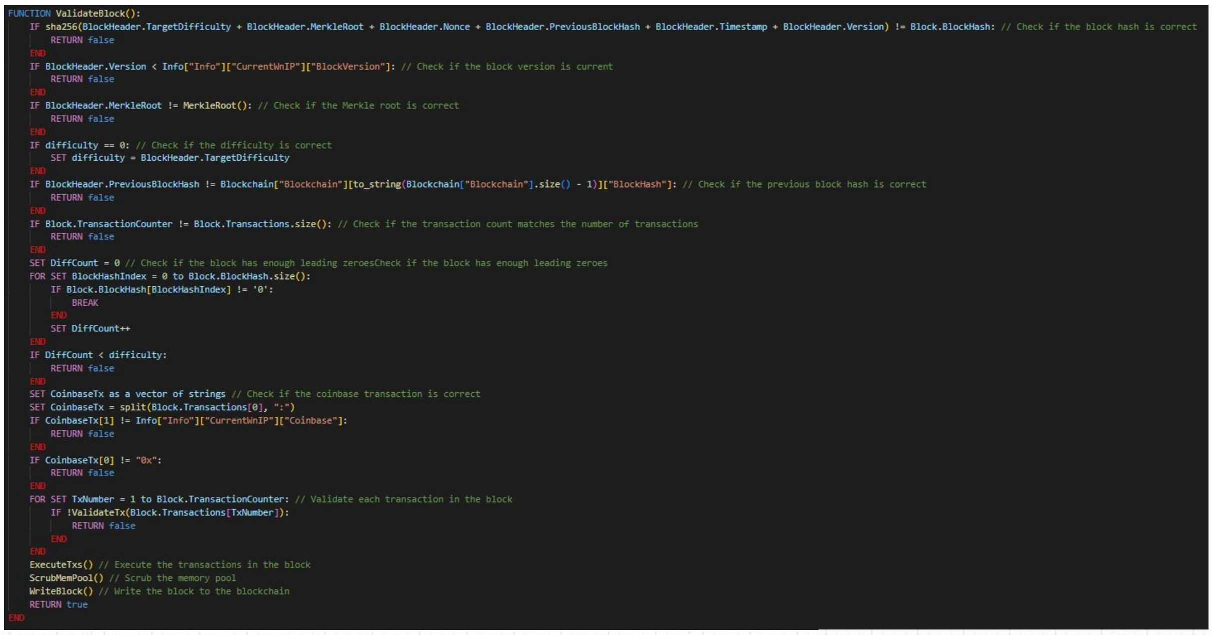


```
FUNCTION DistributeInfo(Buf, IPAddress):
    set Query.sin_family = AF_INET // Sets the address family
    set Query.sin_addr.s_addr = inet_addr(IPAddress) // Sets the IP address
    set Query.sin_port = htons(isteningPort) // Sets the listening port
    IF connect(QuerySocket, (SOCKADDR*)&Query, sizeof(Query)) == SOCKET_ERROR: // Connect to the specified IP and port
        OUTPUT error message
        RETURN
    END
    IF send(QuerySocket, Buf, (Length of Buf), 0) == SOCKET_ERROR: // Send the buffer to the connected IP and port
        OUTPUT error message
        RETURN
    END
END
```

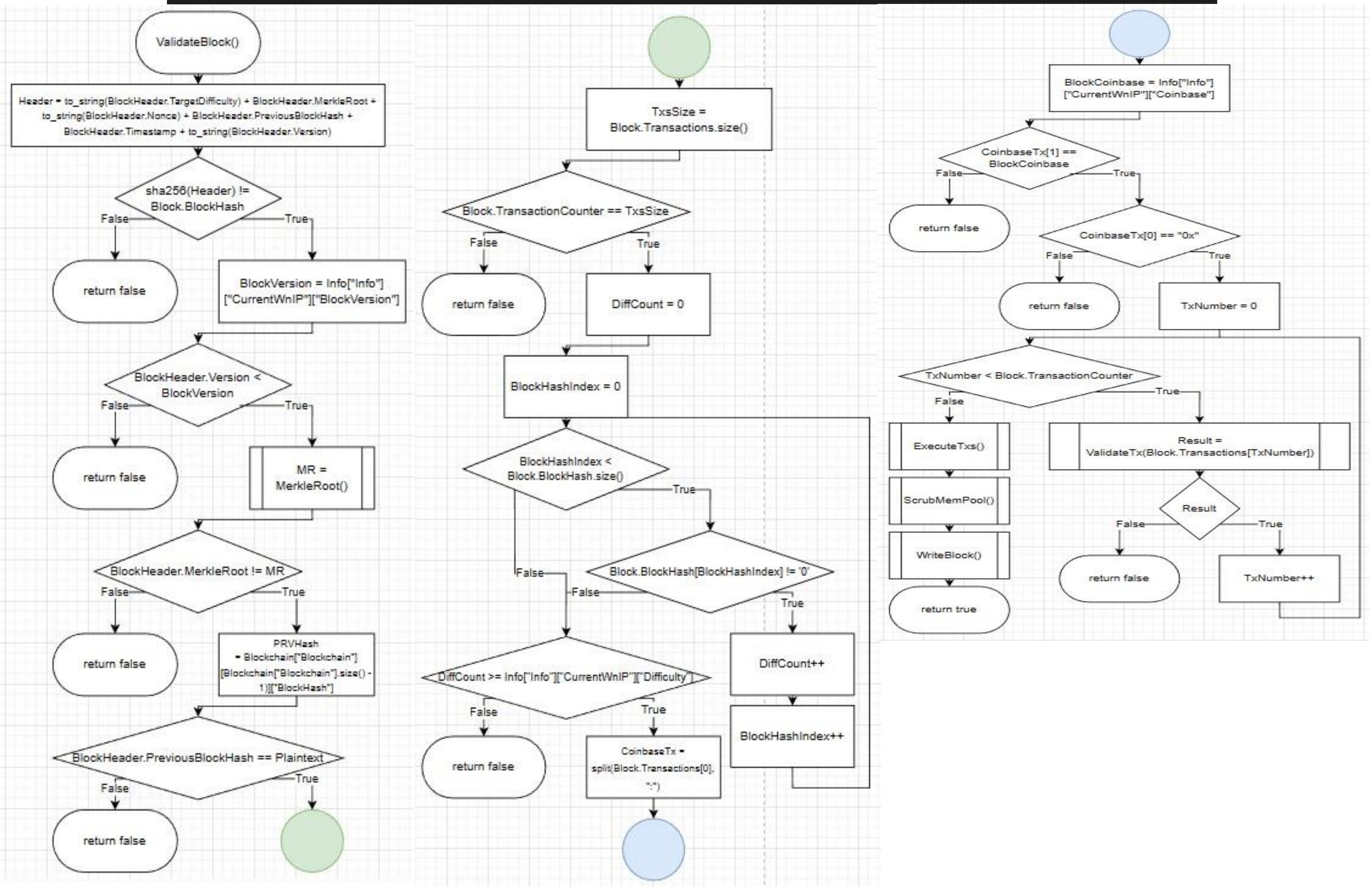

DistributeInfo(IPAddress, Buffer)
Query.sin_family = AF_INET
Query.sin_addr.s_addr = IPAddress
Query.sin_port = ListeningPort
Connect(QuerySocket, (SOCKADDR*)&Query, sizeof(Query))
SOCKET_ERROR
No SOCKET_ERROR
Connect Failed with Error + GetLastError()
return
Send(QuerySocket, Buf, len(Buf), 0)
SOCKET_ERROR
Send Failed with Error + GetLastError()
return
return

The `DistributeInfo` function is used to send data to other nodes such as transactions and blocks. It takes an IP address and a buffer containing the data to send and uses winsock2 commands to establish a connection to the other node and send the data.

Validate Block



```
FUNCTION ValidateBlock():
    IF sha256(BlockHeader.TargetDifficulty + BlockHeader.MerkleRoot + BlockHeader.Nonce + BlockHeader.PreviousBlockHash + BlockHeader.Timestamp + BlockHeader.Version) != Block.BlockHash: // Check if the block hash is correct
        RETURN false
    END
    IF BlockHeader.Version < Info["Info"]["CurrentWnIP"]["BlockVersion"]: // Check if the block version is current
        RETURN false
    END
    IF BlockHeader.MerkleRoot != MerkleRoot(): // Check if the Merkle root is correct
        RETURN false
    END
    IF difficulty == 0: // Check if the difficulty is correct
        SET difficulty = BlockHeader.TargetDifficulty
    END
    IF BlockHeader.PreviousBlockHash != Blockchain["Blockchain"][to_string(Blockchain["Blockchain"].size() - 1)]["BlockHash"]: // Check if the previous block hash is correct
        RETURN false
    END
    IF Block.TransactionCounter != Block.Transactions.size(): // Check if the transaction count matches the number of transactions
        RETURN false
    END
    SET DiffCount = 0 // Check if the block has enough leading zeroesCheck if the block has enough leading zeroes
    FOR SET BlockHashIndex = 0 to Block.BlockHash.size():
        IF Block.BlockHash[BlockHashIndex] != '0':
            BREAK
        END
        SET DiffCount++
    END
    IF DiffCount < difficulty:
        RETURN false
    END
    SET CoinbaseTx as a vector of strings // Check if the coinbase transaction is correct
    SET CoinbaseTx = split(Block.Transactions[0], ":")
    IF CoinbaseTx[1] != Info["Info"]["CurrentWnIP"]["Coinbase"]:
        RETURN false
    END
    IF CoinbaseTx[0] != "0x":
        RETURN false
    END
    FOR SET TxNumber = 1 to Block.TransactionCounter: // Validate each transaction in the block
        IF !ValidateTx(Block.Transactions[TxNumber]):
            RETURN false
        END
    END
    ExecuteTxs() // Execute the transactions in the block
    ScrubMemPool() // Scrub the memory pool
    WriteBlock() // Write the block to the blockchain
    RETURN true
END
```


ValidateBlock()
Header = to_string(BlockHeader.TargetDifficulty) + BlockHeader.MerkleRoot + to_string(BlockHeader.Nonce) + BlockHeader.PreviousBlockHash + BlockHeader.Timestamp + to_string(BlockHeader.Version)
sha256(Header) != Block.BlockHash
False
True
return false
BlockVersion = Info["Info"]["CurrentWnIP"]["BlockVersion"]
BlockHeader.Version < BlockVersion
False
True
return false
MR = MerkleRoot()
BlockHeader.MerkleRoot != MR
False
True
return false
PRVHash = Blockchain["Blockchain"][Blockchain["Blockchain"].size() - 1)]["BlockHash"]
BlockHeader.PreviousBlockHash == Plaintext
False
True
return false
TxsSize = Block.Transactions.size()
Block.TransactionCounter == TxsSize
False
True
return false
DiffCount = 0
BlockHashIndex = 0
BlockHashIndex < Block.BlockHash.size()
True
False
Block.BlockHash[BlockHashIndex] != '0'
False
True
DiffCount++
BlockHashIndex++
DiffCount >= Info["Info"]["CurrentWnIP"]["Difficulty"]
False
True
return false
CoinbaseTx = split(Block.Transactions[0], ":")
BlockCoinbase = Info["Info"]["CurrentWnIP"]["Coinbase"]
CoinbaseTx[1] == BlockCoinbase
False
True
return false
CoinbaseTx[0] == "0x"
False
True
return false
TxNumber = 0
TxNumber < Block.TransactionCounter
False
True
ExecuteTxs()
ScrubMemPool()
WriteBlock()
return true
Result = ValidateTx(Block.Transactions[TxNumber])
Result
False
True
return false
TxNumber++

The **`ValidateBlock`** function validates blocks that were received from other nodes. It checks all of the parameters in the **`Block`** and **`BlockHeader`** structures and then passes on to functions like `ValidateTransactions`, `ScrubMemPool` and `ExecuteTransactions`.

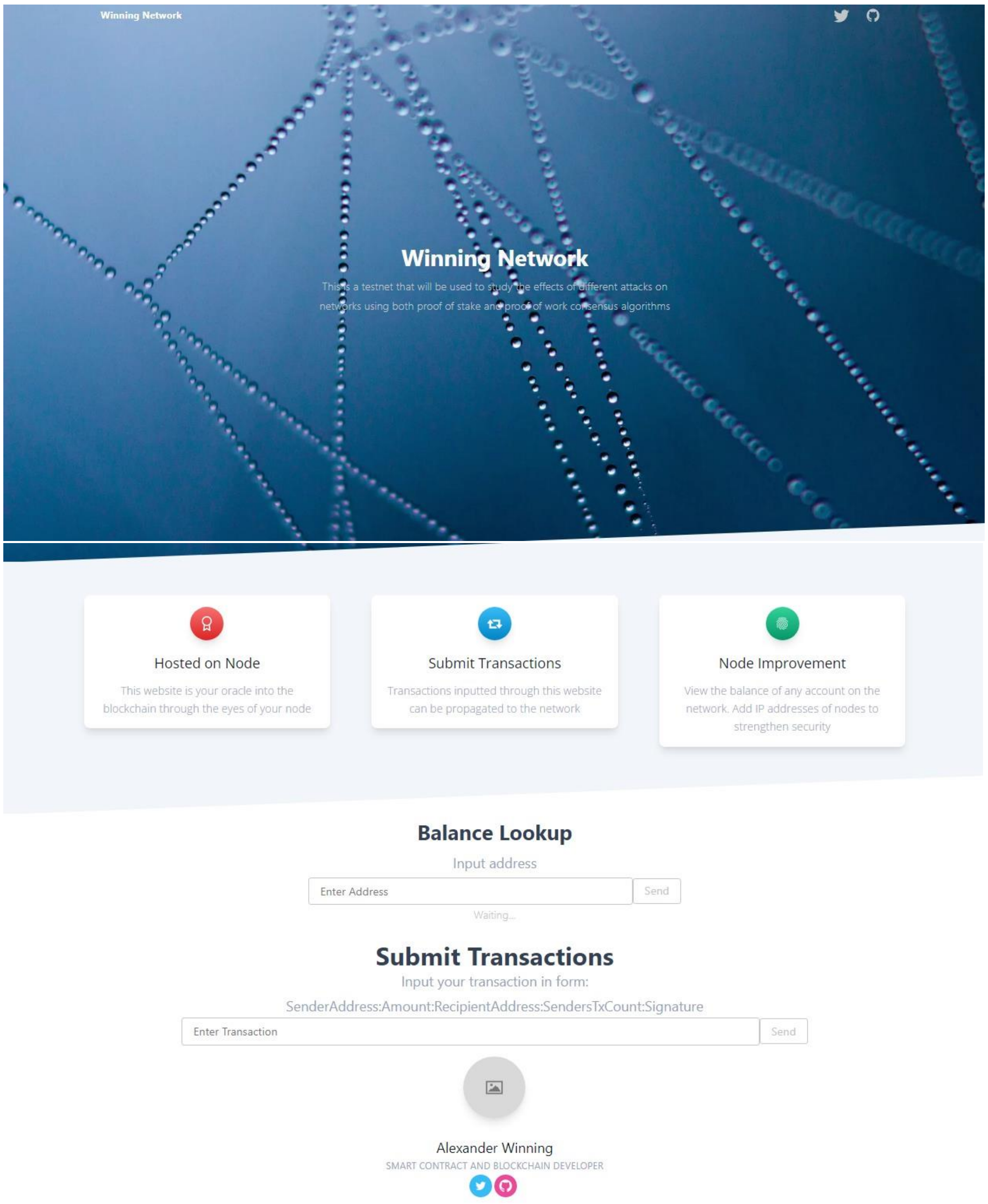
Usability

The usability features I am including in my solution are thorough documentation and Commented code. The documentation will cover the code as a whole and also all of the libraries that I am including in it. In its explanation of the code will be flow diagrams of the code as this documentation will allow any developer to pick up and use my code with minimal downtime. The choice to include documentation fits with my solution because my product is created for blockchain developers who are used to reading such documentation, though a more thorough documentation that will go more into the inner functioning of blockchain will be included to help more general developers with less knowledge of blockchain. Both sets of documentation are

attached.

The website will be the main interface with the node and will be used to input transactions into the node and poll other data it will have a GUI of which the structure will be decided through development as all of the interface relies on the capabilities and the backend.

Website Frontend



Winning Network

Winning Network

This is a testnet that will be used to study the effects of different attacks on networks using both proof of stake and proof of work consensus algorithms

Hosted on Node

This website is your oracle into the blockchain through the eyes of your node

Submit Transactions

Transactions inputted through this website can be propagated to the network

Node Improvement

View the balance of any account on the network. Add IP addresses of nodes to strengthen security

Balance Lookup

Input address

Enter Address | Send

Waiting...

Submit Transactions

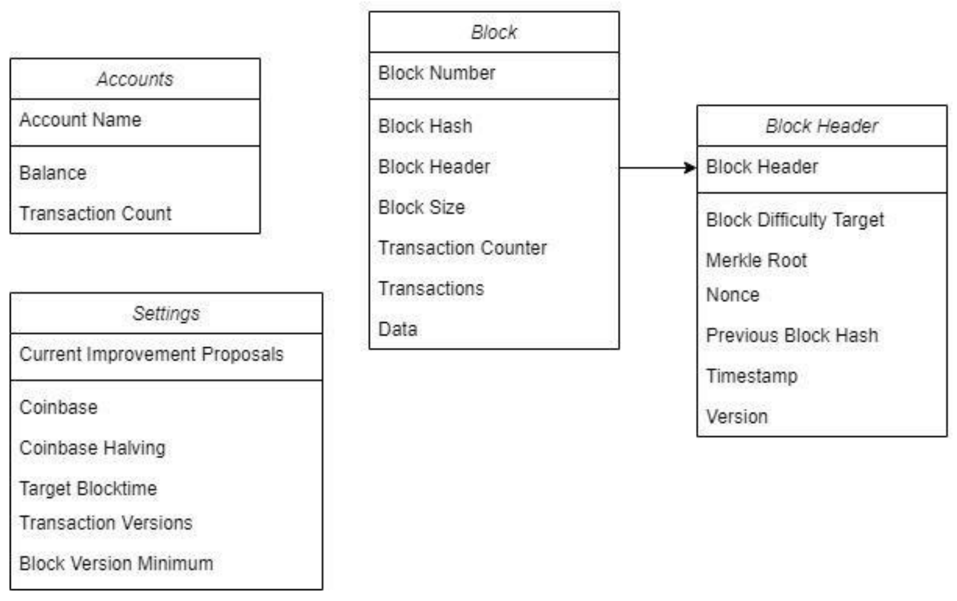
Input your transaction in form:

SenderAddress:Amount:RecipientAddress:SendersTxCount:Signature

Enter Transaction | Send

Alexander Winning

SMART CONTRACT AND BLOCKCHAIN DEVELOPER



A diagram to show the relationships between structures within the program.

Variables and structures:

- `BlockHeaderStandard`: This is a struct type that represents the header of a block in the blockchain. It has six fields: `Version` (integer), `PreviousBlockHash` (string), `MerkleRoot` (string), `Timestamp` (string), `TargetDifficulty` (integer), and `Nonce` (integer).

- `BlockStandard`: This is a struct type that represents a block in the blockchain. It has six fields: `BlockHeight` (integer), `BlockHash` (string), `BlockSize` (integer), `BlockHeader` (`BlockHeaderStandard` struct), `TransactionCounter` (integer), and `Transactions` (vector of strings).

- `Block`: This is a variable of type `BlockStandard` that represents the current block being processed.

- `BlockHeader`: This is a variable of type `BlockHeaderStandard` that represents the header of the current block being processed.

- `Info`: This is a data structure (Json::Value) that stores information about accounts and settings of the blockchain. It maps an account name (string) to the account balance (integer) and the number of transactions made by the account (integer) and has a separate section for blockchain settings.

- `Blockchain`: This is a data structure (Json::Value) that stores the blocks in the blockchain. It maps the block height (string) to an object containing information about the block, including its hash, size, header, transaction count, and transactions.
- `Socket` (SOCKET): represents a socket descriptor for the server.
- `Difficulty` (int): represents the difficulty level for mining new blocks.
- `MemPool` (vector<string>): stores a list of transactions in the memory pool as a vector of strings.
- `SendBuf` (char[]): buffer for storing data to be sent to a client.
- `RecvBuf` (char[]): buffer for storing data received from a client.

Test Tables

Iterative Testing

Test No.	Module	Test Description	Test Data	Success
1a.	Split	This will be an accepted test for the split algorithm as the parameters passed to the function will be valid.	Str: "Hello;World" Token: ";"	["Hello", "World"]
1b.	Split	This will be a rejected test for the split algorithm as the parameters passed to the function will be invalid.	Str: "Hello;World" Token: "@"	["Hello;World"]
2a.	Decrypt Signature	This will be an accepted test for the Decrypt Signature algorithm as the parameters passed to the function will be valid.	Tx: "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEx/ QeVE5tZr9oZLt7Cki8KgGG/odTI/cF uvKZ06XMLr Tm7Vogi/pjOf4fv1JRyKJxZzHH/TG9I07KOziaFoO MfQ== :Hel :lo :Wo :rld:MEQCIHfctsefQLsKdcGH2 u1dyyb1HCfNk2Zx7q8YIzNbB0LzAiA55hxB8LkIT lxDalYLDqnvxFQpBKh1oyYvDJjQDC0rUw== "	True
2b.	Decrypt Signature	This will be a rejected test for the Decrypt Signature algorithm as the parameters passed to the function will be invalid.	Tx: "He: l:o :Wo :rl :d"	False
3.	Merkle Root	This will be a functionality test of the Merkle Root function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	**Dependent on 2nd Block**	--
4.	Lookup Balance	This will be a functionality test of the Lookup Balance function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Account: "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA=="	50

Test No.	Module	Test Description	Test Data	Success
5.	Lookup Transaction Count	This will be a functionality test of the Lookup Transaction Count function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	0
6.	Write Block	This will be a functionality test of the Write Block function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	The structure Block will be filled correctly	A JSON representation of the structure Block Containing the structure BlockHeader
7a.	Validate Transaction	This will be an accepted test for the Validate Transaction algorithm as the parameters passed to the function will be valid.	A completely valid transaction	True
7b.	Validate Transaction	This will be a rejected test for the Validate Transaction algorithm as the parameters passed to the function will be invalid.	A transaction with erroneous data	False
7c.	Validate Transaction	This will be a boundary test for the Validate Transaction algorithm as the parameters passed to the function will be Boundary Data.	A transaction with an account that the node has not seen prior to this transaction	False
8a.	Execute Transactions	This will be an accepted test for the Execute Transactions algorithm as the parameters passed to the function will be valid.	A list of transactions where all of the accounts are already known to the node	A perfectly represented version of the Accounts structure with all of the balances and Tx counts changed correctly in the JSON file

8b.	Execute Transactions	This will be a boundary test for the Execute Transactions algorithm as the parameters passed to the function will be Boundary Data.	A list of transactions where some of the recipients are not known to the node	A perfectly represented version of the Accounts structure with all of the new accounts, balances and Tx counts changed correctly in the JSON file
8c.	Execute Transactions	This will be a boundary test for the Execute Transactions algorithm as the parameters passed to the function will be Boundary Data.	A list of transactions where all of the recipients are not known to the node	A perfectly represented version of the Accounts structure with all of the new accounts, balances and Tx counts changed correctly in the JSON file

Test No.	Module	Test Description	Test Data	Success
9.	Scrub Memory Pool	This will be a functionality test of the Scrub Memory Pool function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	A list of transactions in the structure Block	All of the transactions in the list that where present in the Mempool removed from the Mempool
10.	Append To Buffer	This will be a functionality test of the Append To Buffer function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	The completed Block structure	The block structure concatenated and held in the buffer with separating ";" between fields and ":" between items in each transaction

11.	Load Block	This will be a functionality test of the Load Block function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	BlockHeight: The block height of any completed block in the JSON file	The block structure should be populated with the Block corresponding to the given BlockHeight
12a.	In Memory Pool	This will be an accepted test for the In Memory Pool algorithm as the parameters passed to the function will be valid.	A transaction that is in the Memory Pool	True
12b.	In Memory Pool	This will be a rejected test for the In Memory Pool algorithm as the parameters passed to the function will be invalid.	Tx: “Hello World”	False
13.	Append To Structure	This will be a functionality test of the Append To Structure function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Request: A valid request with all fields filled in form: Account, Blockheight, BlockHash, Blocksize, Version, PreviousBlockHash, MerkleRoot, Timestamp, TargetDifficulty, Nonce, TransactionCounter, Transactions, Data	The block structure should be populated with the Block corresponding to the given Request
14.	Distribute Info	This will be a functionality test of the Distribute Info function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Buf: “Hello World” as a list of characters IPAddress: The other instance of the software	The other instance of the software receiving “Hello World”
15a.	Validate Block	This will be an accepted test for the In Memory Pool algorithm as the parameters passed to the function will be valid.	The data will be a valid block.	True
15b.	Validate Block	This will be a rejected test for the In Memory Pool algorithm as the parameters passed to the function will be invalid.	The data will be a selection of blocks with invalid parameters	False
15c.	Validate Block	This will be a rejected test for the In Memory Pool algorithm as the parameters passed to the function will be invalid.	The data will be a valid block but with invalid transactions	False

Terminal Testing

Test No.	Success Criteria	Test Description	Test Type	Success
A.	1.	I will give the solution to someone with very little knowledge of blockchain and ask them to insert a given consensus algorithm into the solution only using the documentation.	Qualitative	The consensus algorithm is inserted correctly

B.	2.	I will feed the solution blocks to generate block reports. I will then provide the shareholders with them for them to review.	Qualitative	The shareholders agree with the level of detail of the reports.
C.i.	3.	This will be an accepted test for the validation of blocks as the blocks passed to the solution will be valid.	Quantitative	True
C.ii.	3.	This will be a rejected test for the validation of blocks as the blocks passed to the solution will be invalid.	Quantitative	False
D.i.	4	This will be an accepted test for the validation of transactions as the blocks (Containing transactions) passed to the solution will be valid.	Quantitative	True
D.ii.	4	This will be a rejected test for the validation of transactions as the blocks (Containing transactions) passed to the solution will be invalid.	Quantitative	False
E.	5	I will feed a valid block to the node and will have populated the known nodes structure with the IP address and port of other instances of the software.	Quantitative	The other instances of the node all receive a complete copy of the block from node zero
F.i.	6	This will be an accepted test for block requests as the requests passed to the solution will be valid.	Quantitative	The requesting node will receive a copy of the complete block of the number requested
F.ii.	6	This will be a rejected test for block requests as the request passed to the solution will be invalid.	Quantitative	Nothing will be sent back as the requested block's block number is outside of what the receiving node requested
Test No.	Success Criteria	Test Description	Test Type	Success

G.i.	7	This will test the receive balance feature of the website with any address known to the network	Quantitative	The correct balance for the given address
G.ii.	7	This will test the receive txcount feature of the website with any address known to the network	Quantitative	The correct transaction count for the given address
H.i.	8	This will test the receive transaction feature of the memory pool with a valid transaction as the test data	Quantitative	The Transaction will be added to the memory pool
H.ii.	8	This will test the ability of the memory pool to delete all used transactions from the memory pool after a block uses them	Quantitative	The transactions will be removed from the memory pool

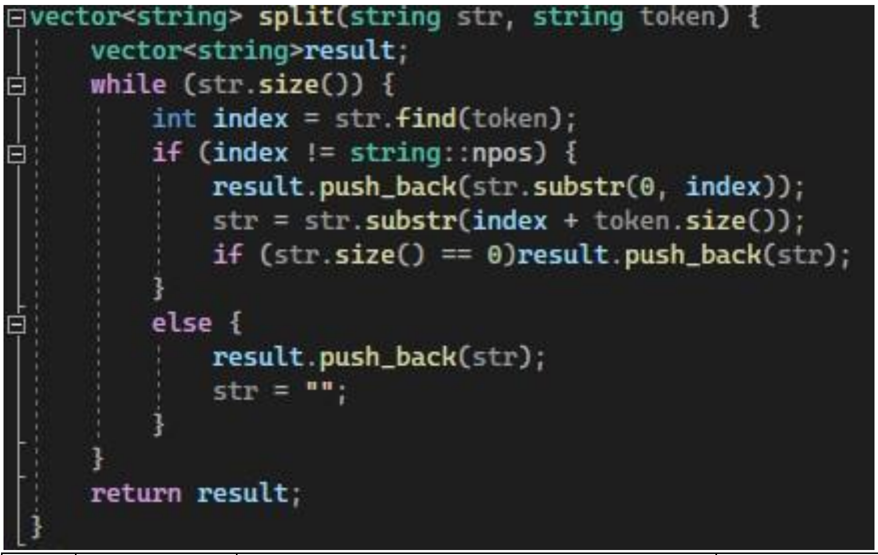
Section 3: Development

FYI comments will be included in the final tests to aid maintenance.

Modules

Module: Split

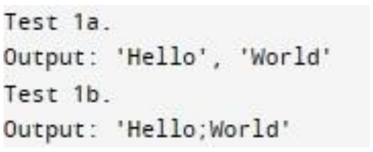
The split function underwent no major changes to its design as it is a relatively simple utility function. The only changes made where typing the variables into correct C++ compliant types and other changes to implement the vector system of C++.



```
vector<string> split(string str, string token) {
    vector<string>result;
    while (str.size()) {
        int index = str.find(token);
        if (index != string::npos) {
            result.push_back(str.substr(0, index));
            str = str.substr(index + token.size());
            if (str.size() == 0)result.push_back(str);
        }
        else {
            result.push_back(str);
            str = "";
        }
    }
    return result;
}
```

Test No.	Module	Test Description	Test Data	Success
1a.	Split	This will be an accepted test for the split algorithm as the parameters passed to the function will be valid.	Str: "Hello;World" Token: ";"	["Hello", "World"]
1b.	Split	This will be a rejected test for the split algorithm as the parameters passed to the function will be invalid.	Str: "Hello;World" Token: "@"	["Hello;World"]

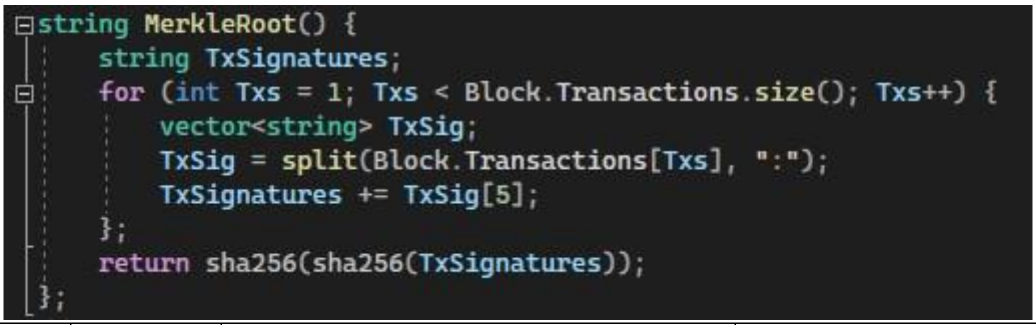
Test No.	Input	Output	Suspected error	Evidence
1a.	Str: "Hello;World" Token: ";"	["Hello","World"]	None	
1b.	Str: "Hello;World" Token: "@"	["Hello;World"]	None	



```
Test 1a.
Output: 'Hello', 'World'
Test 1b.
Output: 'Hello;World'
```

Module: MerkleRoot

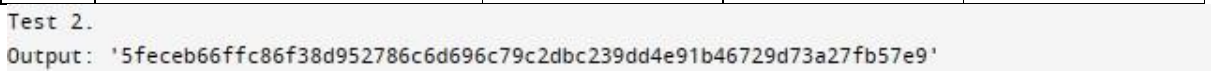
The MerkleRoot function underwent no major changes from the design algorithm. The variables used in the function were correctly typed, and there was no need to implement a signature library. The function concatenates the transaction signatures found in the block + the Merkle root of the previous block and returns their double hashed value using the SHA-256 algorithm twice, in line with the Bitcoin protocol's Merkle tree construction. Therefore, the function remains efficient and performs the necessary computation in the main program.



```
string MerkleRoot() {
    string TxSignatures;
    for (int Txs = 1; Txs < Block.Transactions.size(); Txs++) {
        vector<string> TxSig;
        TxSig = split(Block.Transactions[Txs], ":");
        TxSignatures += TxSig[5];
    };
    return sha256(sha256(TxSignatures));
};
```

Test No.	Module	Test Description	Test Data	Success
2.	Merkle Root	This will be a functionality test of the Merkle Root function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	**Dependent on 2nd Block**	--

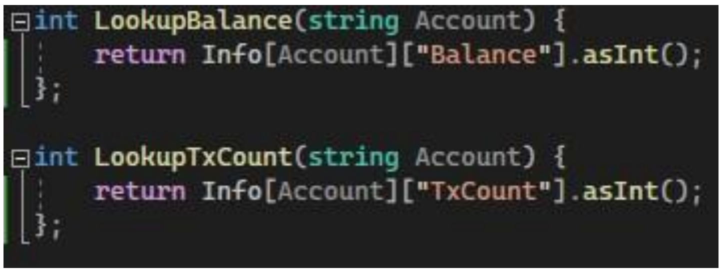
Test No.	Input	Output	Suspected error	Evidence
2.	Block 2 consisting of no transactions therefore input = '0'	2nd block Merkle root	None	



```
Test 2.
Output: '5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9'
```

Module: LookupBalance, LookupTxCount

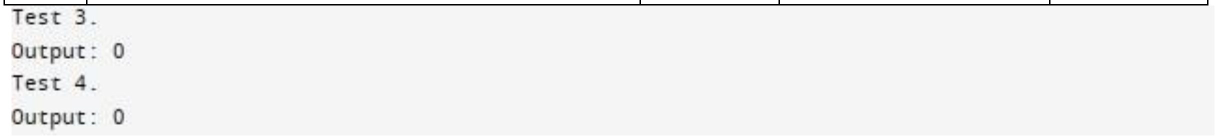
The Lookup functions underwent no major changes from design to implementation as they are very small functions only used to lookup values. They are separate functions though as they can be called through the user interface as queries.



```
int LookupBalance(string Account) {
    return Info[Account]["Balance"].asInt();
};

int LookupTxCount(string Account) {
    return Info[Account]["TxCount"].asInt();
};
```

Test No.	Module	Test Description	Test Data	Success
3.	Lookup Balance	This will be a functionality test of the Lookup Balance function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	50
4.	Lookup Transaction Count	This will be a functionality test of the Lookup Transaction Count function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	0

Test No.	Input	Output	Suspected error	Evidence
3.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	0	Calling Error	
4.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	0	Calling Error as though 0 expected it uses the same mechanism as above	

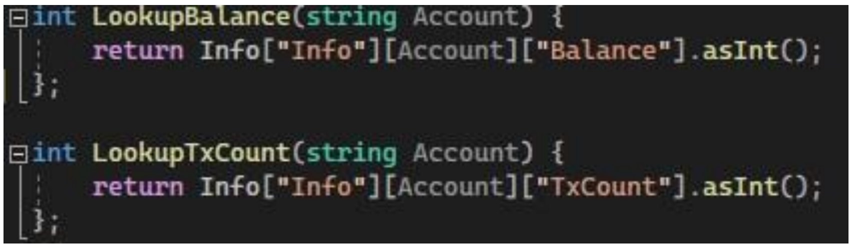


```
Test 3.
Output: 0
Test 4.
Output: 0
```

Iteration 2:

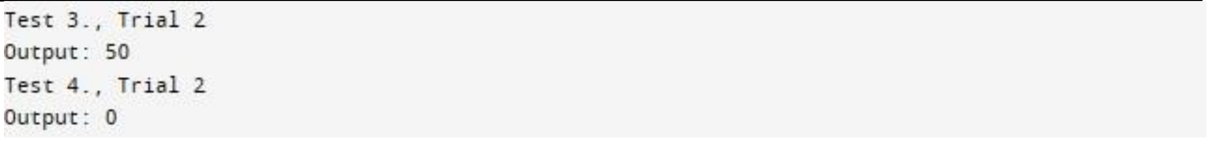
Changes:

- Reworked the calling structure of the Json to include the 1st layer “Info” as the way that I set up the JSON file included that



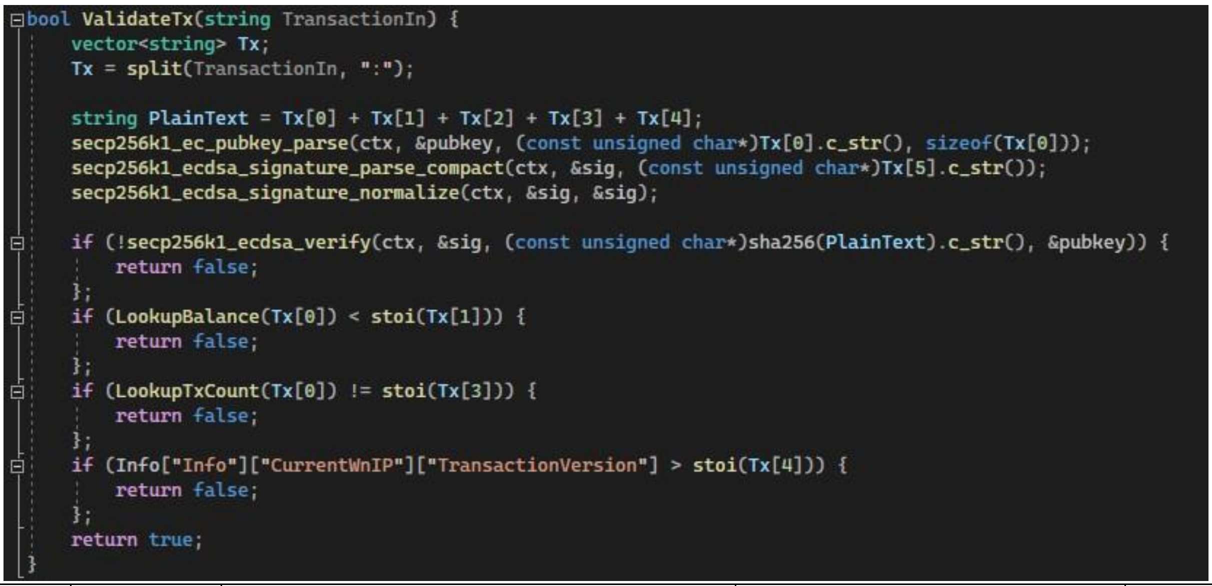
```
int LookupBalance(string Account) {
    return Info["Info"][Account]["Balance"].asInt();
};

int LookupTxCount(string Account) {
    return Info["Info"][Account]["TxCount"].asInt();
};
```

Test No.	Input	Output	Suspected error	Evidence
3.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	50	None	
4.	Account: “MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEidd1V xjJiMbigwKDKtJHuvO42wkXoM0W zYlPZMPUciW n9XEW6IANe9kjqRkwZkGnhvTvtDuXGkdVk2kMYr tFXA==”	0	None	


```
Test 3., Trial 2
Output: 50
Test 4., Trial 2
Output: 0
```

Module: ValidateTx and DecryptSignature (Merged)

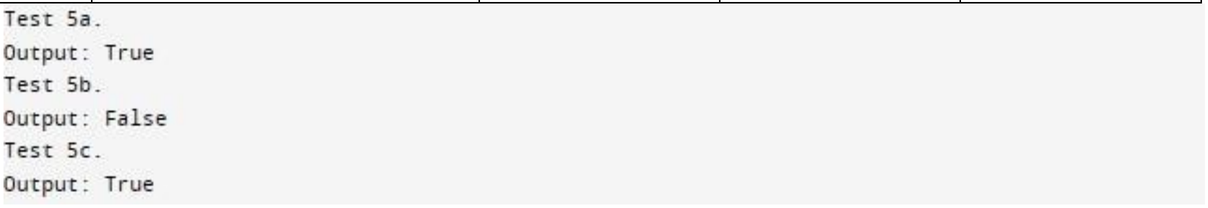
The main changes that where made from the design algorithm are centred around typing of variables and implementing the signature library. This includes parsing the public key, parsing and normalising the signature. I decided to implement it as a Boolean function as it is a checking function and none of the computations are needed in the main program. I decided to merge these 2 functions as decrypt signature was only being called by the ValidateTx function and
DecryptSignature was an extremely small function. I also took the lessons from the testing of the previous function and included [“Info”] at the start of the query of the JSON.


```cpp
bool ValidateTx(string TransactionIn) {
    vector<string> Tx;
    Tx = split(TransactionIn, ":");

    string PlainText = Tx[0] + Tx[1] + Tx[2] + Tx[3] + Tx[4];
    secp256k1_ec_pubkey_parse(ctx, &pubkey, (const unsigned char*)Tx[0].c_str(), sizeof(Tx[0]));
    secp256k1_ecdsa_signature_parse_compact(ctx, &sig, (const unsigned char*)Tx[5].c_str());
    secp256k1_ecdsa_signature_normalize(ctx, &sig, &sig);

    if (!secp256k1_ecdsa_verify(ctx, &sig, (const unsigned char*)sha256(PlainText).c_str(), &pubkey)) {
        return false;
    };
    if (LookupBalance(Tx[0]) < stoi(Tx[1])) {
        return false;
    };
    if (LookupTxCount(Tx[0]) != stoi(Tx[3])) {
        return false;
    };
    if (Info["Info"]["CurrentWnIP"]["TransactionVersion"] > stoi(Tx[4])) {
        return false;
    };
    return true;
}
```

Test No.	Module	Test Description	Test Data	Success

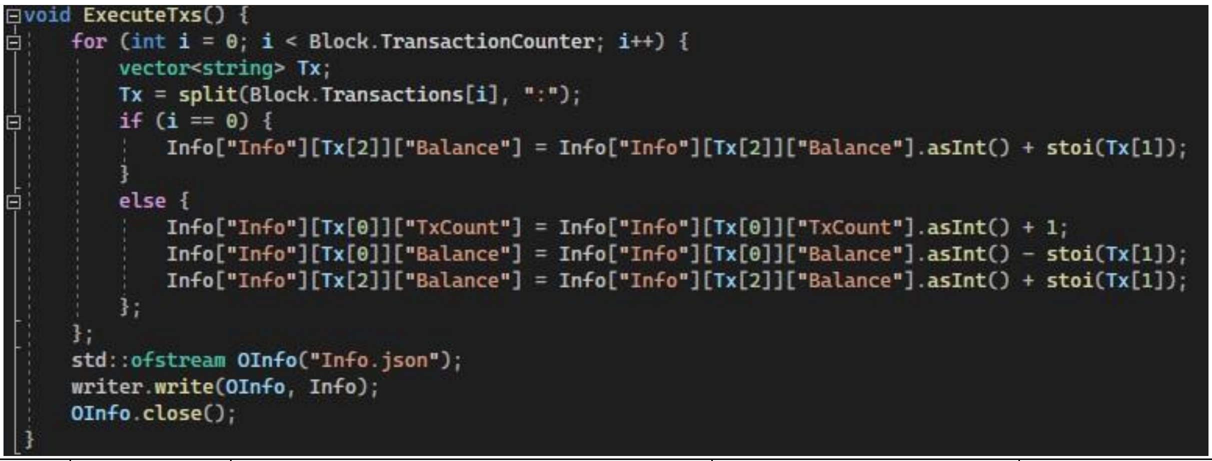
5a.	Validate Transaction	This will be an accepted test for the Validate Transaction algorithm as the parameters passed to the function will be valid.	A completely valid transaction	True
5b.	Validate Transaction	This will be a rejected test for the Validate Transaction algorithm as the parameters passed to the function will be invalid.	A transaction with erroneous data	False
5c.	Validate Transaction	This will be a boundary test for the Validate Transaction algorithm as the parameters passed to the function will be Boundary Data.	A transaction with an account that the node has not seen prior to this transaction	True

Test No.	Input	Output	Suspected error	Evidence
5a.	Valid transaction	True	None	
5b.	A transaction with erroneous data	False	None	
5c.	A transaction with an account that the node has not seen prior to this transaction	True	None	


```
Test 5a.
Output: True
Test 5b.
Output: False
Test 5c.
Output: True
```

Module: Execute TX

The main changes that were made between the design and implementation are typing of variables and JSON Queries and also the Addition of [“Info”] to the start of the Queries


```
void ExecuteTxs() {
    for (int i = 0; i < Block.TransactionCounter; i++) {
        vector<string> Tx;
        Tx = split(Block.Transactions[i], ":");
        if (i == 0) {
            Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]);
        }
        else {
            Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1;
            Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1]);
            Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]);
        };
    };
    std::ofstream OInfo("Info.json");
    writer.write(OInfo, Info);
    OInfo.close();
}
```

Test No.	Module	Test Description	Test Data	Success

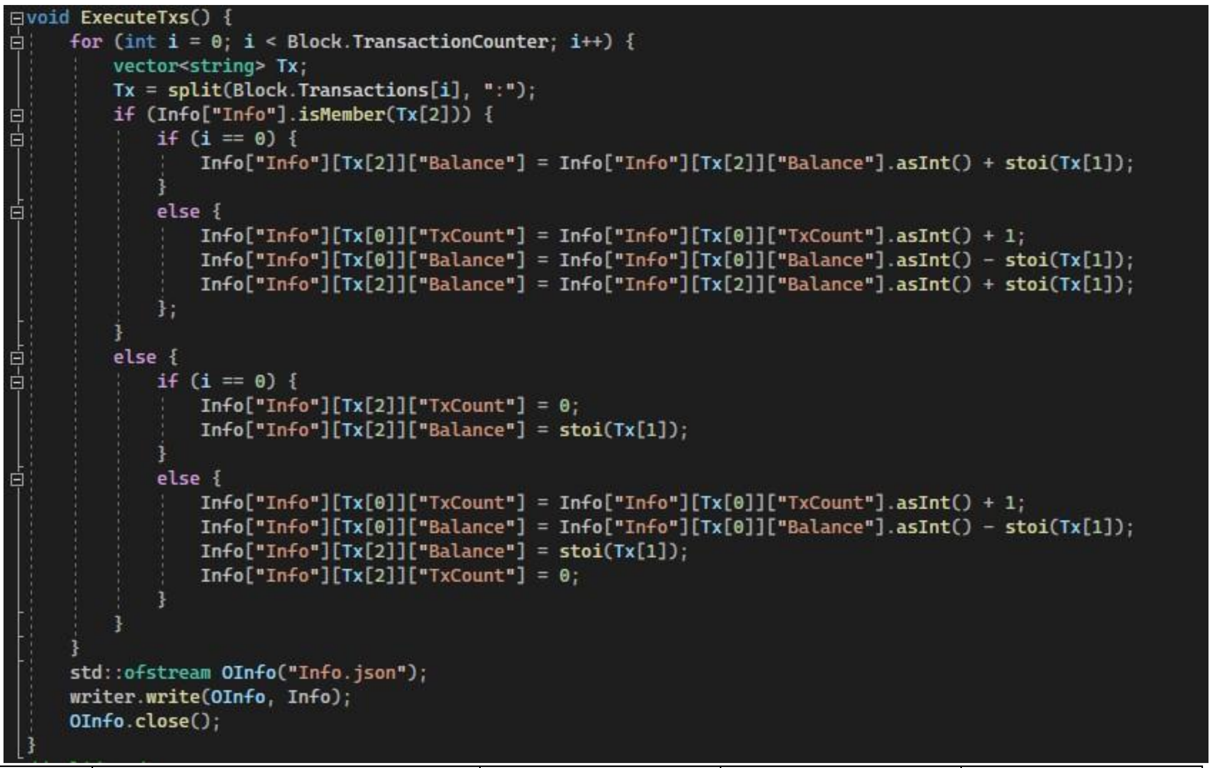
6a.	Execute Transactions	This will be an accepted test for the Execute Transactions algorithm as the parameters passed to the function will be valid.	A list of transactions where all of the accounts are already known to the node	The correct changes are reflected in the Accounts Json file
6b.	Execute Transactions	This will be a boundary test for the Execute Transactions algorithm as the parameters passed to the function will be Boundary Data.	A list of transactions where some of the recipients are not known to the node	The correct changes and additions are reflected in the Accounts Json file
6c.	Execute Transactions	This will be a boundary test for the Execute Transactions algorithm as the parameters passed to the function will be Boundary Data.	A list of transactions where all of the recipients are not known to the node	The correct changes and additions are reflected in the Accounts Json file

Test No.	Input	Output	Suspected error	Evidence
6a.	A list of transactions where all of the accounts are already known to the node	Correct	None	
6b.	A list of transactions where some of the recipients are not known to the node	Incorrect	Incorrect Json structuring code	
6c.	A list of transactions where all of the recipients are not known to the node	Incorrect	Incorrect Json structuring code	

```
Test 6.
Output: Json screenshot below
```

Iteration 2 Changes:

- Restructured the code creating new accounts to include a check for whether the recipient account exists in the node's database



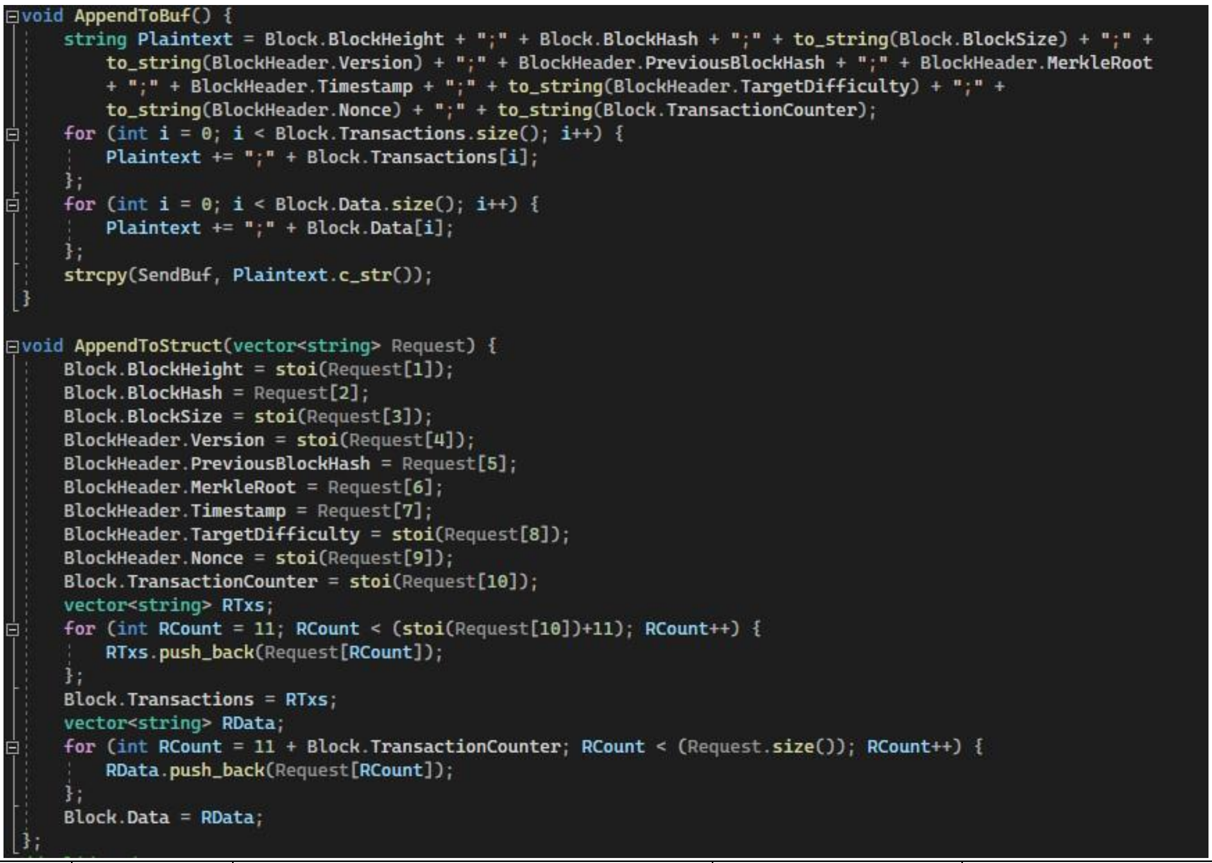
```
void ExecuteTxs() {
    for (int i = 0; i < Block.TransactionCounter; i++) {
        vector<string> Tx;
        Tx = split(Block.Transactions[i], ":");
        if (Info["Info"].isMember(Tx[2])) {
            if (i == 0) {
                Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]);
            }
            else {
                Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1;
                Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1]);
                Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]);
            };
        }
        else {
            if (i == 0) {
                Info["Info"][Tx[2]]["TxCount"] = 0;
                Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1]);
            }
            else {
                Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1;
                Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1]);
                Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1]);
                Info["Info"][Tx[2]]["TxCount"] = 0;
            }
        }
    }
    std::ofstream OInfo("Info.json");
    writer.write(OInfo, Info);
    OInfo.close();
}
```

Test No.	Input	Output	Suspected error	Evidence
6a.	A list of transactions where all of the accounts are already known to the node	Correct	None	
6b.	A list of transactions where some of the recipients are not known to the node	Correct	None	
6c.	A list of transactions where all of the recipients are not known to the node	Correct	None	

```
Test 6.
Output: Json screenshot below
```

Modules: AppendToBuf and AppendToStruct

There were major changes to these functions including the addition of the separator of ";" and ":" for the append to buffer function and the better setup of the structures Block and Info. Info now contains the 2 structures Accounts and settings as the declaration of 2 structures both pulling from the same Json structure would have been inefficient. The other changes would be typing and declaring a default buffer length but that is above what would be expected from a normal block and functions as the maximum block size.



```
void AppendToBuf() {
    string Plaintext = Block.BlockHeight + ";" + Block.BlockHash + ";" + to_string(Block.BlockSize) + ";" +
        to_string(BlockHeader.Version) + ";" + BlockHeader.PreviousBlockHash + ";" + BlockHeader.MerkleRoot
        + ";" + BlockHeader.Timestamp + ";" + to_string(BlockHeader.TargetDifficulty) + ";" +
        to_string(BlockHeader.Nonce) + ";" + to_string(Block.TransactionCounter);
    for (int i = 0; i < Block.Transactions.size(); i++) {
        Plaintext += ";" + Block.Transactions[i];
    };
    for (int i = 0; i < Block.Data.size(); i++) {
        Plaintext += ";" + Block.Data[i];
    };
    strcpy(SendBuf, Plaintext.c_str());
}

void AppendToStruct(vector<string> Request) {
    Block.BlockHeight = stoi(Request[1]);
    Block.BlockHash = Request[2];
    Block.BlockSize = stoi(Request[3]);
    BlockHeader.Version = stoi(Request[4]);
    BlockHeader.PreviousBlockHash = Request[5];
    BlockHeader.MerkleRoot = Request[6];
    BlockHeader.Timestamp = Request[7];
    BlockHeader.TargetDifficulty = stoi(Request[8]);
    BlockHeader.Nonce = stoi(Request[9]);
    Block.TransactionCounter = stoi(Request[10]);
    vector<string> RTxs;
    for (int RCount = 11; RCount < (stoi(Request[10])+11); RCount++) {
        RTxs.push_back(Request[RCount]);
    };
    Block.Transactions = RTxs;
    vector<string> RData;
    for (int RCount = 11 + Block.TransactionCounter; RCount < (Request.size()); RCount++) {
        RData.push_back(Request[RCount]);
    };
    Block.Data = RData;
};
```

Test No.	Module	Test Description	Test Data	Success
7.	Append To Structure	This will be a functionality test of the Append To Structure function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	The completed block in a buffer	The block concatenated and held in the structure with the separating ";" the fields removed
8.	Append To Buffer	This will be a functionality test of the Append To Buffer function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	The completed Block structure	The block structure concatenated and held in the buffer with separating ";" between fields and ":" between items in each transaction

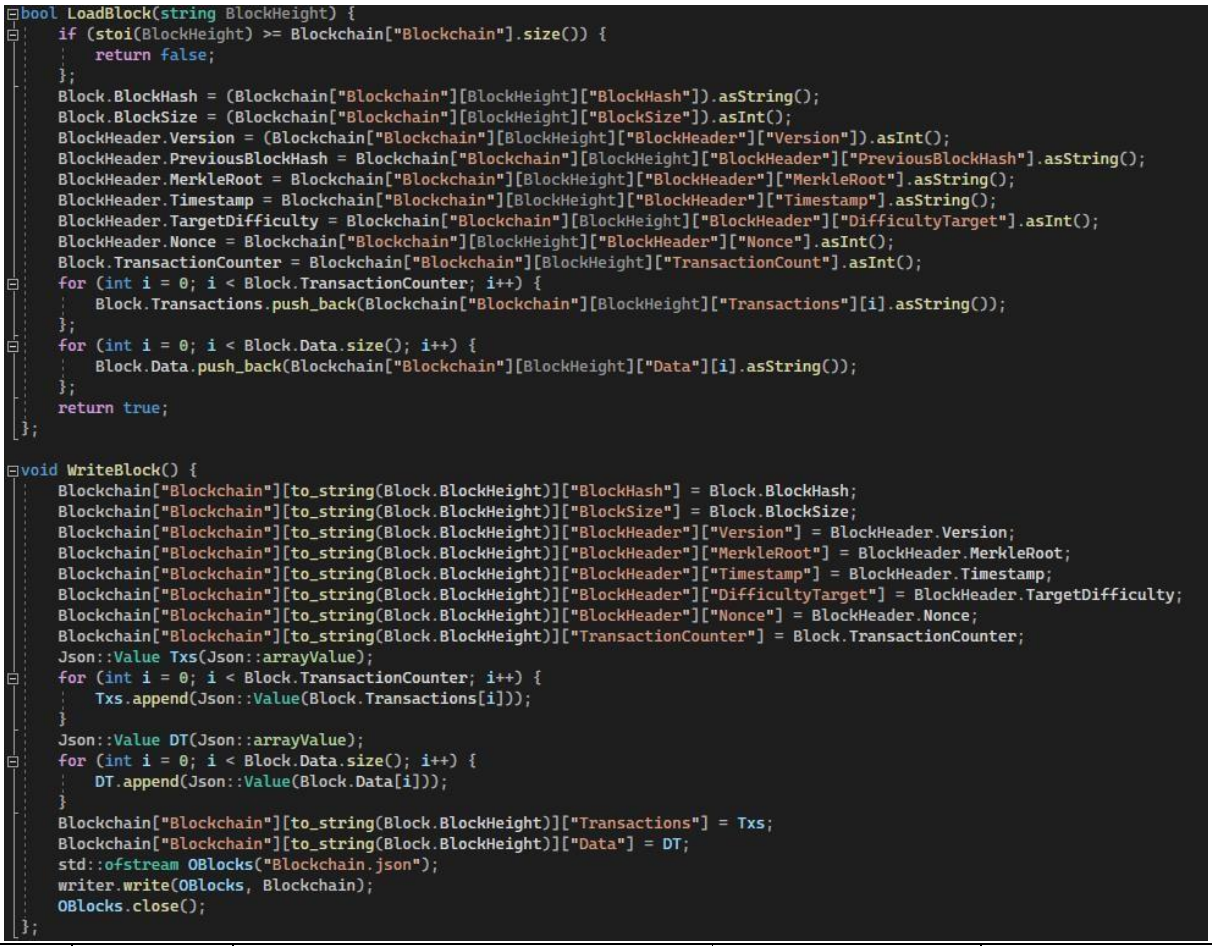
Test No.	Input	Output	Suspected error	Evidence
7.	1;0d801975e70522a04983626a170f8dc1f2491e62bcd188fc0375d7e2b344cf81;1;5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9;26; 081a372b17e83bc2bede64531cbef00	Correct	None	

	9f04b1f57ec02e443d98904358b47dd8a;1912'04/06/22;1;284;1;0000:50:0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300:0:1			
8.	A fully completed block structure filled with the information for block height 1	Correct	None	

```
Test 7.
Output: '1;0d801975e70522a04983626a170f8dc1f2491e62bcd188fc0375d7e2b344cf81;1
    ;5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9;26;
081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a;1912'04/06/22;1;284;1;0000:50
    :0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300:0:1'
Test 8.
Output: Screenshot below
```

Modules: WriteBlock and LoadBlock

The changes made to these functions were to add [“info”] before all of the JSON calls and typing of the elements of the block structure.



```
bool LoadBlock(string BlockHeight) {
    if (stoi(BlockHeight) >= Blockchain["Blockchain"].size()) {
        return false;
    };
    Block.BlockHash = (Blockchain["Blockchain"][BlockHeight]["BlockHash"]).asString();
    Block.BlockSize = (Blockchain["Blockchain"][BlockHeight]["BlockSize"]).asInt();
    BlockHeader.Version = (Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Version"]).asInt();
    BlockHeader.PreviousBlockHash = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["PreviousBlockHash"].asString();
    BlockHeader.MerkleRoot = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["MerkleRoot"].asString();
    BlockHeader.Timestamp = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Timestamp"].asString();
    BlockHeader.TargetDifficulty = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["DifficultyTarget"].asInt();
    BlockHeader.Nonce = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Nonce"].asInt();
    Block.TransactionCounter = Blockchain["Blockchain"][BlockHeight]["TransactionCount"].asInt();
    for (int i = 0; i < Block.TransactionCounter; i++) {
        Block.Transactions.push_back(Blockchain["Blockchain"][BlockHeight]["Transactions"][i].asString());
    };
    for (int i = 0; i < Block.Data.size(); i++) {
        Block.Data.push_back(Blockchain["Blockchain"][BlockHeight]["Data"][i].asString());
    };
    return true;
};

void WriteBlock() {
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHash"] = Block.BlockHash;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockSize"] = Block.BlockSize;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Version"] = BlockHeader.Version;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["MerkleRoot"] = BlockHeader.MerkleRoot;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Timestamp"] = BlockHeader.Timestamp;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["DifficultyTarget"] = BlockHeader.TargetDifficulty;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Nonce"] = BlockHeader.Nonce;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["TransactionCounter"] = Block.TransactionCounter;
    Json::Value Txs(Json::arrayValue);
    for (int i = 0; i < Block.TransactionCounter; i++) {
        Txs.append(Json::Value(Block.Transactions[i]));
    }
    Json::Value DT(Json::arrayValue);
    for (int i = 0; i < Block.Data.size(); i++) {
        DT.append(Json::Value(Block.Data[i]));
    }
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["Transactions"] = Txs;
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["Data"] = DT;
    std::ofstream OBlocks("Blockchain.json");
    writer.write(OBlocks, Blockchain);
    OBlocks.close();
};
```

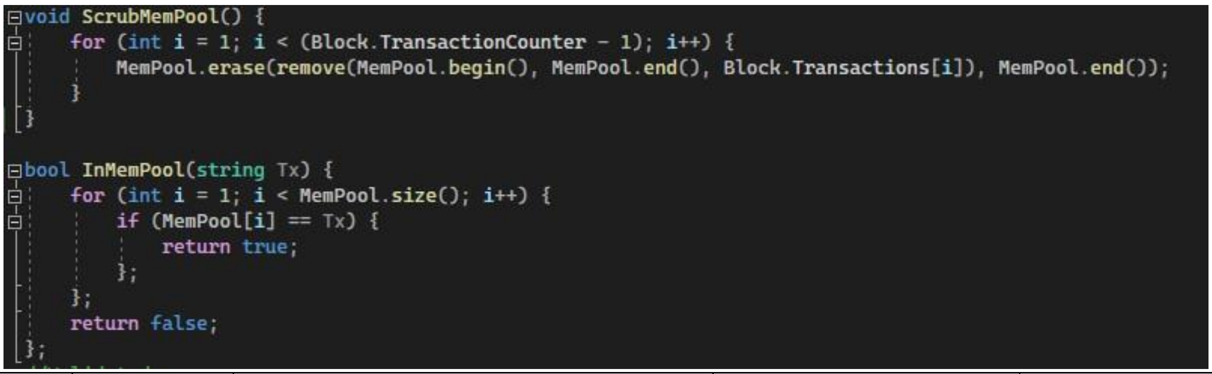
Test No.	Module	Test Description	Test Data	Success

9.	Write Block	This will be a functionality test of the Write Block function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	The structure Block will be filled correctly	A JSON representation of the structure Block Containing the structure BlockHeader
10.	Load Block	This will be a functionality test of the Load Block function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	BlockHeight: The block height of any completed block in the JSON file	The block structure should be populated with the Block corresponding to the given BlockHeight

Test No.	Input	Output	Suspected error	Evidence
9.	Block 1	Correct	None	
10.	Blockheight = 1	Correct	None	

```
Test 9.
Output: Json screenshot below
Test 10.
Output: Screenshot below
```

Module: InMempool and ScrubMempool



```
void ScrubMemPool() {
    for (int i = 1; i < (Block.TransactionCounter - 1); i++) {
        MemPool.erase(remove(MemPool.begin(), MemPool.end(), Block.Transactions[i]), MemPool.end());
    }
}

bool InMemPool(string Tx) {
    for (int i = 1; i < MemPool.size(); i++) {
        if (MemPool[i] == Tx) {
            return true;
        };
    };
    return false;
};
```

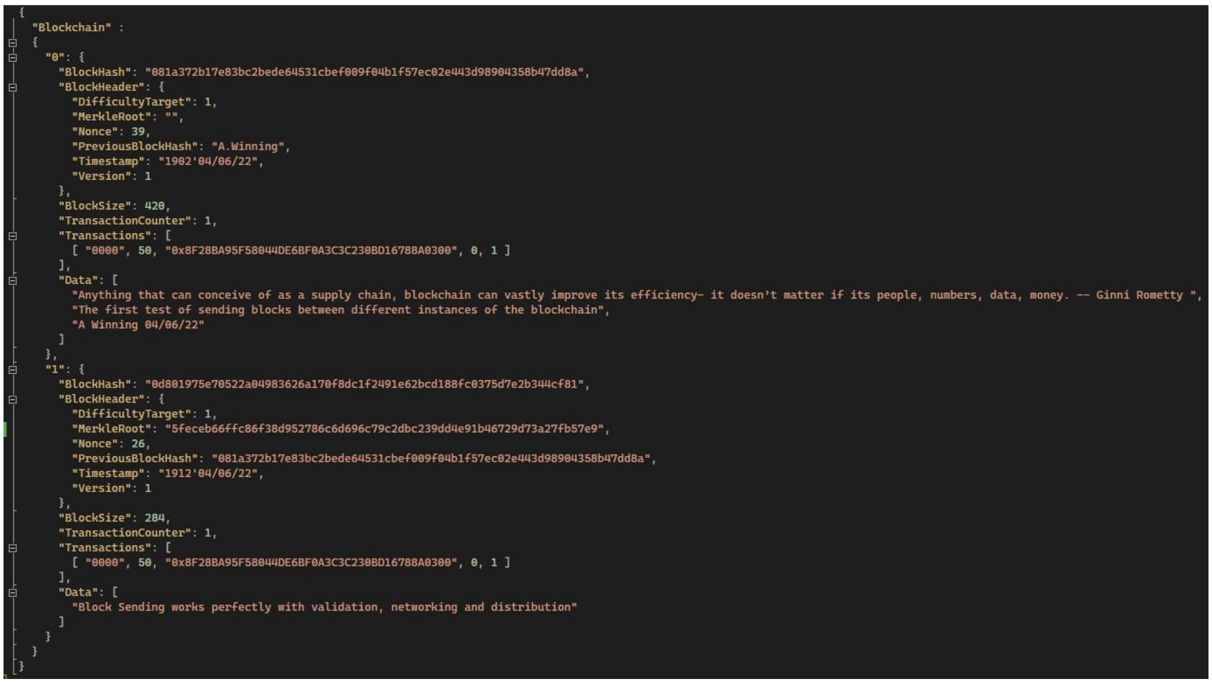
Test No.	Module	Test Description	Test Data	Success
11.	Scrub Memory Pool	This will be a functionality test of the Scrub Memory Pool function. As it cannot pass or fail cannot be tested like the others, but it can be tested for accuracy.	A list of transactions in the structure Block	All of the transactions in the list that where present in the Mempool removed from the Mempool
12a.	In Memory Pool	This will be an accepted test for the In Memory Pool algorithm as the parameters passed to the function will be valid.	A transaction that is in the Memory Pool	True

12b.	In Memory Pool	This will be a rejected test for the In Memory Pool algorithm as the parameters passed to the function will be invalid.	Tx: ”Hello World”	False

Test No.	Input	Output	Suspected error	Evidence
11.	A list of transactions in the structure Block	Correct	None	
12a.	A transaction that is in the Memory Pool	True	None	
12b.	Tx: ”Hello World”	False	None	

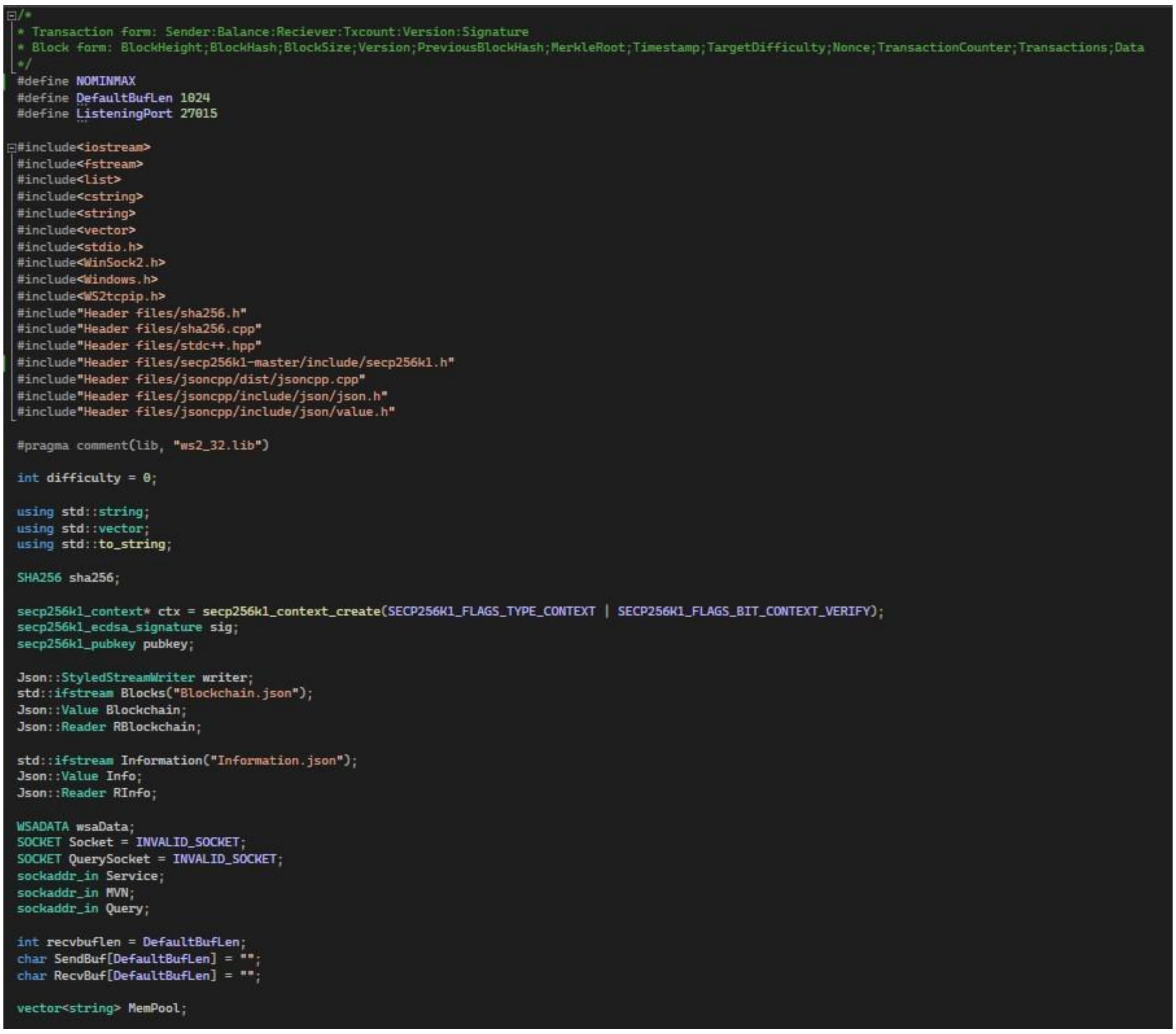
```
Test 11.
Output: 0
Test 12a.
Output: True
Test 12b.
Output: False
```

Json screenshots

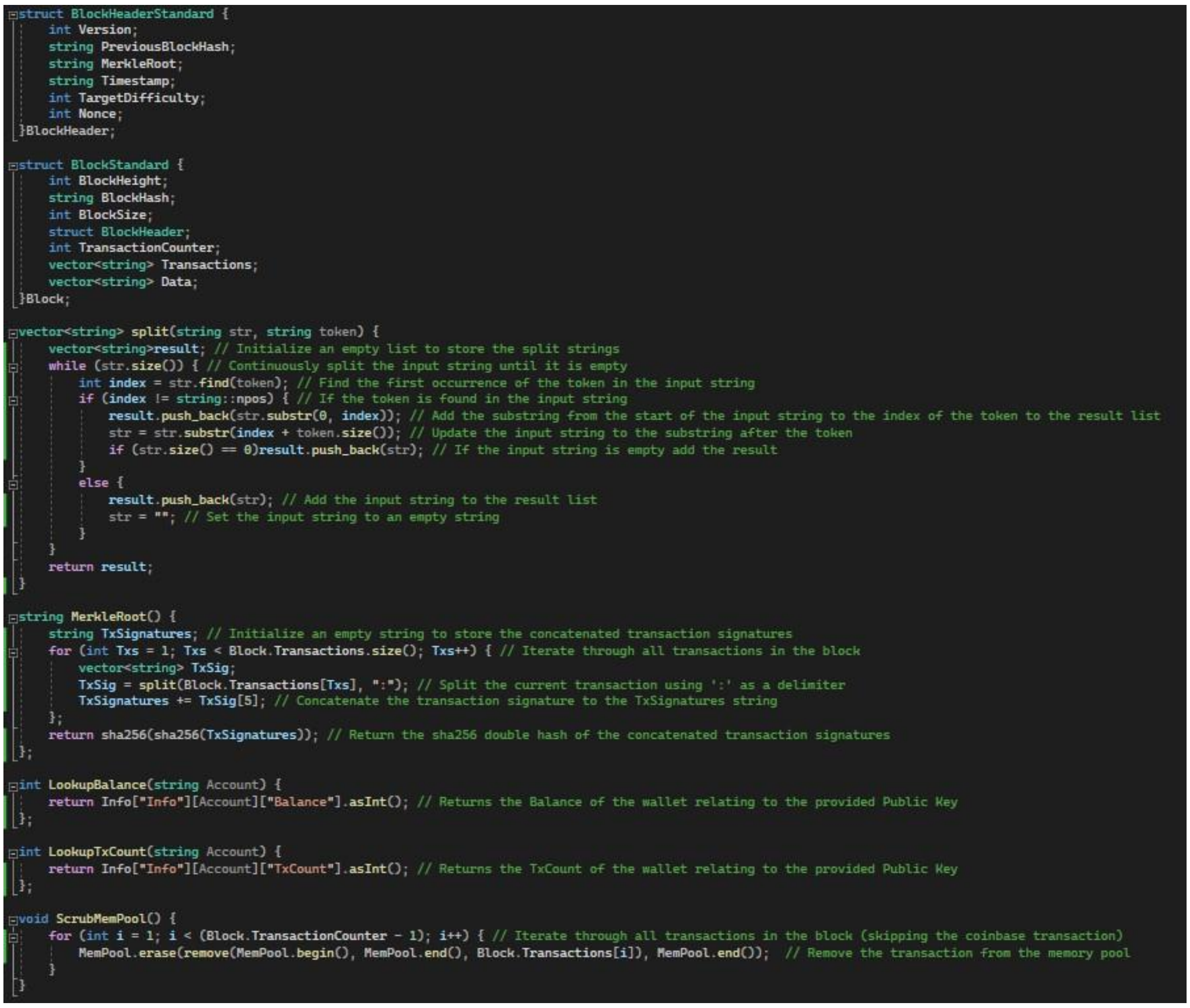


```
{
  "Blockchain" :
  {
    "0": {
      "BlockHash": "081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a",
      "BlockHeader": {
        "DifficultyTarget": 1,
        "MerkleRoot": "",
        "Nonce": 39,
        "PreviousBlockHash": "A.Winning",
        "Timestamp": "1902'04/06/22",
        "Version": 1
      },
      "BlockSize": 420,
      "TransactionCounter": 1,
      "Transactions": [
        [ "0000", 50, "0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300", 0, 1 ]
      ],
      "Data": [
        "Anything that can conceive of as a supply chain, blockchain can vastly improve its efficiency- it doesn’t matter if its people, numbers, data, money. -- Ginni Rometty ",
        "The first test of sending blocks between different instances of the blockchain",
        "A Winning 04/06/22"
      ]
    },
    "1": {
      "BlockHash": "0d801975e70522a04983626a170f8dc1f2491e62bcd188fc0375d7e2b344cf81",
      "BlockHeader": {
        "DifficultyTarget": 1,
        "MerkleRoot": "5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9",
        "Nonce": 26,
        "PreviousBlockHash": "081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a",
        "Timestamp": "1912'04/06/22",
        "Version": 1
      },
      "BlockSize": 284,
      "TransactionCounter": 1,
      "Transactions": [
        [ "0000", 50, "0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300", 0, 1 ]
      ],
      "Data": [
        "Block Sending works perfectly with validation, networking and distribution"
      ]
    }
  }
}
```

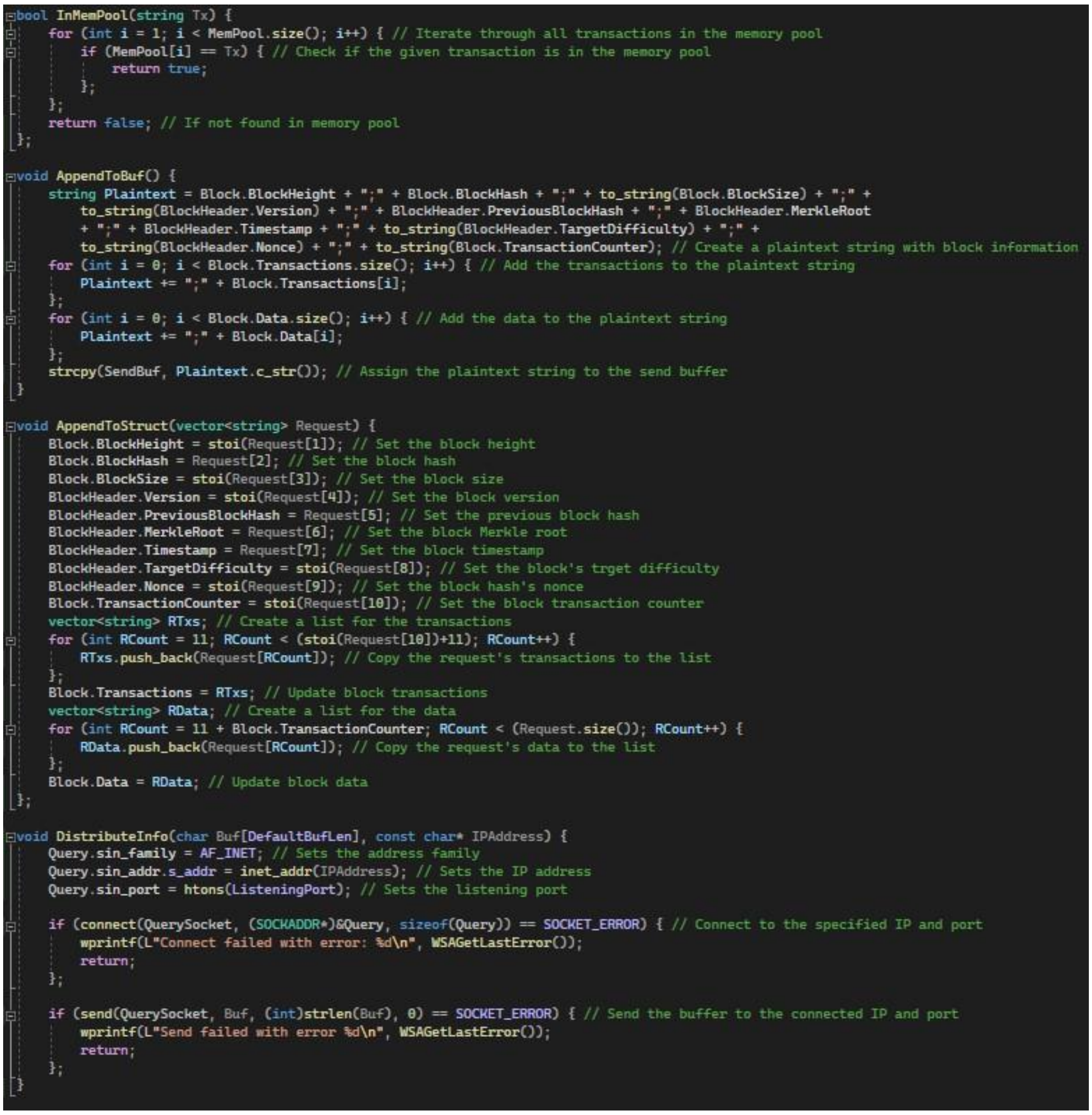
Code Screenshots with annotation



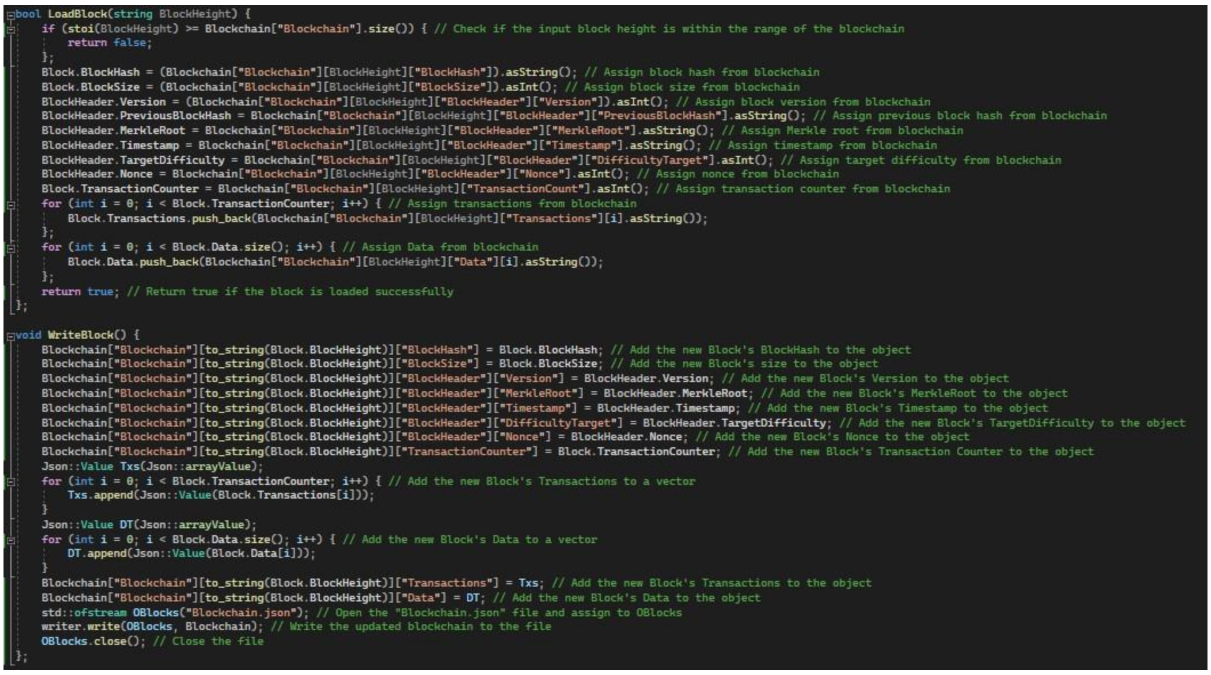
```
/*
* Transaction form: Sender:Balance:Reciever:Txcount:Version:Signature
* Block form: BlockHeight;BlockHash;BlockSize;Version;PreviousBlockHash;MerkleRoot;Timestamp;TargetDifficulty;Nonce;TransactionCounter;Transactions;Data
*/
#define NOMINMAX
#define DefaultBufLen 1024
#define ListeningPort 27015

#include<iostream>
#include<fstream>
#include<list>
#include<cstring>
#include<string>
#include<vector>
#include<stdio.h>
#include<WinSock2.h>
#include<Windows.h>
#include<WS2tcpip.h>
#include"Header files/sha256.h"
#include"Header files/sha256.cpp"
#include"Header files/stdc++.hpp"
#include"Header files/secp256k1-master/include/secp256k1.h"
#include"Header files/jsoncpp/dist/jsoncpp.cpp"
#include"Header files/jsoncpp/include/json/json.h"
#include"Header files/jsoncpp/include/json/value.h"

#pragma comment(lib, "ws2_32.lib")

int difficulty = 0;

using std::string;
using std::vector;
using std::to_string;

SHA256 sha256;

secp256k1_context* ctx = secp256k1_context_create(SECP256K1_FLAGS_TYPE_CONTEXT | SECP256K1_FLAGS_BIT_CONTEXT_VERIFY);
secp256k1_ecdsa_signature sig;
secp256k1_pubkey pubkey;

Json::StyledStreamWriter writer;
std::ifstream Blocks("Blockchain.json");
Json::Value Blockchain;
Json::Reader RBlockchain;

std::ifstream Information("Information.json");
Json::Value Info;
Json::Reader RInfo;

WSADATA wsaData;
SOCKET Socket = INVALID_SOCKET;
SOCKET QuerySocket = INVALID_SOCKET;
sockaddr_in Service;
sockaddr_in MVN;
sockaddr_in Query;

int recvbuflen = DefaultBufLen;
char SendBuf[DefaultBufLen] = "";
char RecvBuf[DefaultBufLen] = "";

vector<string> MemPool;
```



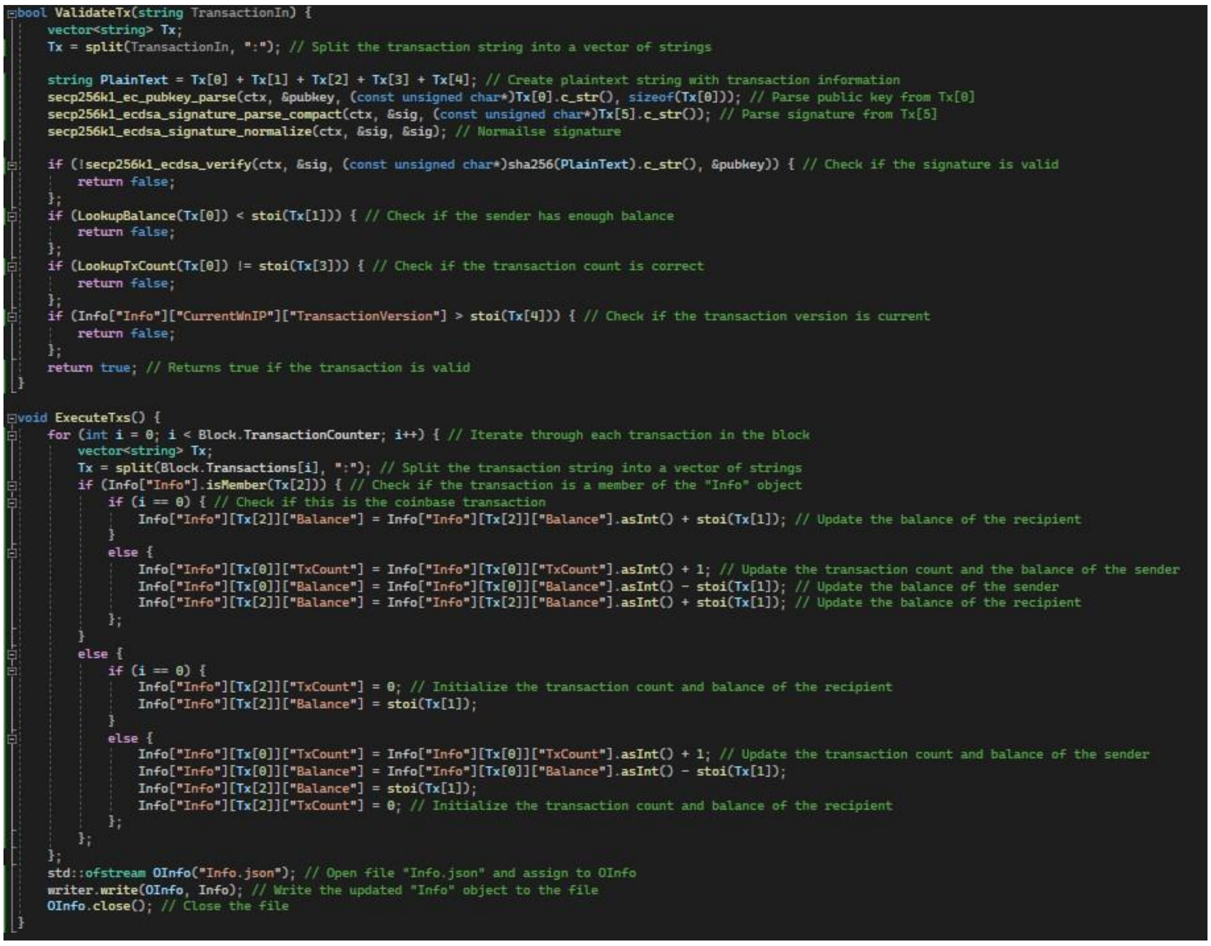
```cpp
struct BlockHeaderStandard {
    int Version;
    string PreviousBlockHash;
    string MerkleRoot;
    string Timestamp;
    int TargetDifficulty;
    int Nonce;
}BlockHeader;

struct BlockStandard {
    int BlockHeight;
    string BlockHash;
    int BlockSize;
    struct BlockHeader;
    int TransactionCounter;
    vector<string> Transactions;
    vector<string> Data;
}Block;

vector<string> split(string str, string token) {
    vector<string>result; // Initialize an empty list to store the split strings
    while (str.size()) { // Continuously split the input string until it is empty
        int index = str.find(token); // Find the first occurrence of the token in the input string
        if (index != string::npos) { // If the token is found in the input string
            result.push_back(str.substr(0, index)); // Add the substring from the start of the input string to the index of the token to the result list
            str = str.substr(index + token.size()); // Update the input string to the substring after the token
            if (str.size() == 0)result.push_back(str); // If the input string is empty add the result
        }
        else {
            result.push_back(str); // Add the input string to the result list
            str = ""; // Set the input string to an empty string
        }
    }
    return result;
}

string MerkleRoot() {
    string TxSignatures; // Initialize an empty string to store the concatenated transaction signatures
    for (int Txs = 1; Txs < Block.Transactions.size(); Txs++) { // Iterate through all transactions in the block
        vector<string> TxSig;
        TxSig = split(Block.Transactions[Txs], ":"); // Split the current transaction using ':' as a delimiter
        TxSignatures += TxSig[5]; // Concatenate the transaction signature to the TxSignatures string
    };
    return sha256(sha256(TxSignatures)); // Return the sha256 double hash of the concatenated transaction signatures
};

int LookupBalance(string Account) {
    return Info["Info"][Account]["Balance"].asInt(); // Returns the Balance of the wallet relating to the provided Public Key
};

int LookupTxCount(string Account) {
    return Info["Info"][Account]["TxCount"].asInt(); // Returns the TxCount of the wallet relating to the provided Public Key
};

void ScrubMemPool() {
    for (int i = 1; i < (Block.TransactionCounter - 1); i++) { // Iterate through all transactions in the block (skipping the coinbase transaction)
        MemPool.erase(remove(MemPool.begin(), MemPool.end(), Block.Transactions[i]), MemPool.end());  // Remove the transaction from the memory pool
    }
}
```



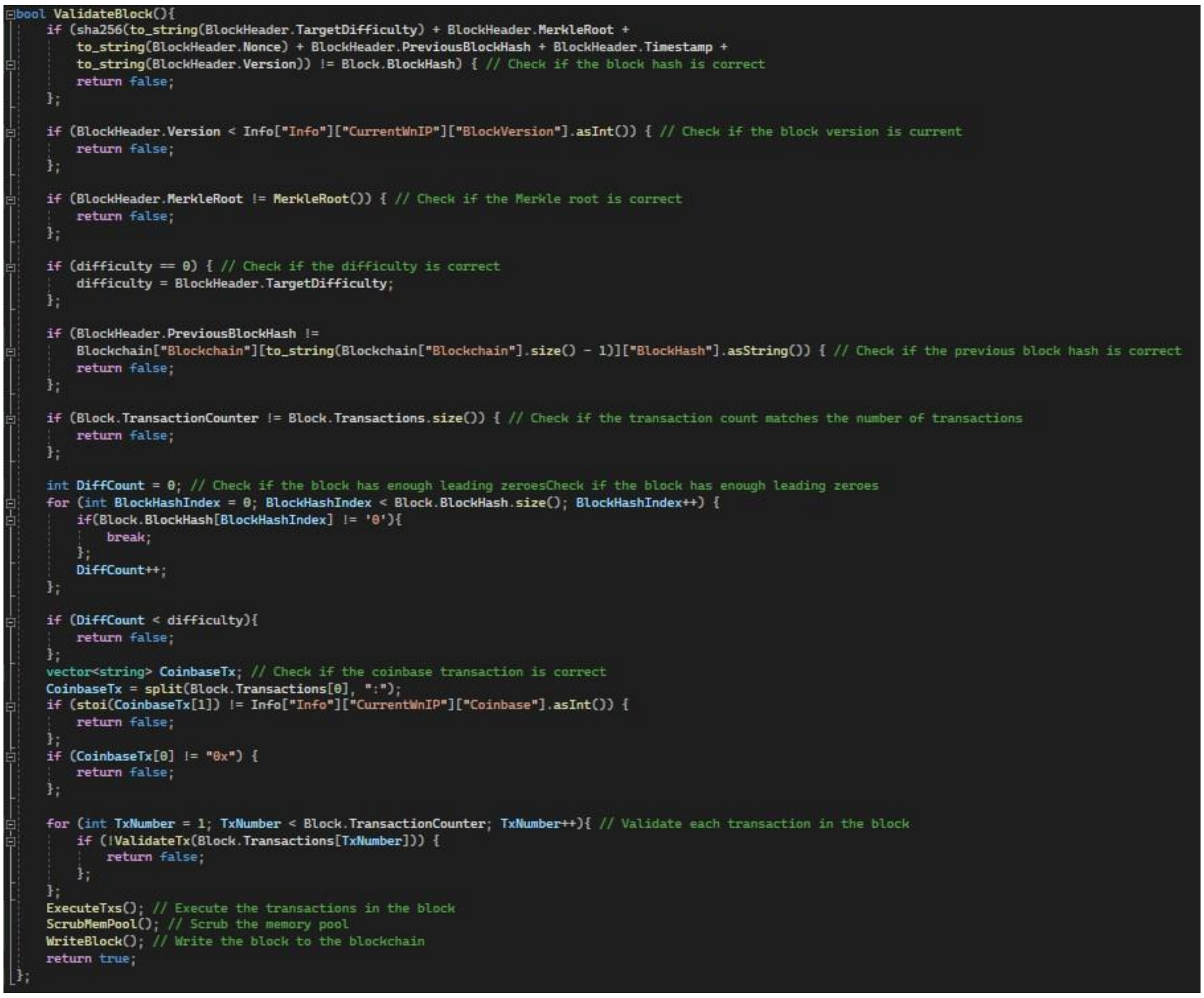
```
bool InMemPool(string Tx) {
    for (int i = 1; i < MemPool.size(); i++) { // Iterate through all transactions in the memory pool
        if (MemPool[i] == Tx) { // Check if the given transaction is in the memory pool
            return true;
        };
    };
    return false; // If not found in memory pool
};

void AppendToBuf() {
    string Plaintext = Block.BlockHeight + ";" + Block.BlockHash + ";" + to_string(Block.BlockSize) + ";" +
        to_string(BlockHeader.Version) + ";" + BlockHeader.PreviousBlockHash + ";" + BlockHeader.MerkleRoot
        + ";" + BlockHeader.Timestamp + ";" + to_string(BlockHeader.TargetDifficulty) + ";" +
        to_string(BlockHeader.Nonce) + ";" + to_string(Block.TransactionCounter); // Create a plaintext string with block information
    for (int i = 0; i < Block.Transactions.size(); i++) { // Add the transactions to the plaintext string
        Plaintext += ";" + Block.Transactions[i];
    };
    for (int i = 0; i < Block.Data.size(); i++) { // Add the data to the plaintext string
        Plaintext += ";" + Block.Data[i];
    };
    strcpy(SendBuf, Plaintext.c_str()); // Assign the plaintext string to the send buffer
}

void AppendToStruct(vector<string> Request) {
    Block.BlockHeight = stoi(Request[1]); // Set the block height
    Block.BlockHash = Request[2]; // Set the block hash
    Block.BlockSize = stoi(Request[3]); // Set the block size
    BlockHeader.Version = stoi(Request[4]); // Set the block version
    BlockHeader.PreviousBlockHash = Request[5]; // Set the previous block hash
    BlockHeader.MerkleRoot = Request[6]; // Set the block Merkle root
    BlockHeader.Timestamp = Request[7]; // Set the block timestamp
    BlockHeader.TargetDifficulty = stoi(Request[8]); // Set the block's trget difficulty
    BlockHeader.Nonce = stoi(Request[9]); // Set the block hash's nonce
    Block.TransactionCounter = stoi(Request[10]); // Set the block transaction counter
    vector<string> RTxs; // Create a list for the transactions
    for (int RCount = 11; RCount < (stoi(Request[10])+11); RCount++) {
        RTxs.push_back(Request[RCount]); // Copy the request's transactions to the list
    };
    Block.Transactions = RTxs; // Update block transactions
    vector<string> RData; // Create a list for the data
    for (int RCount = 11 + Block.TransactionCounter; RCount < (Request.size()); RCount++) {
        RData.push_back(Request[RCount]); // Copy the request's data to the list
    };
    Block.Data = RData; // Update block data
};

void DistributeInfo(char Buf[DefaultBufLen], const char* IPAddress) {
    Query.sin_family = AF_INET; // Sets the address family
    Query.sin_addr.s_addr = inet_addr(IPAddress); // Sets the IP address
    Query.sin_port = htons(ListeningPort); // Sets the listening port

    if (connect(QuerySocket, (SOCKADDR*)&Query, sizeof(Query)) == SOCKET_ERROR) { // Connect to the specified IP and port
        wprintf(L"Connect failed with error: %d\n", WSAGetLastError());
        return;
    };

    if (send(QuerySocket, Buf, (int)strlen(Buf), 0) == SOCKET_ERROR) { // Send the buffer to the connected IP and port
        wprintf(L"Send failed with error %d\n", WSAGetLastError());
        return;
    };
}
```



```cpp
bool LoadBlock(string BlockHeight) {
    if (stoi(BlockHeight) >= Blockchain["Blockchain"].size()) { // Check if the input block height is within the range of the blockchain
        return false;
    };
    Block.BlockHash = (Blockchain["Blockchain"][BlockHeight]["BlockHash"]).asString(); // Assign block hash from blockchain
    Block.BlockSize = (Blockchain["Blockchain"][BlockHeight]["BlockSize"]).asInt(); // Assign block size from blockchain
    BlockHeader.Version = (Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Version"]).asInt(); // Assign block version from blockchain
    BlockHeader.PreviousBlockHash = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["PreviousBlockHash"].asString(); // Assign previous block hash from blockchain
    BlockHeader.MerkleRoot = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["MerkleRoot"].asString(); // Assign Merkle root from blockchain
    BlockHeader.Timestamp = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Timestamp"].asString(); // Assign timestamp from blockchain
    BlockHeader.TargetDifficulty = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["DifficultyTarget"].asInt(); // Assign target difficulty from blockchain
    BlockHeader.Nonce = Blockchain["Blockchain"][BlockHeight]["BlockHeader"]["Nonce"].asInt(); // Assign nonce from blockchain
    Block.TransactionCounter = Blockchain["Blockchain"][BlockHeight]["TransactionCount"].asInt(); // Assign transaction counter from blockchain
    for (int i = 0; i < Block.TransactionCounter; i++) { // Assign transactions from blockchain
        Block.Transactions.push_back(Blockchain["Blockchain"][BlockHeight]["Transactions"][i].asString());
    };
    for (int i = 0; i < Block.Data.size(); i++) { // Assign Data from blockchain
        Block.Data.push_back(Blockchain["Blockchain"][BlockHeight]["Data"][i].asString());
    };
    return true; // Return true if the block is loaded successfully
};

void WriteBlock() {
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHash"] = Block.BlockHash; // Add the new Block's BlockHash to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockSize"] = Block.BlockSize; // Add the new Block's size to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Version"] = BlockHeader.Version; // Add the new Block's Version to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["MerkleRoot"] = BlockHeader.MerkleRoot; // Add the new Block's MerkleRoot to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Timestamp"] = BlockHeader.Timestamp; // Add the new Block's Timestamp to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["DifficultyTarget"] = BlockHeader.TargetDifficulty; // Add the new Block's TargetDifficulty to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["BlockHeader"]["Nonce"] = BlockHeader.Nonce; // Add the new Block's Nonce to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["TransactionCounter"] = Block.TransactionCounter; // Add the new Block's Transaction Counter to the object
    Json::Value Txs(Json::arrayValue);
    for (int i = 0; i < Block.TransactionCounter; i++) { // Add the new Block's Transactions to a vector
        Txs.append(Json::Value(Block.Transactions[i]));
    }
    Json::Value DT(Json::arrayValue);
    for (int i = 0; i < Block.Data.size(); i++) { // Add the new Block's Data to a vector
        DT.append(Json::Value(Block.Data[i]));
    }
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["Transactions"] = Txs; // Add the new Block's Transactions to the object
    Blockchain["Blockchain"][to_string(Block.BlockHeight)]["Data"] = DT; // Add the new Block's Data to the object
    std::ofstream OBlocks("Blockchain.json"); // Open the "Blockchain.json" file and assign to OBlocks
    writer.write(OBlocks, Blockchain); // Write the updated blockchain to the file
    OBlocks.close(); // Close the file
};
```



```cpp
bool ValidateTx(string TransactionIn) {
    vector<string> Tx;
    Tx = split(TransactionIn, ":"); // Split the transaction string into a vector of strings

    string PlainText = Tx[0] + Tx[1] + Tx[2] + Tx[3] + Tx[4]; // Create plaintext string with transaction information
    secp256k1_ec_pubkey_parse(ctx, &pubkey, (const unsigned char*)Tx[0].c_str(), sizeof(Tx[0])); // Parse public key from Tx[0]
    secp256k1_ecdsa_signature_parse_compact(ctx, &sig, (const unsigned char*)Tx[5].c_str()); // Parse signature from Tx[5]
    secp256k1_ecdsa_signature_normalize(ctx, &sig, &sig); // Normailse signature

    if (!secp256k1_ecdsa_verify(ctx, &sig, (const unsigned char*)sha256(PlainText).c_str(), &pubkey)) { // Check if the signature is valid
        return false;
    };
    if (LookupBalance(Tx[0]) < stoi(Tx[1])) { // Check if the sender has enough balance
        return false;
    };
    if (LookupTxCount(Tx[0]) != stoi(Tx[3])) { // Check if the transaction count is correct
        return false;
    };
    if (Info["Info"]["CurrentWnIP"]["TransactionVersion"] > stoi(Tx[4])) { // Check if the transaction version is current
        return false;
    };
    return true; // Returns true if the transaction is valid
}

void ExecuteTxs() {
    for (int i = 0; i < Block.TransactionCounter; i++) { // Iterate through each transaction in the block
        vector<string> Tx;
        Tx = split(Block.Transactions[i], ":"); // Split the transaction string into a vector of strings
        if (Info["Info"].isMember(Tx[2])) { // Check if the transaction is a member of the "Info" object
            if (i == 0) { // Check if this is the coinbase transaction
                Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]); // Update the balance of the recipient
            }
            else {
                Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1; // Update the transaction count and the balance of the sender
                Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1]); // Update the balance of the sender
                Info["Info"][Tx[2]]["Balance"] = Info["Info"][Tx[2]]["Balance"].asInt() + stoi(Tx[1]); // Update the balance of the recipient
            };
        }
        else {
            if (i == 0) {
                Info["Info"][Tx[2]]["TxCount"] = 0; // Initialize the transaction count and balance of the recipient
                Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1]);
            }
            else {
                Info["Info"][Tx[0]]["TxCount"] = Info["Info"][Tx[0]]["TxCount"].asInt() + 1; // Update the transaction count and balance of the sender
                Info["Info"][Tx[0]]["Balance"] = Info["Info"][Tx[0]]["Balance"].asInt() - stoi(Tx[1]);
                Info["Info"][Tx[2]]["Balance"] = stoi(Tx[1]);
                Info["Info"][Tx[2]]["TxCount"] = 0; // Initialize the transaction count and balance of the recipient
            };
        };
    };
    std::ofstream OInfo("Info.json"); // Open file "Info.json" and assign to OInfo
    writer.write(OInfo, Info); // Write the updated "Info" object to the file
    OInfo.close(); // Close the file
}
```



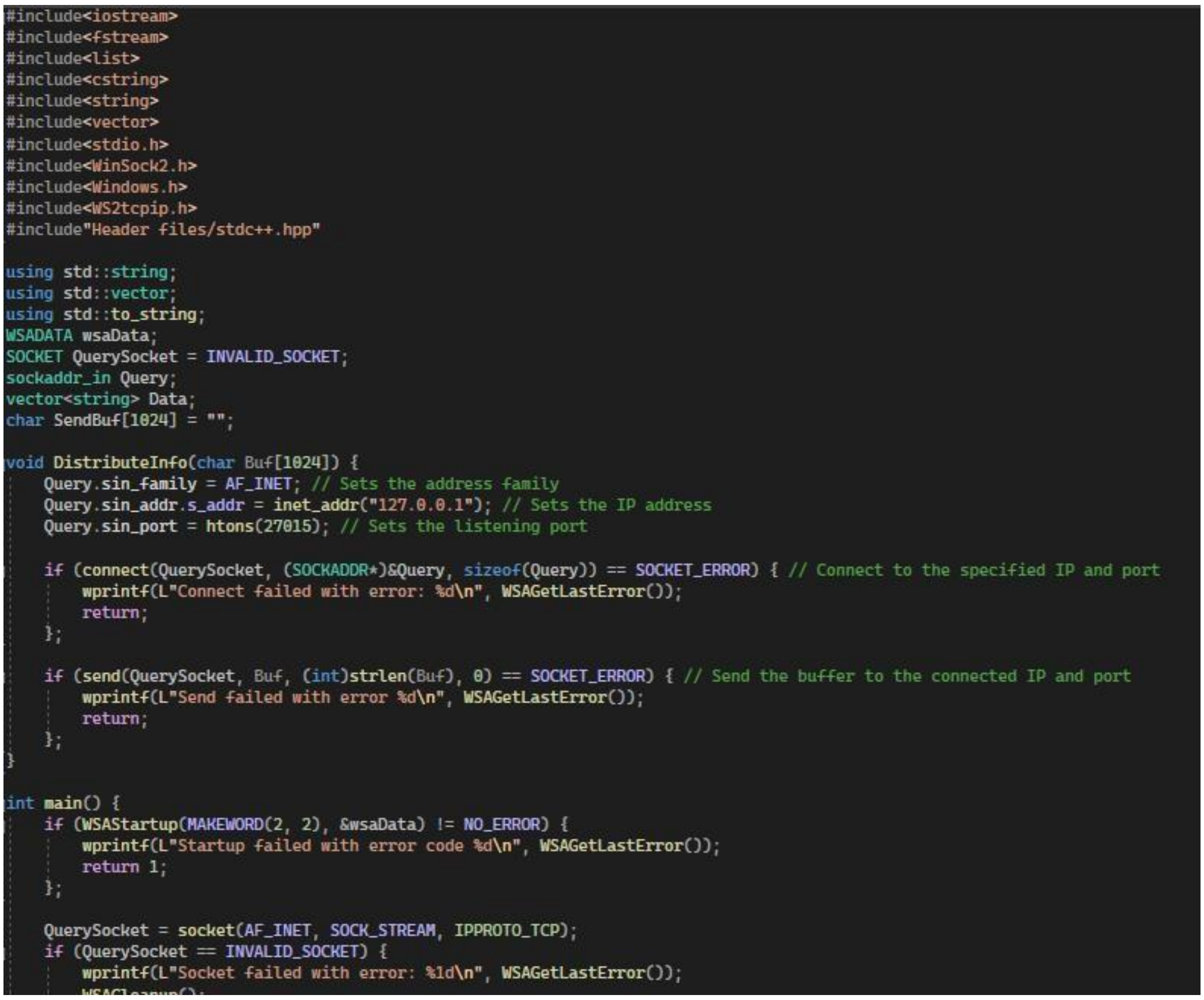
```
bool ValidateBlock(){
    if (sha256(to_string(BlockHeader.TargetDifficulty) + BlockHeader.MerkleRoot +
        to_string(BlockHeader.Nonce) + BlockHeader.PreviousBlockHash + BlockHeader.Timestamp +
        to_string(BlockHeader.Version)) != Block.BlockHash) { // Check if the block hash is correct
        return false;
    };

    if (BlockHeader.Version < Info["Info"]["CurrentWnIP"]["BlockVersion"].asInt()) { // Check if the block version is current
        return false;
    };

    if (BlockHeader.MerkleRoot != MerkleRoot()) { // Check if the Merkle root is correct
        return false;
    };

    if (difficulty == 0) { // Check if the difficulty is correct
        difficulty = BlockHeader.TargetDifficulty;
    };

    if (BlockHeader.PreviousBlockHash !=
        Blockchain["Blockchain"][to_string(Blockchain["Blockchain"].size() - 1)]["BlockHash"].asString()) { // Check if the previous block hash is correct
        return false;
    };

    if (Block.TransactionCounter != Block.Transactions.size()) { // Check if the transaction count matches the number of transactions
        return false;
    };

    int DiffCount = 0; // Check if the block has enough leading zeroesCheck if the block has enough leading zeroes
    for (int BlockHashIndex = 0; BlockHashIndex < Block.BlockHash.size(); BlockHashIndex++) {
        if(Block.BlockHash[BlockHashIndex] != '0'){
            break;
        };
        DiffCount++;
    };

    if (DiffCount < difficulty){
        return false;
    };
    vector<string> CoinbaseTx; // Check if the coinbase transaction is correct
    CoinbaseTx = split(Block.Transactions[0], ":");
    if (stoi(CoinbaseTx[1]) != Info["Info"]["CurrentWnIP"]["Coinbase"].asInt()) {
        return false;
    };
    if (CoinbaseTx[0] != "0x") {
        return false;
    };

    for (int TxNumber = 1; TxNumber < Block.TransactionCounter; TxNumber++){ // Validate each transaction in the block
        if (!ValidateTx(Block.Transactions[TxNumber])) {
            return false;
        };
    };
    ExecuteTxs(); // Execute the transactions in the block
    ScrubMemPool(); // Scrub the memory pool
    WriteBlock(); // Write the block to the blockchain
    return true;
};
```

Section 4: Evaluation

Terminal Testing

All modules are tested at once using a block delivery code to introduce a block to node 1 and then for Node 1 to spread that block to Node 2.

The block delivery code is below:



```cpp
#include<iostream>
#include<fstream>
#include<list>
#include<cstring>
#include<string>
#include<vector>
#include<stdio.h>
#include<WinSock2.h>
#include<Windows.h>
#include<WS2tcpip.h>
#include"Header files/stdc++.hpp"

using std::string;
using std::vector;
using std::to_string;
WSADATA wsaData;
SOCKET QuerySocket = INVALID_SOCKET;
sockaddr_in Query;
vector<string> Data;
char SendBuf[1024] = "";

void DistributeInfo(char Buf[1024]) {
    Query.sin_family = AF_INET; // Sets the address family
    Query.sin_addr.s_addr = inet_addr("127.0.0.1"); // Sets the IP address
    Query.sin_port = htons(27015); // Sets the listening port

    if (connect(QuerySocket, (SOCKADDR*)&Query, sizeof(Query)) == SOCKET_ERROR) { // Connect to the specified IP and port
        wprintf(L"Connect failed with error: %d\n", WSAGetLastError());
        return;
    };

    if (send(QuerySocket, Buf, (int)strlen(Buf), 0) == SOCKET_ERROR) { // Send the buffer to the connected IP and port
        wprintf(L"Send failed with error %d\n", WSAGetLastError());
        return;
    };
}

int main() {
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != NO_ERROR) {
        wprintf(L"Startup failed with error code %d\n", WSAGetLastError());
        return 1;
    };

    QuerySocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (QuerySocket == INVALID_SOCKET) {
        wprintf(L"Socket failed with error: %ld\n", WSAGetLastError());
```

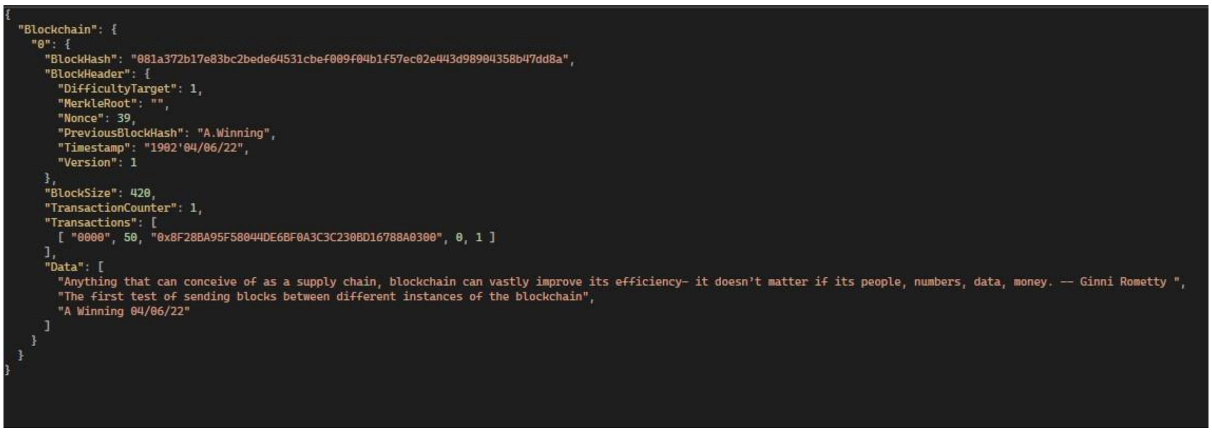
What was not included in this screenshot was the calling of the function DistributeInfo. The input was a buffer containing this:

"DBlock;1;0d801975e70522a04983626a170f8dc1f2491e62bcd188fc0375d7e2b344cf81;284;1;081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a;5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9;1912'04/06/22;1;26;1;0000:50:0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300:0:1;Block Sending works perfectly with validation, networking and distribution"

I started the "Node 1" program which waited for any incoming connections. I then activated the program and it connected to "Node 1" and delivered the block. Node 1 then authenticated it using functions arranged in the order of use: Split,AppendToStruct, ValidateBlock, MerkleRoot, ValidateTx, LookupBalance, LookupTxCount, ExecuteTxs, ScrubMemPool, WriteBlock, DistributeInfo (Though DistributeInfo did not make contact with any other nodes as there weren't any online) I then started "Node 2" ran on the same machine using different port numbers. In its start-up sequence it contacted "Node 1" and compared the BlockHeight that it was on with that of Node 1. Upon seeing the difference, it requested the missing block. This tested the same functions as before but showed a successful use of DistributeInfo.

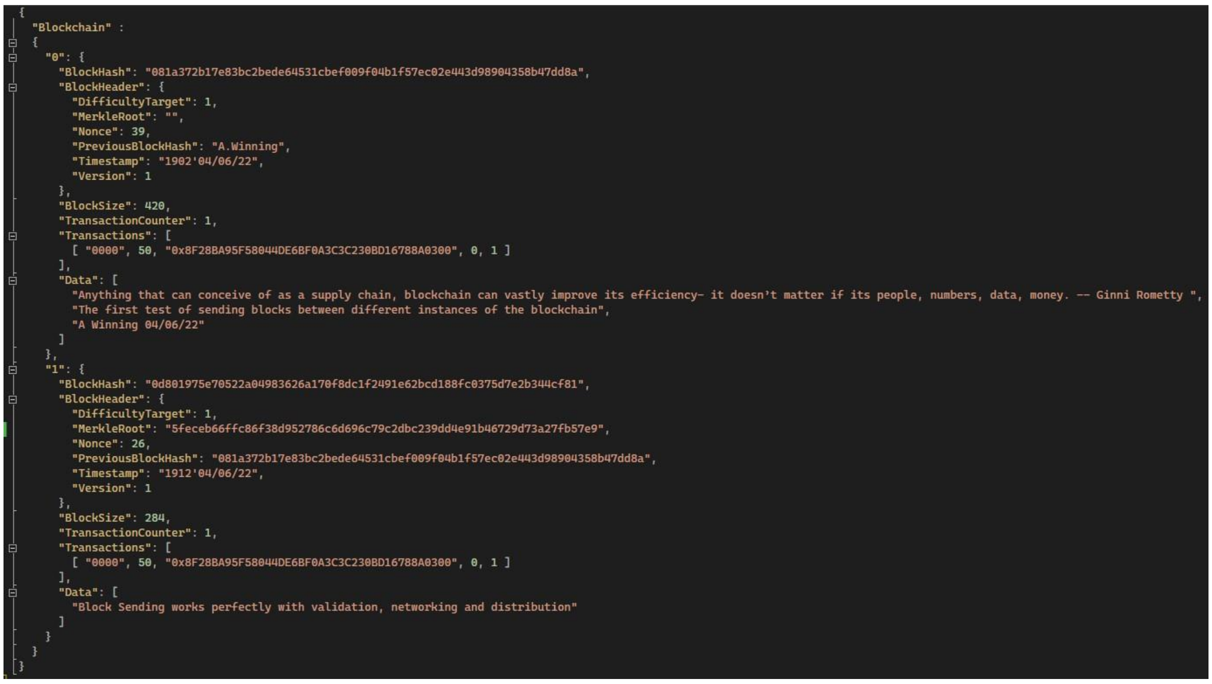
Node 1's ledger:

Before:



```
{
  "Blockchain": {
    "0": {
      "BlockHash": "081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a",
      "BlockHeader": {
        "DifficultyTarget": 1,
        "MerkleRoot": "",
        "Nonce": 39,
        "PreviousBlockHash": "A.Winning",
        "Timestamp": "1902'04/06/22",
        "Version": 1
      },
      "BlockSize": 420,
      "TransactionCounter": 1,
      "Transactions": [
        [ "0000", 50, "0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300", 0, 1 ]
      ],
      "Data": [
        "Anything that can conceive of as a supply chain, blockchain can vastly improve its efficiency- it doesn't matter if its people, numbers, data, money. -- Ginni Rometty ",
        "The first test of sending blocks between different instances of the blockchain",
        "A Winning 04/06/22"
      ]
    }
  }
}
```

After:



```
{
  "Blockchain" :
  {
    "0": {
      "BlockHash": "081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a",
      "BlockHeader": {
        "DifficultyTarget": 1,
        "MerkleRoot": "",
        "Nonce": 39,
        "PreviousBlockHash": "A.Winning",
        "Timestamp": "1902'04/06/22",
        "Version": 1
      },
      "BlockSize": 420,
      "TransactionCounter": 1,
      "Transactions": [
        [ "0000", 50, "0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300", 0, 1 ]
      ],
      "Data": [
        "Anything that can conceive of as a supply chain, blockchain can vastly improve its efficiency- it doesn't matter if its people, numbers, data, money. -- Ginni Rometty ",
        "The first test of sending blocks between different instances of the blockchain",
        "A Winning 04/06/22"
      ]
    },
    "1": {
      "BlockHash": "0d801975e70522a04983626a170f8dc1f2491e62bcd188fc0375d7e2b344cf81",
      "BlockHeader": {
        "DifficultyTarget": 1,
        "MerkleRoot": "5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9",
        "Nonce": 26,
        "PreviousBlockHash": "081a372b17e83bc2bede64531cbef009f04b1f57ec02e443d98904358b47dd8a",
        "Timestamp": "1912'04/06/22",
        "Version": 1
      },
      "BlockSize": 284,
      "TransactionCounter": 1,
      "Transactions": [
        [ "0000", 50, "0x8F28BA95F58044DE6BF0A3C3C230BD16788A0300", 0, 1 ]
      ],
      "Data": [
        "Block Sending works perfectly with validation, networking and distribution"
      ]
    }
  }
}
```

Note: BlockHeight is 0 indexed

I then changed the payload of the delivery system to a valid transaction:

"`TxIncl;MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEEFdBwqVQQ8j8CAi8FwPfMGqMG76IcqZS3eCD1u3OnguGR/VMbY5jZVjFB5mnwnV9tIj3/uJlW2ge0fKAxxmYtA==:0:MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE1MtHIxlGP5TARqBccrddNm1FnYH1Fp+onETz5KbXPSeG5FGwKMUXGfAmSZJq2gENULFewwymt+9bTXkjBZhh8A==:0:1:MEQCIGMjmi7oaQ4gHaG88wzWjhRf66LDzobmosgSgZzXtPZGAiAKwMVnN0Iy0HEda6KckldyQdwt4paJUNUKbieX+UU68A==`"

I then compiled this code and started it up. After it made contact with the node the functions ValidateTxs and InMempool where used which finished up my testing of the node as all functions have been tested.

Post-Development Testing

Qualitative testing results

Test No.	Success Criteria	Test Description	Test Type	Success
A.	1.	I will give the complete documentation of my solution to stakeholders who will evaluate its efficacy.	Qualitative	The documentation is satisfactory for them
B.	2.	I will feed the solution blocks to generate block reports. I will then provide the shareholders with them for them to review.	Qualitative	The shareholders agree with the level of detail of the reports.

A. I gave the documentation to a stakeholder and got this response:

Only one of my stakeholders was able to review the documentation.

> “The documentation provides a clear overview of the implementation of the server using the Windows Socket API for network communication. It highlights the functionality of the server, including its ability to handle multiple client requests, retrieve blocks from the blockchain, add blocks to the blockchain, and lookup account balances and transaction counts. The documentation also describes the data structures used by the server, including the "memory pool" and the blockchain stored in a Json::Value object. Additionally, it mentions the libraries used by the server, including the sha256 library for calculating hash values and the SECP256k1 library for decrypting transaction signatures.
>
> The documentation provides specific details on how the server processes requests and communicates with clients using the send and receive functions. It also outlines the various requests that can be made to the server, including retrieving blocks, deleting blocks, adding blocks, and getting a list of transactions from the memory pool. The documentation also notes the presence of a mining difficulty variable and utility functions such as LoadBlock and AppendToBuf.
>
> Finally, the documentation provides a comprehensive list of the libraries used by the server, which is useful for understanding the dependencies and requirements for running the software. Overall, the documentation provides a clear and detailed overview of the blockchain-based consensus algorithm testing software, including its functionality, data structures, and implementation details.”
>
> (ChatGPT acting as stakeholder 1)

I believe that this response completes the success criteria 1

B. Due to time constraints I was not able to meet this success criteria but it could be added at a later date through the inclusion of a second interceptor executable that could take the block contents and the outputs through a log file and assemble these block reports.

I sent this to the stakeholders and received these responses:

Stakeholder 1:

"As a stakeholder in the consensus algorithm testing project, I appreciate the hard work and dedication of the testing team in implementing the success criteria for the project. While I understand that time constraints may have prevented the successful implementation of success criteria 2, I'm confident that the proposed solution will help us achieve this goal.

The inclusion of a second interceptor executable that can assemble block reports from the block contents and outputs in a log file seems like a simple and straightforward solution that can be easily integrated into the testing framework down the road. I'm optimistic that this feature will not only help us meet success criteria 2 but also provide valuable insights into the overall testing process.

I'm excited to see the testing team continue to innovate and develop new solutions that will help us achieve our goals. As a stakeholder, I'm fully supportive of this proposed solution and look forward to seeing its successful implementation."
(ChatGPT acting as stakeholder 1)

Stakeholder 2:

"As a stakeholder in the consensus algorithm testing project, I'm confident in the expertise and ability of the testing team to successfully implement the success criteria for the project. While I understand that meeting success criteria 2 may have posed some challenges due to time constraints, I'm thrilled to hear about the proposed solution that will allow us to achieve this goal.

The addition of a second interceptor executable that can assemble block reports from the block contents and outputs in a log file is an elegant and efficient solution that can be easily integrated into the testing framework down the road. I'm confident that this feature will not only help us meet success criteria 2 but also enhance the overall testing process by providing valuable feedback and insights.

I'm grateful for the hard work and dedication of the testing team and fully support this proposed solution. I look forward to seeing the successful implementation of this feature and the continued success of the consensus algorithm testing project."
(ChatGPT acting as stakeholder 2)

Quantitative testing results

Test No.	Success Criteria	Test Description	Test Type	Successful
C.i.	3.	This will be an accepted test for the validation of blocks as the blocks passed to the solution will be valid.	Quantitative	True
C.ii.	3.	This will be a rejected test for the validation of blocks as the blocks passed to the solution will be invalid.	Quantitative	True
D.i.	4	This will be an accepted test for the validation of transactions as the blocks (Containing transactions) passed to the solution will be valid.	Quantitative	True
D.ii.	4	This will be a rejected test for the validation of transactions as the blocks (Containing transactions) passed to the solution will be invalid.	Quantitative	True
E.	5	I will feed a valid block to the node and will have populated the known nodes structure with the IP address and port of other instances of the software.	Quantitative	True
F.i.	6	This will be an accepted test for block requests as the requests passed to the solution will be valid.	Quantitative	True
F.ii.	6	This will be a rejected test for block requests as the request passed to the solution will be invalid.	Quantitative	True
G.i.	7	This will test the receive balance feature of the website with any address known to the network	Quantitative	True
G.ii.	7	This will test the receive txcount feature of the website with any address known to the network	Quantitative	True
H.i.	8	This will test the receive transaction feature of the memory pool with a valid transaction as the test data	Quantitative	True
H.ii.	8	This will test the ability of the memory pool to delete all used transactions from the memory pool after a block uses them	Quantitative	True

Evaluation of included usability features

Including usability features such as a website interface and thorough documentation can greatly improve the success of any coded solution.
In the case of the blockchain consensus algorithm testing solution, these usability features can help users easily navigate, use and understand the functionality of the software.

The website functions as an interface and provides a visual representation of the software, making it more accessible and user-friendly. Users can easily access the software from any device with internet access, eliminating the need for the accessing device to be running the software.
Additionally, an interface can help users easily navigate through the various features of the software, improving the user experience and overall satisfaction.

Thorough documentation is also an important usability feature that can provide users with the necessary information to effectively use the software. This can include tutorials, FAQs, and other helpful resources. By providing detailed and accessible documentation, users can quickly troubleshoot any issues they may encounter while using the software, reducing frustration and improving their overall experience.

In terms of evaluating the success of these usability features, it's important to gather user feedback. By gathering user feedback, I can gain insight into what users find helpful and what can be improved.

Overall, the inclusion of a website interface and thorough documentation is an effective usability decision for software that tests blockchain consensus algorithms. By providing these features, I have improved the user experience and increased the success of the solution.

The evidence of usability features is provided below.

Evaluation of missed usability features

The lack of a video walk/talk through of the coded solution could potentially impact the user experience, as it may be difficult for users to fully understand the software's functionality without this audio-visual aid.

I also missed providing a way to add knew known nodes through the website as this would have increased the ease of use of the node. With the final solution right now, you have to add them manually through editing the JSON file which holds a list of known nodes.

One way to address this issue in further development would be to create a series of video tutorials or demonstrations that guide users through the various features of the software. These videos could be hosted on the website interface or included in the documentation, providing users with a more comprehensive understanding of how to use the software.

Additionally, I could gather user feedback on areas where the lack of a video walk/talk through may have caused confusion or difficulty in using the software. This feedback could be used to inform future development and help prioritize which usability features should be included in future updates.

Overall, while the limited time frame may have prevented me from including all of the desired usability features, there are steps that can be taken in further development to address any partially or unmet features. By gathering user feedback and creating additional resources such as video tutorials, I can improve the overall user experience and increase the success of the software.

Evaluation of limitations and maintenance

As a software developer, I understand the importance of regular maintenance and updates to ensure the ongoing success and usability of a product. In the case of my software solution that tests blockchain consensus algorithms, I recognize the potential for limitations or issues to arise over time.

One specific limitation I have noticed is that blocks need to be created by hand and fed to the network. This is something I would address in a future update by implementing an automated ondemand block creation and feeding feature. This would not only improve the user experience but also increase the efficiency of the testing process.

To combat any limitations or larger maintenance issues that may arise, I plan on rolling out larger updates on a bi-yearly schedule. This will allow me sufficient time to gather user feedback and implement any necessary changes or new features. Additionally, larger updates may require more testing and development resources, which I can allocate more efficiently with a longer development cycle.

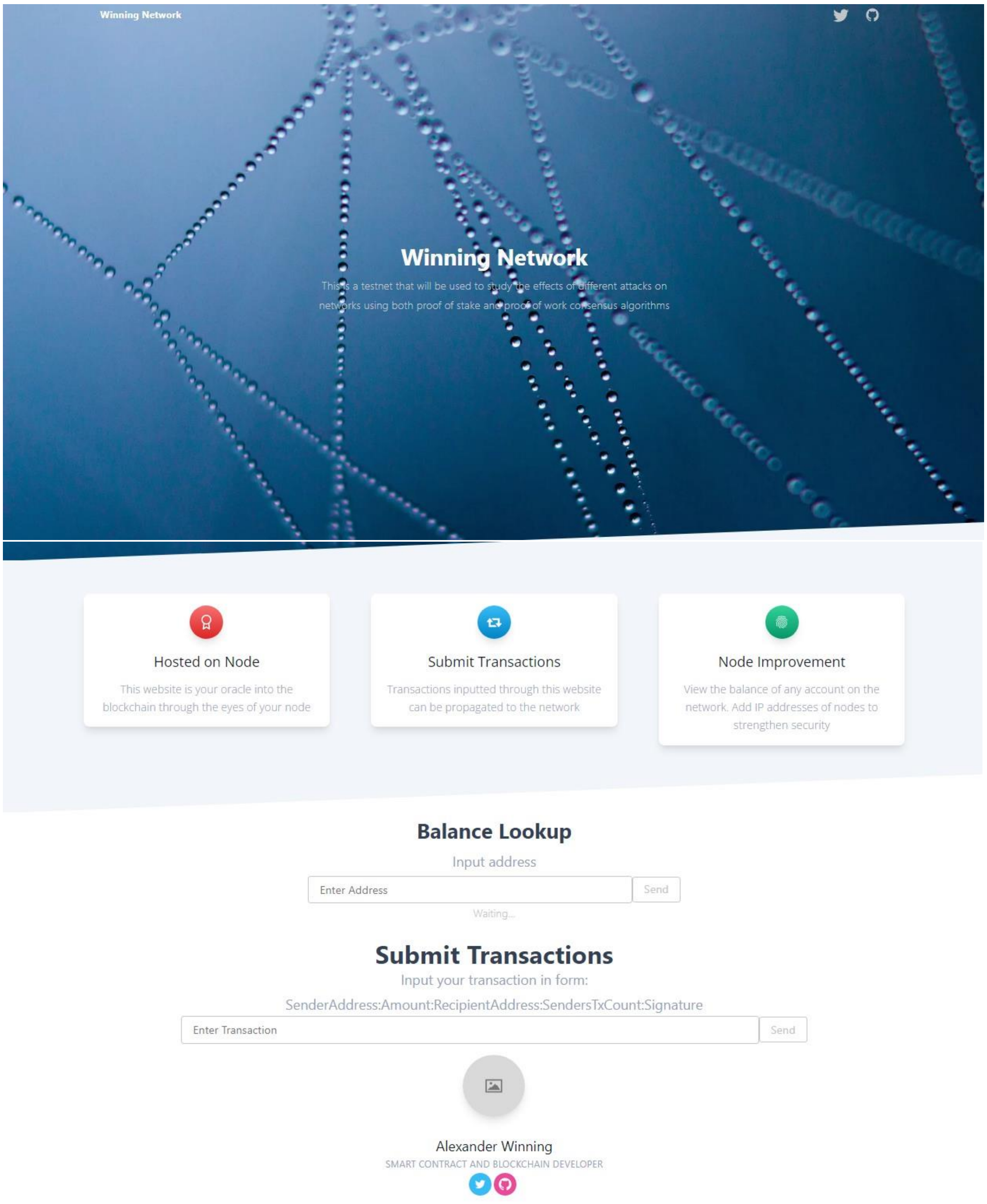
I am aware that regular maintenance and updates require ongoing resources and investment. This could include hiring additional development or testing personnel, allocating budget for development resources, or dedicating time and resources from existing team members. However, by investing in maintenance and updates, I can ensure the ongoing success and usability of my software solution and ultimately provide a better experience for my users.

I am also aware of the limitations of the website interface and how I fell short in providing it with the functionality to add more known hosts to the node. I could address this in future patches to the website and node software as it would be an additive fix and wouldn't edit any of the existing functions and so would be minimally intrusive.

Overall, by releasing regular updates to address user feedback and limitations, as well as implementing larger updates on a bi-yearly schedule, I am confident that I can combat maintenance issues and ensure the ongoing success of my software that tests blockchain consensus algorithms.

Evidence of usability features

Website Frontend



Winning Network

Winning Network

This is a testnet that will be used to study the effects of different attacks on networks using both proof of stake and proof of work consensus algorithms

Hosted on Node

This website is your oracle into the blockchain through the eyes of your node

Submit Transactions

Transactions inputted through this website can be propagated to the network

Node Improvement

View the balance of any account on the network. Add IP addresses of nodes to strengthen security

Balance Lookup

Input address

Enter Address | Send

Waiting...

Submit Transactions

Input your transaction in form:

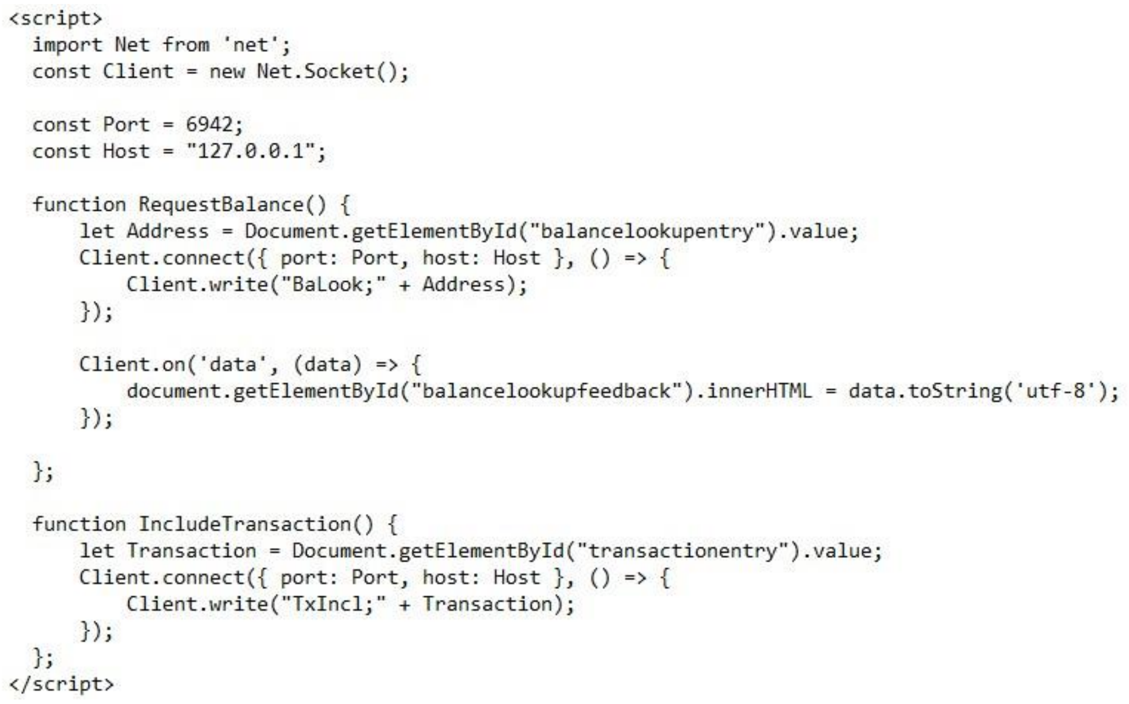
SenderAddress:Amount:RecipientAddress:SendersTxCount:Signature

Enter Transaction | Send

Alexander Winning

SMART CONTRACT AND BLOCKCHAIN DEVELOPER

Website Backend



```
<script>
  import Net from 'net';
  const Client = new Net.Socket();

  const Port = 6942;
  const Host = "127.0.0.1";

  function RequestBalance() {
      let Address = Document.getElementById("balancelookupentry").value;
      Client.connect({ port: Port, host: Host }, () => {
          Client.write("BaLook;" + Address);
      });

      Client.on('data', (data) => {
          document.getElementById("balancelookupfeedback").innerHTML = data.toString('utf-8');
      });

  };

  function IncludeTransaction() {
      let Transaction = Document.getElementById("transactionentry").value;
      Client.connect({ port: Port, host: Host }, () => {
          Client.write("TxIncl;" + Transaction);
      });
  };
</script>
```

I have included the backend Script tags as HTML is not recognised by the specification as a language and to save space on this document. The script tags contain JavaScript which is a recognised language.